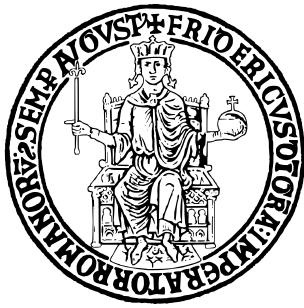


Università degli Studi di Napoli Federico II



SCUOLA POLITECNICA E DELLE SCIENZE DI BASE

DIPARTIMENTO DI INGEGNERIA INDUSTRIALE

Corso di Laurea Magistrale in Ingegneria Aerospaziale

Classe delle Lauree Magistrali in Ingegneria Industriale (LM-20)

ELABORATO DI

Dinamica e Simulazione di Volo

Prof. Agostino DE MARCO

Candidato

Giovanni Battista VARRIALE
M53001880

Anno Accademico 2023–2024

Indice

1 Studio delle caratteristiche aerodinamiche di un velivolo con DATCOM	6
1.1 Introduzione	6
1.2 Geometria	6
1.2.1 Input Datcom del P2010	11
1.3 Caratteristiche Aerodinamiche	12
1.4 Modello 3D	21
2 Equazioni generali del moto di un velivolo	33
2.1 Intro	33
2.2 Esercizio 6.5	37
2.3 Modello Simulink	39
2.4 Risultati del modello	49
2.4.1 Traiettoria del Centro di Massa	54
2.5 Risultati di un secondo modello	61
2.6 Risultati del modello	63
2.6.1 Traiettoria del Centro di Massa	68
3 Moto longitudinal-simmetrico a comandi bloccati	70
3.1 Intro	70
3.1.1 Moto a 3 Gradi di Libertà	72
3.2 Esercizio 1	73
3.2.1 Condizioni Iniziali	73
3.2.2 Risultati	77
3.3 Esercizio 2 (7.6-7.7-7.11) - Ricerca delle condizioni di Trim per il moto a 3 DoF	79
3.3.1 Risultati	84
3.4 Esercizio 3 (7.7) - Velocità di trim al variare dell'angolo iniziale dello stabilitore	88
3.4.1 Risultati	90
3.5 Esercizio 4 - Dinamica del Volo in Atmosfera Turbolenta	91
3.5.1 Gust	91
3.5.2 Gust e δ_e	103
4 Equazioni del moto delle superfici di governo	108
4.1 Intro	108
4.2 Input Esercizi	112
4.3 Esercizio 1 (11.1)	113
4.3.1 Risultati	119
4.4 Esercizio 2 (11.2)	122
4.4.1 Risultati	128
4.5 Esercizio 3 (11.3)	131
4.5.1 Risultati	132
4.6 Esercizio 4 (11.4)	135
4.6.1 Risultati	142
4.7 Esercizio 5 (11.6)	145

5 Piccole perturbazioni nel moto di un velivolo	154
5.1 Introduzione	154
5.1.1 Condizioni di Trim	156
5.1.2 Azioni aerodinamiche linearizzate	157
5.1.3 Azioni Propulsive Linearizzate	157
5.1.4 Sistema LTI	158
5.1.5 Modi di risposta dei velivoli	158
5.1.6 Dinamica Longitudinale	159
5.2 Esercizio 1 (16.1-2)	162
5.2.1 Input	162
5.2.2 Risultati	168
5.3 Esercizio 2 (16.3)	174
5.3.1 Risultati	177
5.4 Esercizio 3	179
5.4.1 Risultati	179

Elenco delle figure

1.1	Viste Generali P2010	8
1.2	P2010 TopView	10
1.3	P2010 SideView	11
1.4	Curve Caratteristiche Tecnam P2010	18
1.5	C_L al variare dell'angolo δ_e e δ_f	19
1.6	C_D al variare dell'angolo δ_e e δ_f	19
1.7	C_M al variare dell'angolo δ_e e δ_f	20
1.8	Carichi di Equilibrio	20
1.9	Modello 3D	32
2.1	Terna	34
2.2	Tecnam P2010	37
2.3	Modello Simulink	40
2.4	Sottosistema Simulink 1	41
2.5	Sottosistema Simulink 2	42
2.6	Sottosistema Simulink 3	42
2.7	Sottosistema Simulink 4	43
2.8	Sottosistema Simulink 5	43
2.9	Sottosistema Simulink 6	44
2.10	Sottosistema Simulink 7	45
2.11	Sottosistema Simulink 8	46
2.12	Sottosistema Simulink 9	46
2.13	Sottosistema Simulink 10	47
2.14	Sottosistema Simulink 11	47
2.15	Sottosistema Simulink 12	47
2.16	Sottosistema Simulink 13	48
2.17	Sottosistema Simulink 14	48
2.18	Velocità, γ e ψ comandate	49
2.19	Massa di carburante consumato	50
2.20	Spinta, Resistenza e Velocità	51
2.21	Portanza, α e γ , rateo di salita	52
2.22	Angolo di bank e heading	53
2.23	Storie temporali delle coordinate del CG	59
2.24	Visualizzazione 3D delle storie temporali delle CG	60
2.25	Sottosistema Simulink per la spinta variabile con la velocità	62
2.26	Velocità, γ e ψ comandate 2	63
2.27	Massa di carburante consumato 2	64
2.28	Spinta, Resistenza e Velocità 2	65
2.29	Portanza, α e γ , rateo di salita 2	66
2.30	Angolo di bank e heading 2	67
2.31	Storie temporali delle coordinate del CG 2	68
2.32	Visualizzazione 3D delle storie temporali delle CG 2	69
3.1	Legame tra le componenti del vettore velocità del vento relativo negli assi velivolo ed angoli aerodinamici (si è posto per semplicità $\alpha \equiv \alpha_B$)	71
3.2	Dati in INPUT esercizio 7.1	74
3.3	Vista dell'esercizio 7.1 in APP	77
3.4	Storie temporali dell'HARV - Esercizio 7.1	78

3.5	Dati in Input - Esercizio 2	79
3.6	Storie temporali	85
3.7	Fattori di carico	86
3.8	Comandi	87
3.9	Input dell'esercizio 3	88
3.10	Risultati dell'esercizio 3	90
3.11	Gust - Velocità, q, coordinate del CG, θ	98
3.12	Gust - C_M stazionario e dinamico	99
3.13	Gust - Vento e comandi del δ_e	100
3.14	Gust - Angolo di attacco, fattore di carico, frequenza ridotta (numero di Strouhal = $\dot{\alpha}\bar{c}/2V$), rapporto fra portanza instazionaria e stazionaria	101
3.15	Gust - C_L stazionario e instazionario	102
3.16	Gust e δ_e - Velocità, q, coordinate del CG, θ	103
3.17	Gust e δ_e - C_M stazionario e dinamico	104
3.18	Gust e δ_e - Vento e comandi del δ_e	105
3.19	Gust e δ_e - Angolo di attacco, fattore di carico, frequenza ridotta (numero di Strouhal = $\dot{\alpha}\bar{c}/2V$), rapporto fra portanza instazionaria e stazionaria	106
3.20	Gust e δ_e - C_L stazionario e instazionario	107
4.1	Assi superficie di governo, centro di massa G_{CS} , accelerazione a_C del polo C e velocità angolare istantanea Ω_{CS} , rispetto al riferimento inerziale. Il vettore $\omega_{CS} = \dot{\delta}_{CSI_c}$ è la velocità angolare istantanea dell'organo di comando rispetto al velivolo	108
4.2	Accelerazioni lineari ed angolari nel generico istante di un moto longitudinal-simmetrico. Schema per il calcolo di $a_{G_{z_B}} - g_{z_B}$	111
4.3	Valori in Input	112
4.4	Esercizio 1 - Leggi di comando, angolo di deflessione, fattore di carico e accelerazione angolare	119
4.5	Esercizio 1 - Storie temporali delle variabili di stato	120
4.6	Esercizio 1 - Storie temporali del momento di cerniera	121
4.7	Esercizio 2 - Leggi di comando, angolo di deflessione, fattore di carico e accelerazione angolare	128
4.8	Esercizio 2 - Storie temporali delle variabili di stato	129
4.9	Esercizio 2 - Storie temporali del momento di cerniera	130
4.10	Esercizio 3 - Leggi di comando, angolo di deflessione, fattore di carico e accelerazione angolare	132
4.11	Esercizio 3 - Storie temporali delle variabili di stato	133
4.12	Esercizio 3 - Storie temporali del momento di cerniera	134
4.13	Esercizio 4 - Leggi di comando, angolo di deflessione, fattore di carico e accelerazione angolare	142
4.14	Esercizio 4 - Storie temporali delle variabili di stato	143
4.15	Esercizio 4 - Storie temporali del momento di cerniera	144
4.16	Esercizio 5 - Leggi di comando, angolo di deflessione, fattore di carico e accelerazione angolare	151
4.17	Esercizio 5 - Storie temporali delle variabili di stato	152
4.18	Esercizio 5 - Storie temporali del momento di cerniera	153
5.1	Vista del capitolo 16 in APP	162
5.2	Esercizio 1 - Configurazione 2	169
5.3	Esercizio 1 - Configurazione 5	170
5.4	Esercizio 1 - Configurazione 7	171
5.5	Esercizio 1 - Configurazione 9	172
5.6	Esercizio 1 - Configurazione 10	173
5.7	Esercizio 2 - Input	174
5.8	Esercizio 2 - Configurazione 2	178
5.9	Esercizio 3	179

Elenco delle tabelle

1.1	Specifiche tecniche	7
2.1	Modello Atmosferico ISA	37
2.2	Dati Velivolo - Tecnam P2010	37
2.3	Valori comandati in volo	38
5.1	Derivate di stabilità dimensionali del moto longitudinale	160
5.2	Caratteristiche geometriche Boeing 747	162
5.3	Caratteristiche inerziali e derivate aerodinamiche longitudinali di un Boeing 747 in diverse condizioni di volo nominali. Gli angoli di attacco sono misurati rispetto alla retta di riferimento della fusoliera e le derivate angolari in rad^{-1}	167
5.4	Confronto tra i dati ottenuti nelle varie configurazioni	168

Capitolo 1

Studio delle caratteristiche aerodinamiche di un velivolo con DATCOM

1.1 Introduzione

Lo scopo di questa esercitazione è il calcolo delle caratteristiche aerodinamiche del velivolo Tecnam P2010 tramite il software Digital DATCOM. Per tale scopo si è partiti da dati del velivolo presenti sul sito del produttore. Per i dati mancanti riguardanti la geometria si è ricorso alle viste dell'aereo.

Una volta trovata la geometria del velivolo, esso viene modellato su DATCOM e vengono mostrate le principali caratteristiche aerodinamiche.

Infine si procede con un codice a trasformare gli input DATCOM in un modello 3D, in formato .stl (Figura 1.9). Tale modello del velivolo verrà utilizzato nei capitoli successivi per rappresentazioni grafiche.

1.2 Geometria

Nella seguente sottosezione analizzeremo le scelte fatte per la geometria del velivolo, necessaria nelle fasi successive dell'elaborato. In Tabella 1.1 sono presenti i dati ricavati dal sito del produttore.

Lunghezza	$L = 7.92 \text{ m}$
Altezza	$H = 2.64 \text{ m}$
Apertura Alare	$b = 10.3 \text{ m}$
Posizione del Controvento	a 2.15m dalla radice
Superficie Alare	$S = 13.9 \text{ m}^2$
Corda Alare Media Geometrica	$c = 1.35 \text{ m}$
Profilo Alare	<i>NACA 63₁A012</i>
Corda alla Radice	$c = 1.3 \text{ m}$ 0-20%
Corda nella zona "rettangolare"	$c = 1.4 \text{ m}$ 20-70%
Corda all'estremità	$c = 1.1 \text{ m}$ 70-100%
Corda Media Aerodinamica	$c_{ma} = 1.29 \text{ m}$
Rapporto di Rastremazione 1	$\lambda_1 = 1.41$
Rapporto di Rastremazione 2	$\lambda_2 = 0.79$
Fattore di Oswald	$e = 0.9$
Allungamento alare	$\mathcal{R} = 7.63$
Potenza	$P = 170 \text{ HP}$
Peso a vuoto	$W = 820 \text{ kg}$
Peso massimo al decollo	$W_{TO} = 1200 \text{ kg}$
Carburante	Diesel (EN 590)
Capacità carburante	240 <i>lt</i>
Peso carburante	$W_{fuel} = 200 \text{ kg}$
Carico alare	$\frac{W_{TO}}{S} = 86.33 \frac{\text{kg}}{\text{m}^2}$
Velocità di stallo	$V_S = 98 \frac{\text{km}}{\text{h}}$
Velocità di crociera massima	$V_{C,Max} = 259 \frac{\text{km}}{\text{h}}$
Velocità di crociera al 65% di potenza	$V_C = 237 \frac{\text{km}}{\text{h}}$
Cl_α 2D	$Cl_\alpha = 2\pi \text{ rad}^{-1}$
CL_α 3D	$CL_\alpha = 4.87 \text{ rad}^{-1}$
CL 3D massimo	$CL_{Max} = 1.792 \text{ rad}^{-1}$
CL 3D massimo in volo rovescio	$CL_{Max,r} = -0.717 \text{ rad}^{-1}$

Tabella 1.1: Specifiche tecniche

Nel listato di seguito viene mostrato come si ricava la geometria da visualizzare tramite matlab. La definizione per punti della fusoliera è una approssimazione necessaria a causa del limite di punti in input per la stessa in DATCOM (20 punti).

I punti per la fusoliera sono stati trovati in maniera non adimensionale, per cui è necessario scalarli alle grandezze reali. Procedimenti analoghi sono stati utilizzati per il resto delle componenti del velivolo, per entrambe le viste.

```

1 % P2010   DSV
2 %% Side View
3 clc; clear; close all;
4 x = [0 50 60 100 110 160 200 240 256 320 385 420 480 550 620 665 780 800];
5 zU = [165 147 146 144 140 130 110 100 95 90 92 100 120 140 150 155 158 165];
6 zL = [165 180 191 215 225 230 232 232 232 232 230 225 223 216 210 208 190 165];
7 R = [71 46 26 27 20 19 14 10 8 7 6 8 11 18 29 40 60 71];
8 % la lunghezza x=800 è 7.6m
9 % devo scalare
10 FdS = 7.6/800; % fattore di scala
11 x = x*FdS;
12 zU = (-zU+zU(1))*FdS;
13 zL = (-zL+zL(1))*FdS;
14 R = -(R-R(1))*FdS;
15
16 side=figure('NumberTitle', 'off', 'Name', 'Side View');
17 plot(x,zU,'-c','LineWidth',2); hold on;
18 plot(x,zL,'-c','LineWidth',2);
19 axis([0 (x(end)+0.5) -(x(end)+0.5)/2 (x(end)+0.5)/2]); grid on;
20 axis equal;
21 % piano di coda
22 xv=5.83;
23 h = 1.47; % height
24 a = 0.52; % top side
25 b = 1.55; % base
26 coda=b+xv;
27 A = [0+xv 0] ;
28 B = [b+xv 0] ;
29 C = [b+xv h] ;
30 D = [b+a+xv h] ;
31 coor = [A ; C; D; B; A] ;

```

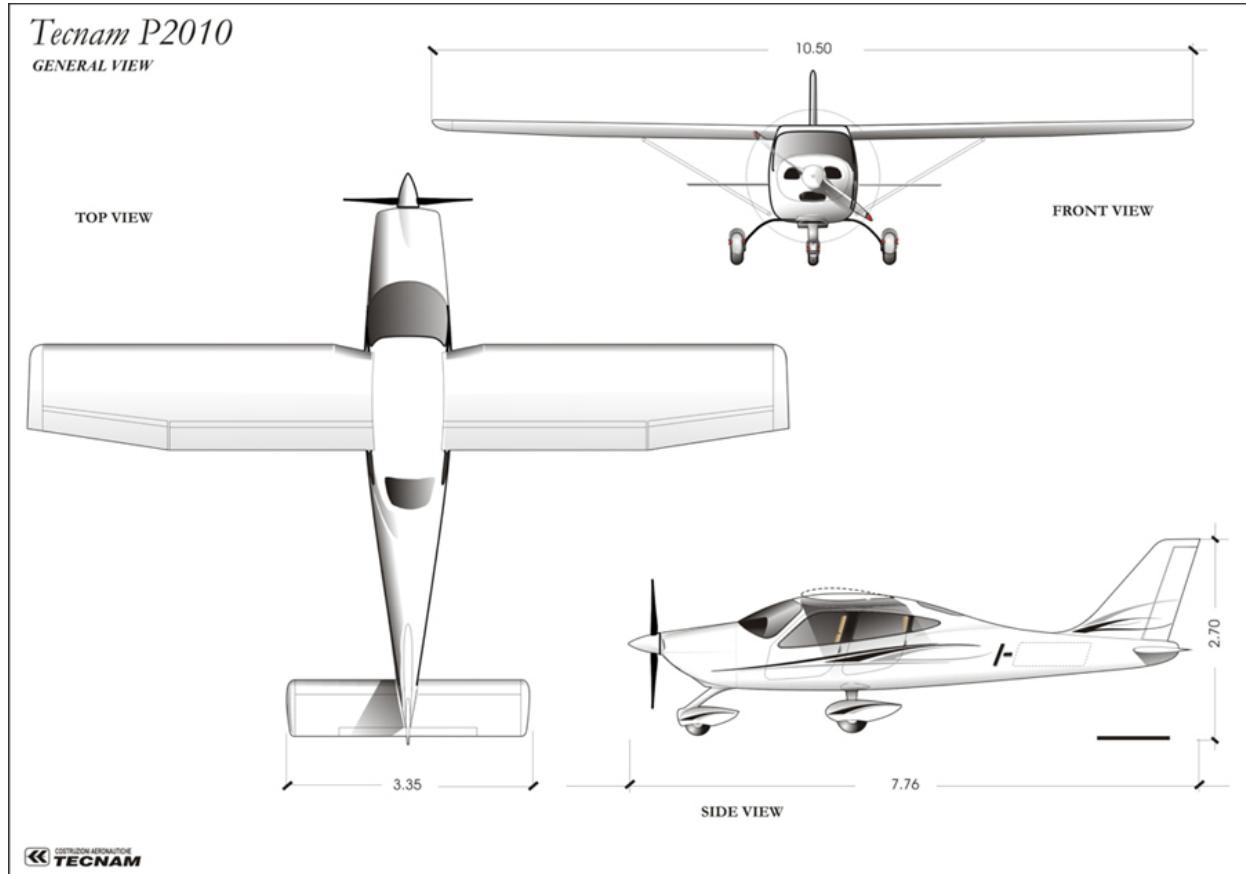


Figura 1.1: Viste Generali P2010

```

32 plot(coor(:,1), coor(:,2),'Color','#0072BD','LineWidth',1.2)
33 % rudder
34 hrud=0.37;
35 zrud=0.2;
36 R1=[D(1)-hrud D(2)];
37 R2=[B(1)-hrud B(2)];
38 coor =[R1;R2];
39 plot(coor(:,1), coor(:,2),'--','Color','#0072BD','LineWidth',1.2)
40 % ala
41 c=1.4;
42 xa=[0 0.5 0.75 1.25 2.5 5.0 7.5 10 15 20 25 30 35 40 45 50 55 60 65 70 75 80 85 90 95 100];
43 ya=[0 0.973 1.173 1.492 2.078 2.895 3.504 3.994 4.747 5.287 5.664 5.901 5.995 5.957 5.792 5.517 5.148 4.7 4.186
     3.621 3.026 2.426 1.826 1.225 0.625 0.025];
44
45 xg=2.36; yg=0.739;
46 xa=xa.*c/100+xg;
47 yau=ya.*c/100+yg;
48 yal=-ya.*c/100+yg;
49 plot(xa,yau,'-b','LineWidth',2); hold on;
50 plot(xa,yal,'-b','LineWidth',2); hold on;
51
52 xlabel('X (m)'); ylabel('Z (m)');
53 title("P2010      Side View");
54
55 % piano orizzontale
56 horiz=readmatrix('XYZ1.xls');
57 horiz=horiz(:,2); horiz=flip(horiz(1:99));
58 xx=linspace(0,0.76,99);
59 xh=6.87;
60 xx=xx+xh;
61 plot(xx,horiz,'-b','LineWidth',2); hold on;
62 plot(xx,-horiz,'-b','LineWidth',2); hold on;
63
64 % elica
65 loc=0.37;
66 r=0.89;
67 A = [loc r];
68 B = [loc -r];
69 AB = [A; B];
70 plot(AB(:,1), AB(:,2),'Color',"red",'LineWidth',1.7)
71
72 XCG=2.36+1.4/4; ZCG=0.6;

```

```

73 plot(XCG,ZCG,'.','.','Color', "#D95319",'Marker','hexagram','MarkerSize',7,'LineWidth',2); hold on;
75
76 pos1 = get(gcf,'Position'); % get position of Figure(1)
77 set(gcf,'Position', pos1 + [0,-pos1(2)+50,0,0]) % Shift position of Figure(1)
78 %% Top view
79 top=figure('NumberTitle', 'off', 'Name', 'Top View');
80 % body
81 plot(x,R,'-c','LineWidth',2); hold on;
82 % elica
83 A = [loc r] ;
84 B = [loc O] ;
85 AB = [A; B];
86 plot(AB(:,1), AB(:,2),'Color',"red",'LineWidth',1.7)
87 % ala
88 wtip=1.18;
89 b=10.5;
90 A = [xg 0] ;
91 B = [xg+c 0] ;
92 C = [xg b/2] ;
93 D = [xg+wtip b/2] ;
94 E = [xg+c (b/2)*0.6] ;
95 coor = [ B; E; D; C; A];
96 plot(coor(:,1), coor(:,2),'Color',"blue",'LineWidth',1.2); hold on;
97 % flap
98 hflap=0.37;
99 yflap=0.72;
100 F1=[xg+c-0.37 yflap];
101 F2=[xg+c yflap];
102 F3=E;
103 F4=[E(1)-hflap E(2)];
104 coor = [ F1; F2; F3; F4; F1] ;
105 plot(coor(:,1), coor(:,2),'--','Color',"blue",'LineWidth',1.0); hold on;
106 % aileron
107 hail=0.3;
108 A1=F4;
109 A2=F3;
110 A3=D;
111 A4=[D(1)-hail D(2)];
112 coor = [ A2; A3; A4; A1] ;
113 plot(coor(:,1), coor(:,2),'--','Color',"blue",'LineWidth',1.0); hold on;
114 % piano di coda
115 ch=0.76;
116 bh=3.35;
117 A = [xh 0] ;
118 B = [xh+ch 0] ;
119 C = [xh+ch bh/2] ;
120 D = [xh bh/2] ;
121 coor = [ B; C; D; A] ;
122 plot(coor(:,1), coor(:,2),'Color',"blue",'LineWidth',1.2); hold on;
123
124 title("P2010      Top View");
125 xlabel('X (m)'); ylabel('Y (m)');
126 axis([0 (x(end)+0.5) -0.5 (x(end))]); grid on;
127 axis equal;
128
129 plot(XCG,O,'.','.','Color', "#D95319",'Marker','hexagram','MarkerSize',7,'LineWidth',2); hold on;
130 % equilibratore
131 hflap=0.37/2;
132 yflap=0.3;
133 F1=[B(1) yflap];
134 F2=[B(1) C(2)-yflap/4];
135 F3=[B(1)-hflap F2(2)];
136 F4=[B(1)-hflap F1(2)];
137 coor = [ F1; F2; F3; F4; F1] ;
138 plot(coor(:,1), coor(:,2),'--','Color',"blue",'LineWidth',1.0); hold on;
139
140 % piano verticale
141 vert=readmatrix('codal1.xls');
142 vert=vert(:,2); vert=flip(vert(1:100));
143 xx=linspace(0,1.55,100);
144 xv=5.83;
145 xx=xx+xv;
146 [Y1,n]=max(abs(vert));
147 X1=[xx(n), Y1];
148 plot(xx,vert,'.','.','Color',"#0072BD",'LineWidth',1);hold on;
149 plot(xx,-vert,'.','.','Color',"#0072BD",'LineWidth',1);hold on;
150
151
152 xx=linspace(0,0.52,100);
153 xv=coda;
154 xx=xx+xv;
155 [Y2,n]=max(abs(vert));
156 X2=[xx(n), Y2];
157 coor = [ X1;X2] ;
158 plot(coor(:,1), coor(:,2),'Color',"#0072BD",'LineWidth',1.2); hold on;
159 plot(coor(:,1), -coor(:,2),'Color',"#0072BD",'LineWidth',1.2); hold on;
160 plot(xx,vert,'Color',"#0072BD",'LineWidth',1);hold on;
161 plot(xx,-vert,'Color',"#0072BD",'LineWidth',1);hold on;

```

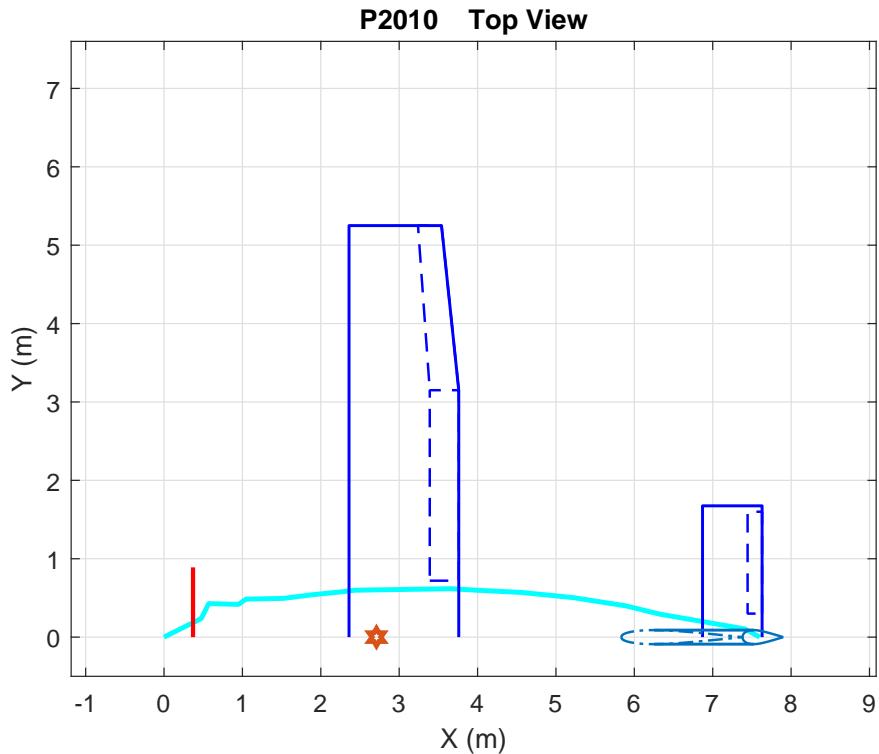


Figura 1.2: P2010 TopView

```

162
163
164
165 set(gcf,'Position', get(gcf,'Position') + [0,0,0,0]); % When Figure(2) is not the same size as Figure(1)
166 pos2 = get(gcf,'Position'); % get position of Figure(2)
167 set(gcf,'Position', pos2 + [0,-8,0,0]) % Shift position of Figure(2)
168
169 exportgraphics(side,'sideview.eps')
170 exportgraphics(top,'topview.eps')
171
172
173
174 %clc; clear;

```

Nelle figure 1.2 e 1.3 vengono mostrati i risultati del precedente listato. Le ali e il piano di coda sono in blu scuro, la fusoliera celeste, il piano verticale azzurro e l'elica in rosso. Il centro di massa è rappresentato dal simbolo arancione. Le superfici mobili sono evidenziate con le linee tratteggiate. Gli unici elementi esclusi sono l'asta di controvento e i carrelli.

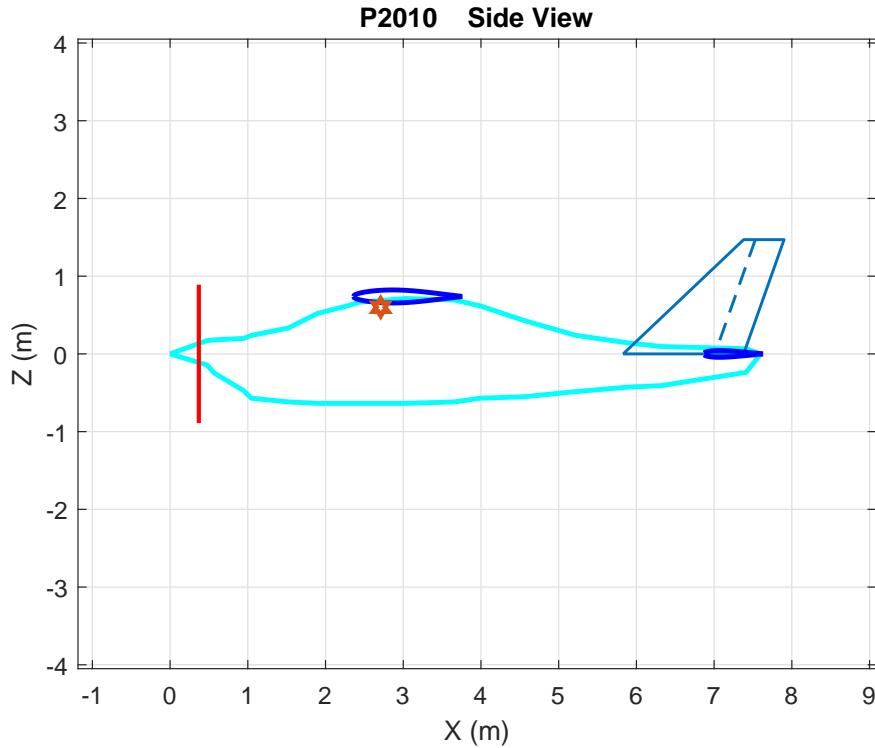


Figura 1.3: P2010 SideView

1.2.1 Input Datcom del P2010

Grazie ai dati della geometria calcolati è ora possibile creare il modello DATCOM del velivolo. Nel successivo listato viene presentato il file .dcm necessario all'analisi delle caratteristiche aerodinamiche del P2010. Per differenziare i vari casi vengono utilizzate diverse BUILD (in totale 3), necessarie per il calcolo dei successivi grafici in MATLAB.

```

1 DIM M
2 DERIV DEG
3 *DAMP
4 *PART
5 *DUMP IOM
6 BUILD
7
8
9 $FLTCON WT=1808.0, LOOP=2.0,
10   NMACH=1.0, MACH(1)=0.3,
11   NALT=1.0, ALT(1)=3000.0,
12   NALPHA=20.0,
13   ALSCHD(1)= -16.0, -8.0, -6.0, -4.0, -2.0, 0.0, 2.0, 4.0, 8.0, 9.0,
14     10.0, 12.0, 13.0, 14.0, 15.0, 16.0, 17.0, 18.0, 19.0, 20.0,
15   STMACH=0.6, TSMACH=1.4, TR=1.0$ 
16
17 $OPTINS SREF=13.9, CBARR=1.29, BLREF=10.63, ROUGFC=0.25E-3$
18
19 $SYNTHS XCG=2.71, ZCG=0.6,
20   XW=2.36, ZW=0.739, ALIW=1.5,
21   XH=6.87, ZH=0.01, ALIH=0.0,
22   XV=5.83, ZV=0.,
23   SCALE=1.0, VERTUP=.TRUE.$
24
25 ****
26
27 $WGPLNF CHRDR=1.4,     CHRDTP=1.28,     CHRDBP=1.4,
28   SSPNOP=2.1,
29     SSPN=5.25,    SSPNE=5.0,
30   SAVSI=0.0,    SAVSO=0.,
31   CHSTAT=0.0,   DHDADO=1.0,  TWISTA=0.,
32   TYPE=1.0$ 
33
34 NACA W 6 63A012
35
36 $ASYFLP STYPE=4.0, NDELTA=9.0,
37   DELTAL(1)=-32.0,-20.0,-10.0,-5.0,0.0,5.0,10.0,20.0,32.0,
38   DELTAR(1)=32.0,20.0,10.0,5.0,0.0,-5.0,-10.0,-20.0,-32.0,
39   SPANFI=3.15, SPANFO=5.25,
40   CHRDFI=0.37, CHRDFO=0.3$
```

```

41 CASEID Wing+Ailerons
42 NEXT CASE
43
44 $SYMF LP FTYP E=1.0, NDELTA=9.0,
45     DELTA(1)=0.0,5.0,10.0,15.0,20.0,25.0,30.0,35.0,40.0,
46     CHRDFI=0.37, CHRDFO=0.37,
47     SPANFI=0.72, SPANFO=3.15,
48     NTYP E=1.0$
49
50 CASEID Wing+Flaps
51 NEXT CASE
52 $BODY NX=17.0,
53     X(1)=0.4,0.5,0.6,1.0,1.5,1.9,2.3,2.4,3.0,3.65,4.0,4.56,5.2,5.9,6.3,7.4,7.6,
54     R(1)=0.,0.2,0.4,0.45,0.5,0.5,0.6,0.6,0.6,0.62,0.6,0.57,0.5,0.4,0.3,0.1,0.0,
55     ZU(1)=0.,0.17,0.18,0.24,0.33,0.5,0.6,0.7,0.7,0.7,0.6,0.4,0.2,0.1,0.1,0.,0.,
56     ZL(1)=0.,-1.,-2.,-6.,-6.,-6.,-6.,-6.,-6.,-6.,-5.,-5.,-4.,-4.,-2.,0.,
57     BNOSE=1.0, BLN=8.8,
58     BTAIL=1.0, BLA=2.9,
59     ITYP E=1.0, METHOD=1.0$
60
61 $HTPLNF CHRDR=0.76, CHRDP T=0.76,
62     SSPN=1.68, SSPNE=1.6,
63     SAVSI=0.0,
64     CHSTAT=0.0, TWISTA=0.0,
65     DHDADI=0.0,
66     TYPE=1.0$
67
68 NACA H 4 0012
69
70
71 $SYMF LP FTYP E=1.0, NDELTA=9.0,
72     DELTA(1)=-20.,-15.,-10.,-5.,0.,5.,10.,15.,20.,
73     CHRDFI=0.19, CHRDFO=0.19,
74     SPANFI=0.3, SPANFO=1.6,
75     NTYP E=1.0$
76
77 *CASEID Wing+Body+Htail+Elev
78
79
80 $VTPLNF CHRDTP=0.52, SSPNE=1.38, SSPN=1.47, CHRDR=1.55,
81     SAVSI=45., CHSTAT=0.25, TYPE=1.0$
82
83
84 NACA V 4 0012
85
86 NACA V 4 0012
87 NACA F 4 0012
88
89 *CASEID Wing+Body+Htail+Vtail
90
91
92 $PROPWR NENGSP=1., THSTCP=0.,
93     PHALOC=0.37, PHVLOC=0., PRPRAD=0.89,
94     ENGFCT=0.8, NOPBPE=2.0,
95     CROT=.FALSE.$
96
97
98 CASEID Wing+Body+Htail+Vtail+Engines

```

1.3 Caratteristiche Aerodinamiche

Il file .dcm viene processato da DATCOM, che da come output un file .out. Tale file viene dato in input a MATLAB tramite la funzione datcomimport, che lo trasforma in una struttura dati in formato .mat. Tramite il successivo codice MATLAB è dunque possibile graficare le caratteristiche aerodinamiche fondamentali del velivolo. Nell'APP è possibile scegliere da un menu il grafico desiderato, per poi visualizzarlo tramite l'apposito bottone.

```

1 % -----
2 % Esercizi DATCOM
3 % Studente: Giovanni Battista Varriale
4 % Data: Gennaio 2024
5 %
6
7 % Preliminary commands
8 clc; clear all; close all;
9 scrsz = get(0,'ScreenSize');
10 set(0,'DefaultFigurePosition',...
11     [scrsz(3)/2-10 50 scrsz(3)/2 0.82*scrsz(4)]);
12 set(0,'defaulttextfontsize',10);
13 set(0,'defaulttextfontname','Consolas');
14
15 %% Importazione del file completo DATCOM
16

```

```

17 %P2010=datcomimport('p2010_6_gennaio.out');
18 %save("P2010.mat","P2010");
19 load("P2010.mat");
20 % =====
21 % LEGENDA
22 % =====
23 %
24 % P2010{1} = Wing+Ailerons
25 %
26 % P2010{2} = Wing+Flaps
27 %
28 % P2010{3} = TOTAL (Clean Wing, Body, Htail + Elevator, Vtail, Engines)
29 %
30 % Sottocasi (BUILD OPTION):
31 % es. P2010{3}.cl(:,:,1,k), con k=
32 % 1: BODY ALONE
33 % 2: WING ALONE
34 % 3: HTAIL ALONE
35 % 4: VTAIL ALONE
36 % 5: WING-BODY
37 % 6: BODY-HTAIL
38 % 7: BODY-VTAIL
39 % 8: WING-BODY-HTAIL
40 % 9: WING-BODY-VTAIL
41 % 10: WING-BODY-HTAIL-VTAIL
42 % 11: WING-BODY-HTAIL-VTAIL-VFIN-ENGINES
43 %
44 % =====
45 %
46 %% Casi da esaminare
47 % (a) Wing
48 % (b) Wing-Body
49 % (c) Wing-Body-Vertical Tail
50 % (d) Velivolo completo
51 %
52 %% Curva di Portanza
53 figure("Name","Curve caratteristiche",'numbertitle','off');
54 subplot(2,2,1)
55 plot(P2010{3}.alpha,P2010{3}.cl(:,:,1,2),'k--','LineWidth',1,...
56 'MarkerSize',5,'MarkerFaceColor','k','MarkerEdgeColor','k'); hold on;
57 plot(P2010{3}.alpha,P2010{3}.cl(:,:,1,5),'k-.diamond','LineWidth',1,...
58 'MarkerSize',5,'MarkerFaceColor','k','MarkerEdgeColor','k'); hold on;
59 plot(P2010{3}.alpha,P2010{3}.cl(:,:,1,9),'b-o','LineWidth',1,...
60 'MarkerSize',2,'MarkerFaceColor','b','MarkerEdgeColor','b'); hold on;
61 plot(P2010{3}.alpha,P2010{3}.cl(:,:,1,11),'k-square','LineWidth',1,...
62 'MarkerSize',5,'MarkerFaceColor','k','MarkerEdgeColor','k'); hold on;
63 plot([min(P2010{3}.alpha) max(P2010{3}.alpha)],[0 0],'k',...
64 [0 0],[-2 2],':k','LineWidth',1); hold off;
65
66 title('Curva di portanza','FontName','Consolas');
67 xlabel('\alpha (°)','FontName','Consolas');
68 ylabel('C_L','FontName','Consolas');
69 grid on;
70 set(gca,'XLim',[min(P2010{3}.alpha) max(P2010{3}.alpha)],'YLim',[-1.5 1.5],...
71 'FontName','Consolas','XMinorTick','on','YMinorTick','on');
72
73 legend('Wing','Wing-Body','Wing-Body-V Tail','Velivolo Completo','Location','SouthEast');
74
75 %% Curva di Resistenza
76 subplot(2,2,2)
77
78 plot(P2010{3}.alpha,P2010{3}.cd(:,:,1,2),'k--','LineWidth',1,...
79 'MarkerSize',5,'MarkerFaceColor','k','MarkerEdgeColor','k'); hold on;
80 plot(P2010{3}.alpha,P2010{3}.cd(:,:,1,5),'k-.diamond','LineWidth',1,...
81 'MarkerSize',5,'MarkerFaceColor','k','MarkerEdgeColor','k'); hold on;
82 plot(P2010{3}.alpha,P2010{3}.cd(:,:,1,9),'b-o','LineWidth',1,...
83 'MarkerSize',2,'MarkerFaceColor','b','MarkerEdgeColor','b'); hold on;
84 plot(P2010{3}.alpha,P2010{3}.cd(:,:,1,11),'k-square','LineWidth',1,...
85 'MarkerSize',5,'MarkerFaceColor','k','MarkerEdgeColor','k'); hold on;
86 plot([min(P2010{3}.alpha) max(P2010{3}.alpha)],[0 0],'k',...
87 [0 0],[-2 2],':k','LineWidth',1); hold off;
88
89 title('Curva di resistenza','FontName','Consolas');
90 xlabel('\alpha (°)','FontName','Consolas');
91 ylabel('C_D','FontName','Consolas');
92 grid on;
93 set(gca,'XLim',[min(P2010{3}.alpha) max(P2010{3}.alpha)],'YLim',[0.001 0.14],...
94 'FontName','Consolas','XMinorTick','on','YMinorTick','on');
95
96 legend('Wing','Wing-Body','Wing-Body-V Tail','Velivolo Completo','Location','North');
97
98 %% Polare Aerodinamica
99 subplot(2,2,3)
100
101 plot(P2010{3}.cd(:,:,1,2),P2010{3}.cl(:,:,1,2),'k--','LineWidth',1,...
102 'MarkerSize',5,'MarkerFaceColor','k','MarkerEdgeColor','k'); hold on;
103 plot(P2010{3}.cd(:,:,1,5),P2010{3}.cl(:,:,1,5),'k-.diamond','LineWidth',1,...
104 'MarkerSize',5,'MarkerFaceColor','k','MarkerEdgeColor','k'); hold on;
105 plot(P2010{3}.cd(:,:,1,9),P2010{3}.cl(:,:,1,9),'b-o','LineWidth',1,...

```

```

106 'MarkerSize',2,'MarkerFaceColor','b','MarkerEdgeColor','b'); hold on;
107 plot(P2010{3}.cd(:,:,1,11),P2010{3}.cl(:,:,1,11),'k-square','LineWidth',1, ...
108 'MarkerSize',5,'MarkerFaceColor','k','MarkerEdgeColor','k'); hold on;
109
110 title('Polare aerodinamica','FontName','Consolas');
111 xlabel('C_D','FontName','Consolas');
112 ylabel('C_L','FontName','Consolas');
113 grid on;
114 set(gca,'XLim',[0 0.05],'YLim',[-0.8 0.8],...
115 'FontName','Consolas','XMinorTick','on','YMinorTick','on');
116
117 legend('Wing','Wing-Body','Wing-Body-V Tail','Velivolo Completo','Location','East');
118
119 %% Coefficiente di Momento
120 subplot(2,2,4)
121
122 plot(P2010{3}.alpha,P2010{3}.cm(:,:,1,2),'k--','LineWidth',1, ...
123 'MarkerSize',5,'MarkerFaceColor','k','MarkerEdgeColor','k'); hold on;
124 plot(P2010{3}.alpha,P2010{3}.cm(:,:,1,5),'k-diamond','LineWidth',1, ...
125 'MarkerSize',5,'MarkerFaceColor','k','MarkerEdgeColor','k'); hold on;
126 plot(P2010{3}.alpha,P2010{3}.cm(:,:,1,9),'b-o','LineWidth',1, ...
127 'MarkerSize',2,'MarkerFaceColor','b','MarkerEdgeColor','b'); hold on;
128 plot(P2010{3}.alpha,P2010{3}.cm(:,:,1,11),'k-square','LineWidth',1, ...
129 'MarkerSize',5,'MarkerFaceColor','k','MarkerEdgeColor','k'); hold on;
130 plot([min(P2010{3}.alpha) max(P2010{3}.alpha)], [0 0],':k',...
131 [0 0],[-2 2],':k','LineWidth',1); hold off;
132
133 title('Coefficiente di Momento di Beccaggio','FontName','Consolas');
134 xlabel('\alpha (°)','FontName','Consolas');
135 ylabel('C_M','FontName','Consolas');
136 grid on;
137 set(gca,'XLim',[-14 14],'YLim',[-0.4 0.4],...
138 'FontName','Consolas','XMinorTick','on','YMinorTick','on');
139
140 legend('Wing','Wing-Body','Wing-Body-V Tail','Velivolo Completo','Location','SouthWest');
141
142 saveas(gcf,'CurveCarattP2010.eps');
143
144 %% CL al variare della deflessione dei flap/elevator
145 figure("Name","C_L al variare della deflessione dei flap/elevator",'numbertitle', 'off');
146 tcl = tiledlayout(1,2,'TileSpacing','Tight','Padding','loose');
147
148 % flap
149 nexttile
150 deltacl = P2010{2}.cl+P2010{2}.dcl_sym(1);
151 plot(P2010{2}.alpha,deltacl,'b-','LineWidth',0.5, ...
152 'MarkerSize',5,'MarkerFaceColor','k','MarkerEdgeColor','k'); hold on;
153 deltacl = P2010{2}.cl+P2010{2}.dcl_sym(3);
154 plot(P2010{2}.alpha,deltacl,'k-','LineWidth',0.5, ...
155 'MarkerSize',5,'MarkerFaceColor','k','MarkerEdgeColor','k'); hold on;
156 deltacl = P2010{2}.cl+P2010{2}.dcl_sym(5);
157 plot(P2010{2}.alpha,deltacl,'k--','LineWidth',0.5, ...
158 'MarkerSize',5,'MarkerFaceColor','k','MarkerEdgeColor','k'); hold on;
159 deltacl = P2010{2}.cl+P2010{2}.dcl_sym(7);
160 plot(P2010{2}.alpha,deltacl,'k:','LineWidth',0.5, ...
161 'MarkerSize',5,'MarkerFaceColor','k','MarkerEdgeColor','k'); hold on;
162 deltacl = P2010{2}.cl+P2010{2}.dcl_sym(9);
163 plot(P2010{2}.alpha,deltacl,'k-','LineWidth',0.5, ...
164 'MarkerSize',5,'MarkerFaceColor','k','MarkerEdgeColor','k'); hold on;
165
166 plot([min(P2010{3}.alpha) max(P2010{3}.alpha)], [0 0],':k',...
167 [0 0],[-2 2],':k','LineWidth',1); hold off;
168
169 title('C_L al variare di \delta_f','FontName','Consolas');
170 xlabel('\alpha (°)','FontName','Consolas');
171 ylabel('C_L','FontName','Consolas');
172 grid on;
173 set(gca,'XLim',[min(P2010{3}.alpha) max(P2010{3}.alpha)],'YLim',[-1.5 1.8],...
174 'FontName','Consolas','XMinorTick','on','YMinorTick','on');
175 axis square;
176 legend('\delta_f=0°','\delta_f=10°','\delta_f=20°','\delta_f=30°','\delta_f=40°','Location','SouthEast');
177
178 % elevator
179 nexttile
180 deltacl = P2010{3}.cl(:,:,1,11)+P2010{3}.dcl_sym(1);
181 plot(P2010{3}.alpha,deltacl,'k--','LineWidth',0.5, ...
182 'MarkerSize',5,'MarkerFaceColor','k','MarkerEdgeColor','k'); hold on;
183 deltacl = P2010{3}.cl(:,:,1,11)+P2010{3}.dcl_sym(3);
184 plot(P2010{3}.alpha,deltacl,'k-','LineWidth',0.5, ...
185 'MarkerSize',5,'MarkerFaceColor','k','MarkerEdgeColor','k'); hold on;
186 deltacl = P2010{3}.cl(:,:,1,11)+P2010{3}.dcl_sym(5);
187 plot(P2010{3}.alpha,deltacl,'b-','LineWidth',0.5, ...
188 'MarkerSize',5,'MarkerFaceColor','k','MarkerEdgeColor','k'); hold on;
189 deltacl = P2010{3}.cl(:,:,1,11)+P2010{3}.dcl_sym(7);
190 plot(P2010{3}.alpha,deltacl,'k:','LineWidth',0.5, ...
191 'MarkerSize',5,'MarkerFaceColor','k','MarkerEdgeColor','k'); hold on;
192 deltacl = P2010{3}.cl(:,:,1,11)+P2010{3}.dcl_sym(9);
193 plot(P2010{3}.alpha,deltacl,'k-','LineWidth',0.5, ...

```

```

195     'MarkerSize',5,'MarkerFaceColor','k','MarkerEdgeColor','k'); hold on;
196
197 plot([min(P2010{3}.alpha) max(P2010{3}.alpha)], [0 0],':k',...
198 [0 0],[-2 2],':k','LineWidth',1); hold off;
199
200 title('C_L al variare di \delta_e','FontName','Consolas');
201 xlabel('\alpha (°)','FontName','Consolas');
202 ylabel('C_L','FontName','Consolas');
203 grid on;
204 set(gca,'XLim',[min(P2010{3}.alpha) max(P2010{3}.alpha)],'YLim',[-1.5 1.8],...
205 'FontName','Consolas','XMinorTick','on','YMinorTick','on');
206 axis square;
207 legend('\delta_e=-20°','\delta_e=-10°','\delta_e=0°','\delta_e=10°','\delta_e=20°','Location','SouthEast');
208
209 saveas(gcf,'CLflapEl.eps');
210
211 %% CD al variare della deflessione dei flap
212 figure("Name","C_D al variare della deflessione dei flap/elevator",'numbertitle', 'off');
213 tcd = tiledlayout(1,2,'TileSpacing','Tight','Padding','loose');
214 % flap
215 nexttile
216 deltacl = P2010{2}.cd+P2010{2}.dcdmin_sym(1);
217 plot(P2010{2}.alpha,deltacl,'b-','LineWidth',0.5,...
218 'MarkerSize',5,'MarkerFaceColor','k','MarkerEdgeColor','k'); hold on;
219 deltacl = P2010{2}.cd+P2010{2}.dcdmin_sym(3);
220 plot(P2010{2}.alpha,deltacl,'k-','LineWidth',0.5,...
221 'MarkerSize',5,'MarkerFaceColor','k','MarkerEdgeColor','k'); hold on;
222 deltacl = P2010{2}.cd+P2010{2}.dcdmin_sym(5);
223 plot(P2010{2}.alpha,deltacl,'k--','LineWidth',0.5,...
224 'MarkerSize',5,'MarkerFaceColor','k','MarkerEdgeColor','k'); hold on;
225 deltacl = P2010{2}.cd+P2010{2}.dcdmin_sym(7);
226 plot(P2010{2}.alpha,deltacl,'k:','LineWidth',0.5,...
227 'MarkerSize',5,'MarkerFaceColor','k','MarkerEdgeColor','k'); hold on;
228 deltacl = P2010{2}.cd+P2010{2}.dcdmin_sym(9);
229 plot(P2010{2}.alpha,deltacl,'k-','LineWidth',0.5,...
230 'MarkerSize',5,'MarkerFaceColor','k','MarkerEdgeColor','k'); hold on;
231
232 plot([min(P2010{3}.alpha) max(P2010{3}.alpha)], [0 0],':k',...
233 [0 0],[-2 2],':k','LineWidth',1); hold off;
234
235 title('C_D al variare di \delta_f','FontName','Consolas');
236 xlabel('\alpha (°)','FontName','Consolas');
237 ylabel('C_D','FontName','Consolas');
238 grid on;
239 set(gca,'XLim',[min(P2010{3}.alpha) max(P2010{3}.alpha)],'YLim',[0 0.145],...
240 'FontName','Consolas','XMinorTick','on','YMinorTick','on');
241 axis square;
242 legend('\delta_f=0°','\delta_f=10°','\delta_f=20°','\delta_f=30°','\delta_f=40°','Location','SouthEast');
243
244 %% elevator
245 nexttile
246 deltacd = P2010{3}.cd(:,:,1,11)+P2010{3}.dcdmin_sym(1);
247 plot(P2010{3}.alpha,deltacd,'k--','LineWidth',0.5,...
248 'MarkerSize',5,'MarkerFaceColor','k','MarkerEdgeColor','k'); hold on;
249 deltacd = P2010{3}.cd(:,:,1,11)+P2010{3}.dcdmin_sym(3);
250 plot(P2010{3}.alpha,deltacd,'k-','LineWidth',0.5,...
251 'MarkerSize',5,'MarkerFaceColor','k','MarkerEdgeColor','k'); hold on;
252 deltacd = P2010{3}.cd(:,:,1,11)+P2010{3}.dcdmin_sym(5);
253 plot(P2010{3}.alpha,deltacd,'k:','LineWidth',0.5,...
254 'MarkerSize',5,'MarkerFaceColor','k','MarkerEdgeColor','k'); hold on;
255 deltacd = P2010{3}.cd(:,:,1,11)+P2010{3}.dcdmin_sym(7);
256 plot(P2010{3}.alpha,deltacd,'b-','LineWidth',0.5,...
257 'MarkerSize',5,'MarkerFaceColor','k','MarkerEdgeColor','k'); hold on;
258 deltacd = P2010{3}.cd(:,:,1,11)+P2010{3}.dcdmin_sym(9);
259 plot(P2010{3}.alpha,deltacd,'k-','LineWidth',0.5,...
260 'MarkerSize',5,'MarkerFaceColor','k','MarkerEdgeColor','k'); hold on;
261 deltacd = P2010{3}.cd(:,:,1,11)+P2010{3}.dcdmin_sym(11);
262 plot(P2010{3}.alpha,deltacd,'k-','LineWidth',0.5,...
263 'MarkerSize',5,'MarkerFaceColor','k','MarkerEdgeColor','k'); hold on;
264
265 plot([min(P2010{3}.alpha) max(P2010{3}.alpha)], [0 0],':k',...
266 [0 0],[-2 2],':k','LineWidth',1); hold off;
267
268 title('C_D al variare di \delta_e','FontName','Consolas');
269 xlabel('\alpha (°)','FontName','Consolas');
270 ylabel('C_D','FontName','Consolas');
271 grid on;
272 set(gca,'XLim',[min(P2010{3}.alpha) max(P2010{3}.alpha)],'YLim',[0 0.145],...
273 'FontName','Consolas','XMinorTick','on','YMinorTick','on');
274 axis square;
275
276 legend('\delta_e=-20°','\delta_e=-10°','\delta_e=0°','\delta_e=10°','\delta_e=20°','Location','North');
277
278 saveas(gcf,'CDflapEl.eps');
279
280 %% CM al variare della deflessione dei flap
281 figure("Name","C_M al variare della deflessione dei flap/elevator",'numbertitle', 'off');
282 tcm = tiledlayout(1,2,'TileSpacing','Tight','Padding','loose');

```

```

284 %   flap
285 nexttile
286 deltamc = P2010{2}.cm+P2010{2}.dcm_sym(1);
287 plot(P2010{2}.alpha,deltamc,'b-','LineWidth',0.5,...
288 'MarkerSize',5,'MarkerFaceColor','k','MarkerEdgeColor','k'); hold on;
289 deltamc = P2010{2}.cm+P2010{2}.dcm_sym(3);
290 plot(P2010{2}.alpha,deltamc,'k-','LineWidth',0.5,...
291 'MarkerSize',5,'MarkerFaceColor','k','MarkerEdgeColor','k'); hold on;
292 deltamc = P2010{2}.cm+P2010{2}.dcm_sym(5);
293 plot(P2010{2}.alpha,deltamc,'k--','LineWidth',0.5,...
294 'MarkerSize',5,'MarkerFaceColor','k','MarkerEdgeColor','k'); hold on;
295 deltamc = P2010{2}.cm+P2010{2}.dcm_sym(7);
296 plot(P2010{2}.alpha,deltamc,'k-','LineWidth',0.5,...
297 'MarkerSize',5,'MarkerFaceColor','k','MarkerEdgeColor','k'); hold on;
298 deltamc = P2010{2}.cm+P2010{2}.dcm_sym(9);
299 plot(P2010{2}.alpha,deltamc,'k-','LineWidth',0.5,...
300 'MarkerSize',5,'MarkerFaceColor','k','MarkerEdgeColor','k'); hold on;
301
302
303 plot([min(P2010{3}.alpha) max(P2010{3}.alpha)], [0 0], ':k',...
304 [0 0], [-2 2], ':k','LineWidth',1); hold off;
305
306 title('C_M al variare di \delta_f','FontName','Consolas');
307 xlabel('\alpha (°)','FontName','Consolas');
308 ylabel('C_M','FontName','Consolas');
309 grid on;
310 set(gca,'XLim',[min(P2010{3}.alpha) 14],'YLim',[-0.2 0.011],...
311 'FontName','Consolas','XMinorTick','on','YMinorTick','on');
312 axis square;
313 legend('\delta_f=0°','\delta_f=10°','\delta_f=20°','\delta_f=30°','\delta_f=40°','Location','SouthEast');
314
315 %   elevator
316 nexttile
317 deltamc = P2010{3}.cm(:,:,1,11)+P2010{3}.dcm_sym(1);
318 plot(P2010{3}.alpha,deltamc,'k--','LineWidth',0.5,...
319 'MarkerSize',5,'MarkerFaceColor','k','MarkerEdgeColor','k'); hold on;
320 deltamc = P2010{3}.cm(:,:,1,11)+P2010{3}.dcm_sym(3);
321 plot(P2010{3}.alpha,deltamc,'k-','LineWidth',0.5,...
322 'MarkerSize',5,'MarkerFaceColor','k','MarkerEdgeColor','k'); hold on;
323 deltamc = P2010{3}.cm(:,:,1,11)+P2010{3}.dcm_sym(5);
324 plot(P2010{3}.alpha,deltamc,'b-','LineWidth',0.5,...
325 'MarkerSize',5,'MarkerFaceColor','k','MarkerEdgeColor','k'); hold on;
326 deltamc = P2010{3}.cm(:,:,1,11)+P2010{3}.dcm_sym(7);
327 plot(P2010{3}.alpha,deltamc,'k-','LineWidth',0.5,...
328 'MarkerSize',5,'MarkerFaceColor','k','MarkerEdgeColor','k'); hold on;
329 deltamc = P2010{3}.cm(:,:,1,11)+P2010{3}.dcm_sym(9);
330 plot(P2010{3}.alpha,deltamc,'k-','LineWidth',0.5,...
331 'MarkerSize',5,'MarkerFaceColor','k','MarkerEdgeColor','k'); hold on;
332
333
334 plot([min(P2010{3}.alpha) max(P2010{3}.alpha)], [0 0], ':k',...
335 [0 0], [-2 2], ':k','LineWidth',1); hold off;
336
337 title('C_M al variare di \delta_e','FontName','Consolas');
338 xlabel('\alpha (°)','FontName','Consolas');
339 ylabel('C_D','FontName','Consolas');
340 grid on;
341 set(gca,'XLim',[min(P2010{3}.alpha) max(P2010{3}.alpha)],'YLim',[-1 1],...
342 'FontName','Consolas','XMinorTick','on','YMinorTick','on');
343 axis square;
344
345 legend('\delta_e=-20°','\delta_e=-10°','\delta_e=0°','\delta_e=10°','\delta_e=20°','Location','SouthWest');
346
347 %exportgraphics(tcl,'CL_FlapElevP2010.eps','BackgroundColor','none')
348 %exportgraphics(tcd,'CD_FlapElevP2010.eps','BackgroundColor','none')
349 %exportgraphics(tcm,'CM_FlapElevP2010.eps','BackgroundColor','none')
350 saveas(gcf,'CMflapEl.eps');

```

In Figura 1.4 vengono mostrate le curve caratteristiche del velivolo nell varie configurazioni considerate.

Nelle Figure 1.5,1.6 e 1.7 viene riportato rispettivamente il comportamento dei coefficienti C_L , C_D e C_M al variare dell'angolo di deflessione dell'elevatore (δ_e) e dei flap (δ_f).

In Figura 1.8 viene riportato il risultato del successivo listato. Si ricerca il coefficiente di momento (C'_M)_{WB} intorno a un asse parallelo a quello di beccaggio e passante per il bordo d'attacco della corda media aerodinamica. Si usa la relazione:

$$(C'_M)_{WB} \approx (C_M)_{WB} - \left(\frac{X_{CG} - X_{le,\bar{c}}}{\bar{c}} (C_L)_{WB} \right) \quad (1.1)$$

in cui il prodotto al secondo membro è un momento di trasporto nel passaggio dal polo X_{CG} al polo $X_{le,\bar{c}}$.

```

1 %% CM WB e bordo d'attacco della cma
2 % Bisogna trovare:
3 % Cm'wb = Cm_wb - ((Xcg-Xle)/cma)*(Cl_Wb)

```

```

4 cma = 1.29; % corda media aerodinamica
5 Xle = 2.36; % coordinata del bordo di attacco
6 Xcg = Xle+cma/4; % centro di massa
7
8 Cm_Wb = P2010{3}.cm(:,:,1,5);
9 Cm_Full = P2010{3}.cm(:,:,1,11);
10 Cl_full = P2010{3}.cl(:,:,1,11);
11 Cl_Wb = P2010{3}.cl(:,:,1,5);
12
13 Cm_Wb_new = Cm_Wb-((Xcg-Xle)/cma)*(Cl_Wb);
14
15 t=figure("Name","Carichi di equilibrio e manovra sull'impennaggio orizzontale",'numbertitle','off');
16 plot(Cm_Wb_new,Cl_full,'k-o','LineWidth',1,...
17 'MarkerSize',5); hold on;
18 plot(Cm_Wb,Cl_full,'k--','LineWidth',1); hold on;
19 plot(Cm_Full(1:14),Cl_full(1:14),'k-.','LineWidth',1); hold on;
20
21 legend("(C_M)_Wing-Body",'(C_M)_Wing-Body",'(C_M)_Velivolo Completo','Location','SouthWest');
22 title("Carichi di equilibrio e manovra sull'impennaggio orizzontale",'FontName','Consolas');
23 xlabel('C_M','FontName','Consolas');
24 ylabel('C_L','FontName','Consolas');
25 grid on; axis square;
26 %exportgraphics(t,'CM_primoP2010.eps','BackgroundColor','none')
27 saveas(gcf,'CarichiEquilibrio.eps');
28
29 %clc; clear;

```

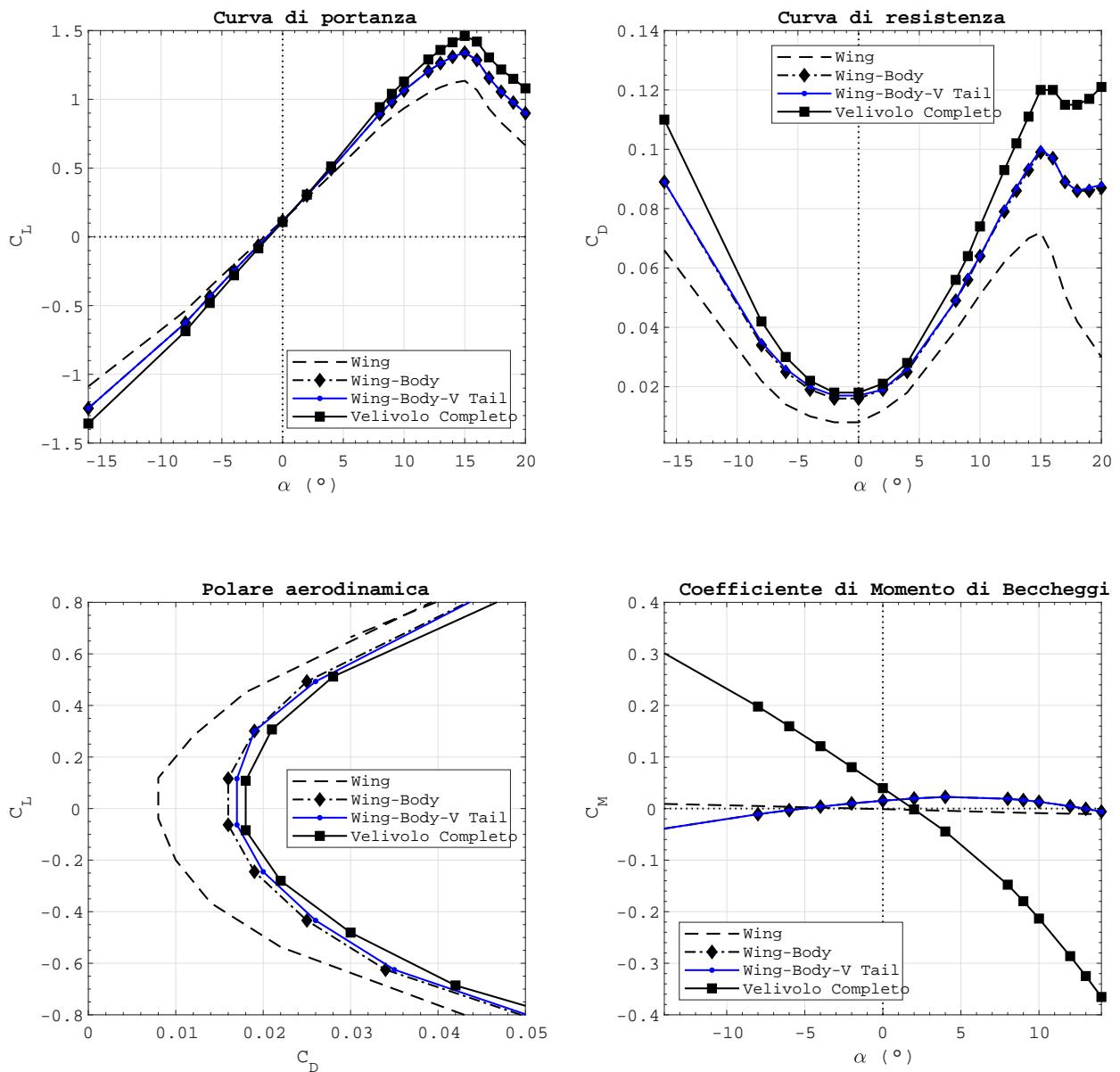


Figura 1.4: Curve Caratteristiche Tecnam P2010

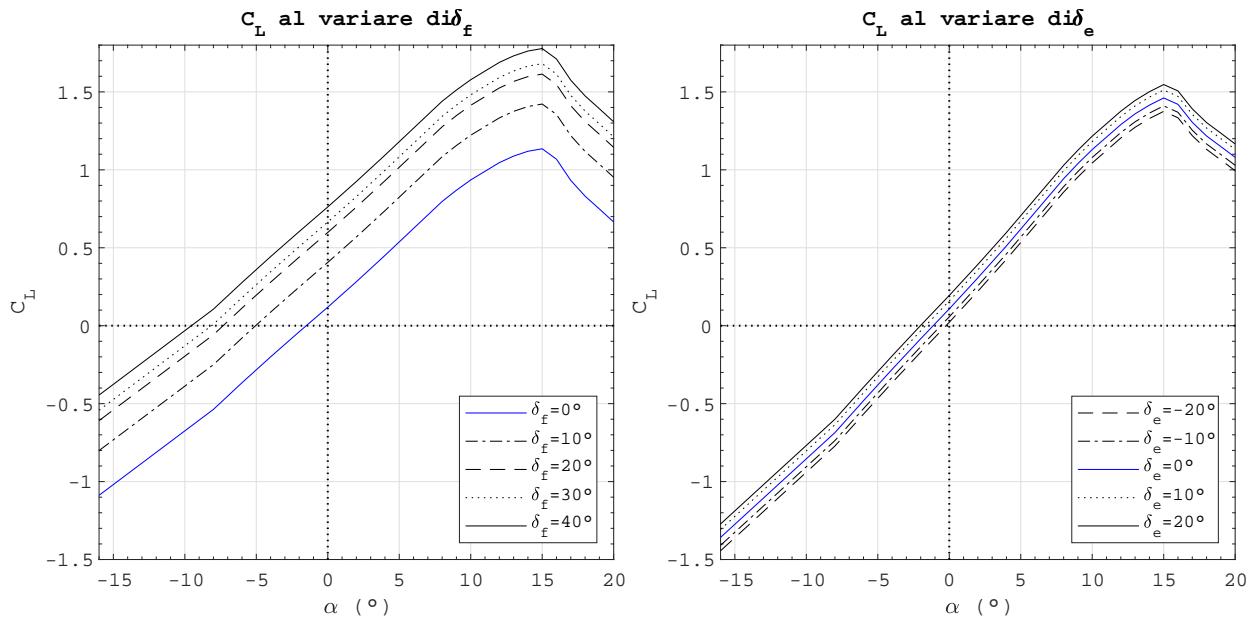


Figura 1.5: C_L al variare dell'angolo δ_e e δ_f

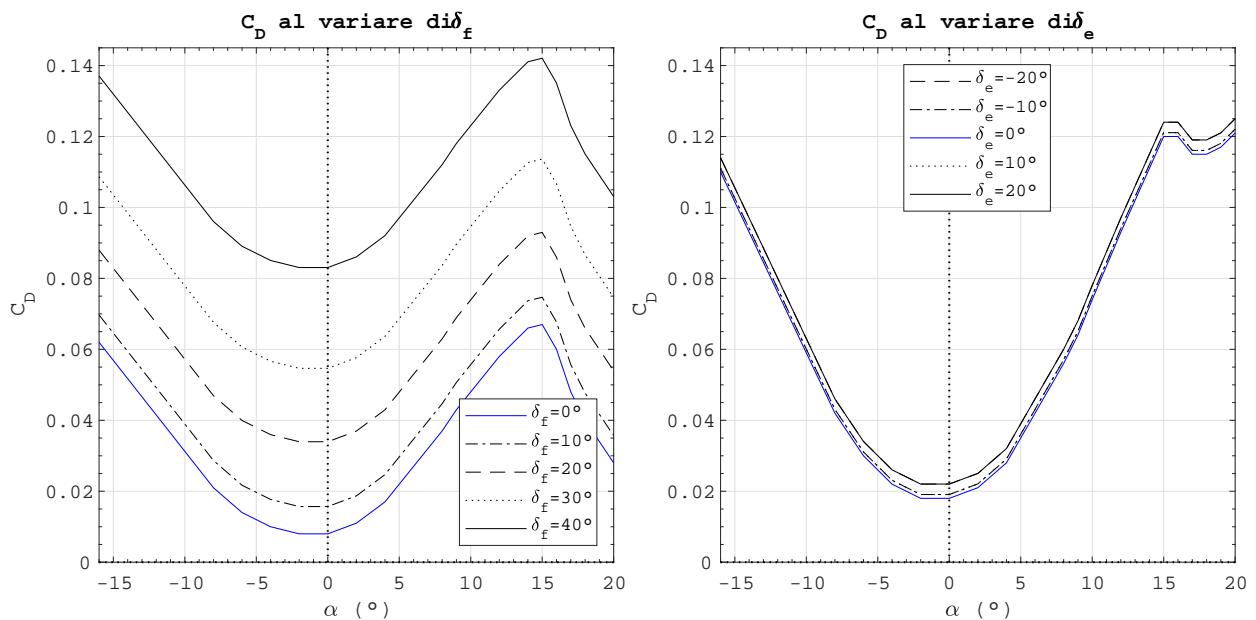


Figura 1.6: C_D al variare dell'angolo δ_e e δ_f

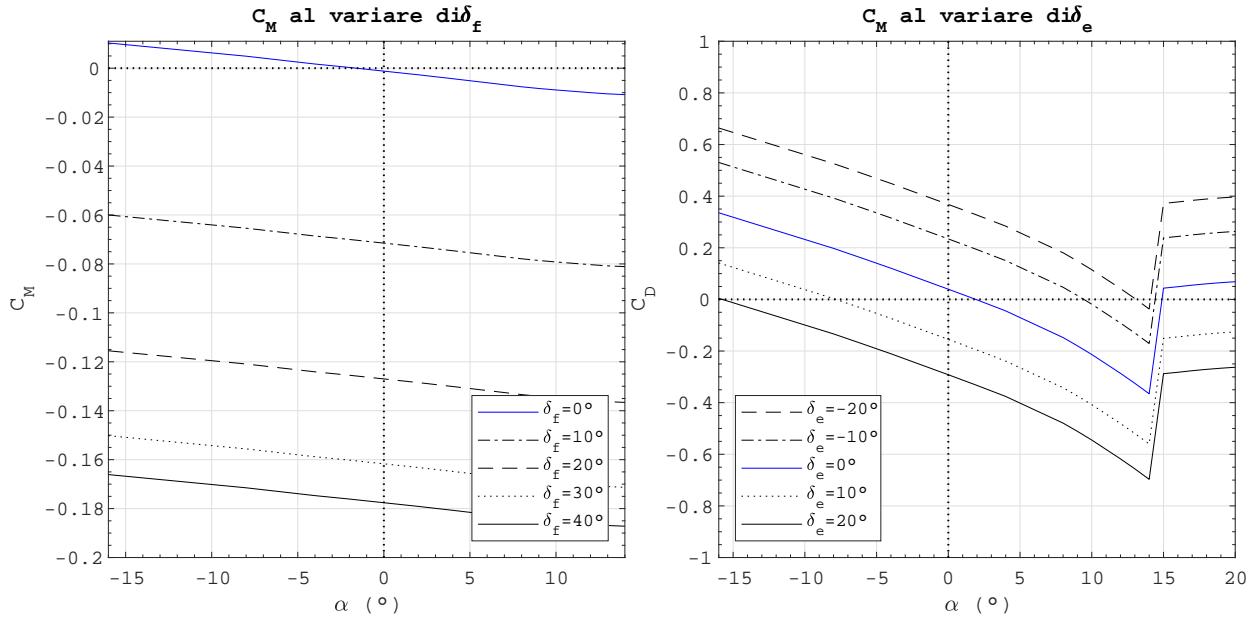


Figura 1.7: C_M al variare dell'angolo δ_e e δ_f

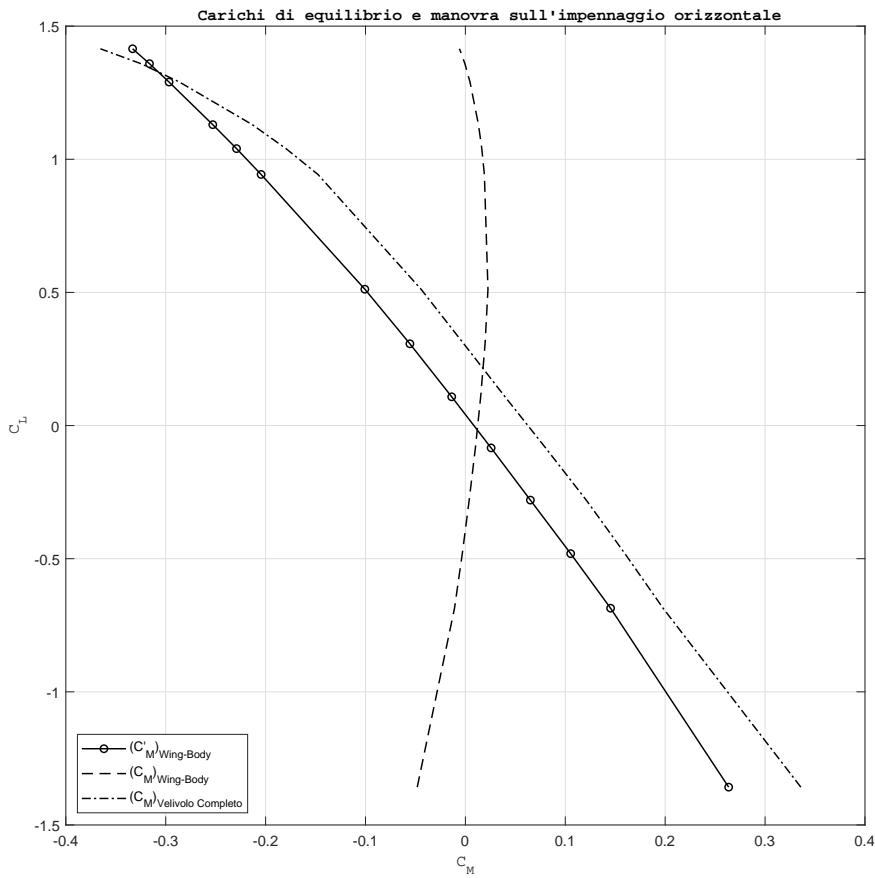


Figura 1.8: Carichi di Equilibrio

1.4 Modello 3D

Grazie ad un codice MATLAB di *Jafar Mohammed*, è possibile (anche se con alcune limitazioni) trasformare gli input DATCOM in un modello tridimensionale visualizzabile in MATLAB. Tale codice è quello del listato qui riportato.

```

1 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
2 % datcom3d v1.2 Input File %
3 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
4 %
5 clear
6 clc
7 clf
8 %
9 %%% VISUALIZATION and RESOLUTION %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
10 wframe = 1;           %0 = Shaded model
11                 %1 = Wireframe model (default)
12 %
13 fusres = 20;         %Fuselage resolution
14 wgres  = 20;         %Wing,HT,VT resolution
15 %
16 %% (DO NOT CHANGE VALUES IN THIS BOX) %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
17 XW=0;ZW=0;ALIW=0;XH=0;ZH=0;ALIH=0;XV=0;ZV=0;numVT=1;VERTUP=1;    %
18 NX=0;X=zeros(20);S=zeros(20);R=zeros(20);ZU=zeros(20);ZL=zeros(20);    %
19 CHRDR_WG=0;CHRDBP_WG=0;CHRDT_P_WG=0;SSPN_WG=0;SSPNOP_WG=0;SAVSI_WG=0;    %
20 SAVSO_WG=0;CHSTAT_WG=0;DHDADI_WG=0;DHDADO_WG=0;TC_WG=.12;                %
21 CHRDR_HT=0;CHRDBP_HT=0;CHRDT_P_HT=0;SSPN_HT=0;SSPNOP_HT=0;SAVSI_HT=0;    %
22 SAVSO_HT=0;CHSTAT_HT=0;DHDADI_HT=0;DHDADO_HT=0;TC_HT=.12;                %
23 CHRDR_VT=0;CHRDBP_VT=0;CHRDT_P_VT=0;SSPN_VT=0;SSPNOP_VT=0;SAVSI_VT=0;    %
24 SAVSO_VT=0;CHSTAT_VT=0;TC_VT=.12;                                         %
25 SPANFI_F=0;SPANFO_F=0;CHRDFI_F=0;CHRDFO_F=0;DELTA_F=0;                  %
26 SPANFI_A=0;SPANFO_A=0;CHRDFI_A=0;CHRDFO_A=0;DELTAL_A=0;DELTAR_A=0;      %
27 SPANFI_E=0;SPANFO_E=0;CHRDFI_E=0;CHRDFO_E=0;DELTA_E=0;                  %
28 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
29 %
30 %%% INPUT PARAMETERS BELOW %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
31 %
32 % SYNTHS parameters
33 XCG=3.296; ZCG=0.6;
34     XW=2.36;   ZW=0.739;   ALIW=1.5;
35     XH=6.87;   ZH=0.0;     ALIH=0.0;
36     XV=5.83;   ZV=0.0;
37     SCALE=1.0;
38 % BODY parameters
39 NX=17.0;
40     X=[0.4,0.5,0.6,1.0,1.5,1.9,2.3,2.4,3.0,3.65,4.0,4.56,5.2,5.9,6.3,7.4,7.6];
41     R=[0.,0.2,0.4,0.45,0.5,0.5,0.6,0.6,0.6,0.62,0.6,0.57,0.5,0.4,0.3,0.1,0.0];
42     ZU=[0.,0.17,0.18,0.24,0.33,0.5,0.6,0.7,0.7,0.7,0.6,0.4,0.2,0.1,0.1,0.];
43     ZL=[0.,-.1,-.2,-.6,-.6,-.6,-.6,-.6,-.6,-.5,-.5,-.4,-.4,-.2,0.];
44     BNOSE=1.0; BLN=8.8;
45     BTAIL=1.0; BLA=2.9;
46     ITYPE=1.0; METHOD=1.0;
47 % WING parameters (add suffix "_WG" to variables)
48 %
49     CHRDT_P_WG = 1.28;
50     CHRDBP_WG=1.4;
51     SSPNOP_WG=2.1;
52 %
53 SSPNE_WG = 5.0;
54 SSPN_WG = 5.25;
55 CHRDR_WG = 1.4;
56 SAVSI_WG = 0.0;
57 CHSTAT_WG = .0;
58 TWISTA_WG = 0.0;
59 DHDADI_WG = 0;
60 DHDADO_WG = 0.0;
61 TC_WG = .18;
62 % WING FLAPS (add suffix "_F" to variables)
63 %
64 %
65 %
66 % WING AILERONS (add suffix "_A" to variables)
67 %
68 % HORIZONTAL TAIL parameters (add suffix "_HT" to variables)
69 CHRDR_HT=0.76; CHRDT_P_HT=0.76;
70     SSPN_HT=1.68; SSPNE_HT=1.6;
71     SAVSI_HT=0.0;
72     CHSTAT_HT=0.0; TWISTA_HT=0.0;
73     DHDADI_HT=0.0;
74     TYP_HTE=1.0;
75 % ELEVATOR (add suffix "_E" to variables)
76 %
77 %
78 % VERTICAL TAIL parameters (add suffix "_VT" to variables)
79 % For twin vertical tails, you need to define:
80 %   numVT - number of vertical tails (for twin VT, should be 2)
81 %       YV - distance from FRL to stb. VT vertex

```

```

82 CHRDTP_VT=0.52; SSPNE_VT=1.38; SSPN_VT=1.47; CHRDR_VT=1.55;
83 SAVSI_VT=45.; CHSTAT_VT=0.25; TYPE_VT=1.0;
84
85 %%% PLOTTING %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
86 warning off MATLAB:divideByZero
87 hold on
88
89 plotFuselage(NX,X,S,R,ZU,ZL,fusres)
90 plotWing(XW,ZW,ALIH,CHRDR_WG,CHRDTP_WG,SSPN_WG,SSPNOP_WG,SAVSI_WG,SAVSO_WG,CHSTAT_WG,DHDADI_WG,
91 DHDADO_WG,...,
92 SPANFI_F,SPANFO_F,CHRDFI_F,CHRDFO_F,DELTA_F,SPANFI_A,SPANFO_A,CHRDFI_A,CHRDFO_A,DELTAL_A,DELTAR_A,
93 TC_WG,wgres)
94 plotHT(XH,ZH,ALIH,CHRDR_HT,CHRDTP_HT,SSPN_HT,SSPNOP_HT,SAVSI_HT,SAVSO_HT,CHSTAT_HT,DHDADI_HT,
95 DHDADO_HT,...,
96 SPANFI_E,SPANFO_E,CHRDFI_E,CHRDFO_E,DELTA_E,TC_HT,wgres)
97 plotVT(XV,YV,ZV,CHRDR_VT,CHRDTP_VT,SSPN_VT,SSPNOP_VT,SAVSI_VT,SAVSO_VT,CHSTAT_VT,VERTUP,TC_VT,
98 wgres)
99 if numVT > 1
100 plotVT(XV,-YV,ZV,CHRDR_VT,CHRDTP_VT,SSPN_VT,SSPNOP_VT,SAVSI_VT,SAVSO_VT,CHSTAT_VT,VERTUP,TC_VT,
101 wgres)
102 end
103 %% VIEWPORT/FIGURE PROPERTIES %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
104 if wframe == R
105 colormap([1 .7 .1]) %Set a/c to gold
106 colormap([0 0 1]) %Set a/c to blue
107 shading interp %Interpolated shading
108 lighting gouraud %Smooth airplane mesh
109 %camlight right %Apply a light source
110
111 %Custom Lighting Options, Note:[X Y Z]
112 light('Position',[1 -2 1],'Style','infinite');
113 light('Position',[1 2 1],'Style','infinite');
114 light('Position',[0 0 -6],'Style','infinite');
115 else
116 colormap([1 1 1]) %Set a/c to white
117 end
118
119 axis off %Turn off axis
120 axis equal %Correct aspect ratio
121 %camva(4.5) %Zoom in a/c to fit figure
122 view(3) %Apply initial viewport rotation
123 %campproj('perspective') %Perspective viewing (not R2006a compatible)
124 rotate3d on %Rotate icon enabled at start up
125
126 %showplottool('plotbrowser') %Enable the plot browser on startup
127 set(gcf,'NumberTitle','off','Name','Aircraft Plot','Color',[1 1 1]);

```

Le funzioni citate in tale codice sono:

```

1 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
2 % datcom3d Version 1.2 %
3 % March 10, 2008 %
4 % Created By: Jafar Mohammed %
5 % File: plotFuselage.m %
6 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
7
8 % Inputs:
9 % NX - number of fuselage stations
10 % Xs - X values at each station (Array of length NX)
11 % S - cross-sectional area at each station (Array of length NX)
12 % R - fuselage half-width at each station (Array of length NX)
13 % ZU - Upper Z-coordinate at each station (Array of length NX)
14 % ZL - Lower Z-coordinate at each station (Array of length NX)
15 % n - fuselage resolution (meshgrid)
16 function plotFuselage(NX,Xs,S,R,ZU,ZL,n)
17
18 ZC = [];
19 X = [];
20 Y = [];
21 Z = [];
22
23 % The following is taken from DATPLOT to get the coordinates
24 % of the fuselage. Please refer to that document for the details
25 % of this code.
26
27 %The center of each fuselage cross section is calculated
28 for i=1:1:NX
29 ZC(i) = (ZU(i)+ZL(i))*.5;
30 end
31
32 %Get Y and Z coordinates
33 for j=1:1:NX
34 W = R(j);
35 if (W == 0)
36 W = sqrt(S(j)/pi);
37 end
38 if (ZU(j) == 0)
39 ZU(j)=W;

```

```

40     ZL(j)=-W;
41   end
42   H = (ZU(j)-ZL(j))*.5;
43   WW=W*W;
44   if (W == 0)
45     WW = W;
46   end
47   HH=H*H;
48   if (H == 0)
49     HH = H;
50   end
51   WH=0;
52   if ((W ~= 0) && (H ~= 0))
53     WH=W*H;
54   end
55   GX2=pi/(n-1);
56   for k=1:1:n
57     THETA=(k-1)*GX2;
58     RHO=0;
59     if (WH ~= 0)
60       RHO = WH/(sqrt(HH*(sin(THETA)^2)+WW*(cos(THETA)^2)));
61     end
62     Z(j,k)=RHO*cos(THETA)+ZC(j);
63     Y(j,k)=RHO*sin(THETA);
64   end
65 end
66 for i=1:n
67   for j=1:NX
68     X(i,j) = Xs(j);
69   end
70 end
71 % Plot right and left halves of fuselage using SURF command
72 surf(X',Y,Z,'EdgeColor','k','DisplayName','Fuselage Stb.')
73 surf(X',-1.*Y,Z,'EdgeColor','k','DisplayName','Fuselage Port')
74 %surf2stl('fusoliera1.stl',X',Y,Z);
75 %surf2stl('fusoliera2.stl',X',-1.*Y,Z);

1 %%%%%%%%%%%%%%%%
2 % datcom3d Version 1.2 %%
3 % March 24, 2008 %%
4 % Created By: Jafar Mohammed %%
5 % File: plotHT.m %%
6 %%%%%%%%%%%%%%%%
7
8 % Please refer to plotWing.m for details of this code
9 function plotHT(XH,ZH,ALIH,CHRDR-HT,CHRDBP-HT,CHRDTp-HT,SSPN-HT,SSPNOP-HT,SAVSI-HT,SAVSO-HT,CHSTAT-HT,DHDADI-HT
10 ,DHDADO-HT, ...
11 SPANFI,SPANFO,CHRDFI,CHRDFO,DELTA,TC-HT,wgres)
12
13 if SSPNOP-HT == 0
14   AR_ht=2*(SSPN-HT*2)/(CHRDR-HT+CHRDTp-HT);
15   TR_ht=CHRDTp-HT/CHRDR-HT;
16
17 X_ht=[];
18 Y_ht=[];
19 Z_ht=[];
20 di_ht=[];
21 ih=[];
22 sweep = atand(tand(SAVSI-HT)-(4*(0-CHSTAT-HT)*(1-TR_ht))/(AR_ht*(1+TR_ht)));
23
24 for i=1:1:(wgres+1)
25   sw=(i-1)*SSPN-HT/wgres*tand(sweep);
26   for j=1:1:(wgres+1)
27     X_ht(i,j) = (((j-1)*CHRDR-HT/wgres)*(i-(i-1)*TR_ht/(wgres*TR_ht/(1-TR_ht)))+sw;
28     Y_ht(j,i) = (j-1)*SSPN-HT/wgres;
29   end
30 end
31
32 for i=1:1:(wgres+1)
33   for j=1:1:(wgres+1)
34     c=X_ht(j,wgres+1)-X_ht(j,1);
35     xval=X_ht(j,i)-X_ht(j,1);
36     Z_ht(j,i)=TC-HT*5*c*(0.2969.*((xval/c).^0.5-0.126.*((xval/c)-...
37       0.3516.*((xval/c).^2+0.2843.*((xval/c).^3-0.1015.*((xval/c).^4));
38     di_ht(j,i)=(j-1)*SSPN-HT/wgres*tand(DHDADI-HT);
39     ih(j,i)=(i-1)*c/wgres*tand(-1*ALIH);
40     %df(j,i)=(i-1)*CHRDR-HT/wgres*tand(-1*DELTA);
41     df(j,i)=(i-1)*c/wgres*tand(-1*DELTA);
42     l(j)=j;
43   end
44 end
45
46 I = [];
47 Iy = [];
48 Ii = [];
49 Io = [];
50 li = [];

```

```

51    l2 = [];
52    ind = [];
53
54    if SPANFO > (SSPNHT-SSPNHT/(wgres+1))
55        SPANFO = SPANFO + SSPNHT/(wgres+1);
56    end
57
58    [I,Iy] = find((Y_ht >= SPANFI) & (Y_ht <= SPANFO));
59
60    Iymin = min(I);
61    Iymax = max(I);
62    Iyminf = Iymin+1;
63    Iymaxf = Iymax-1;
64
65    for i=Iymin:Iymaxf, ind((i)-(Iyminf-1))=i;, end
66    for i=1:Iymin-1, Ii(i)=i;, end
67    for i=Iymax+1:(wgres+1), Io((i-1)-(Iymax-1))=i;, end
68
69    ci=X_ht(Iymin,wgres+1)-X_ht(Iymin,1);
70    fc=int32(CHRDFI/ci*wgres);
71    for i=1:fc, l2(i)=(wgres+1)-(i-1);, end
72    for i=1:(wgres)-fc, l1(i)=i;, end
73
74    Y_flap = Y_ht;
75    Y_flap(Ii,l2) = NaN;
76    Y_flap(Io,l2) = NaN;
77    Y_flap(l1,l1) = NaN;
78    Y_ht(ind,l2) = NaN;
79
80    %df = df - df(1,(wgres+1)-fc);
81    for i=1:(wgres+1)
82        dft = df(i,(wgres+1)-fc);
83        for j=1:(wgres+1)
84            df(i,j) = df(i,j) - dft;
85        end
86    end
87
88    surf(X_ht+XH,Y_ht,Z_ht+ZH+ih+di_ht,'EdgeColor','k','DisplayName','HT Upper-Stb.')
89    surf(X_ht+XH,Y_ht,-1.*Z_ht+ZH+ih+di_ht,'EdgeColor','k','DisplayName','HT Lower-Stb.')
90    surf(X_ht+XH,-1.*Y_ht,Z_ht+ZH+ih+di_ht,'EdgeColor','k','DisplayName','HT Upper-Port')
91    surf(X_ht+XH,-1.*Y_ht,-1.*Z_ht+ZH+ih+di_ht,'EdgeColor','k','DisplayName','HT Upper-Port')
92
93    %%surf2stl('ht1.stl',X_ht+XH,Y_ht,Z_ht+ZH+ih+di_ht);
94    %%surf2stl('ht2.stl',X_ht+XH,Y_ht,-1.*Z_ht+ZH+ih+di_ht);
95    %%surf2stl('ht3.stl',X_ht+XH,-1.*Y_ht,Z_ht+ZH+ih+di_ht);
96    %%surf2stl('ht4.stl',X_ht+XH,-1.*Y_ht,-1.*Z_ht+ZH+ih+di_ht);
97
98
99    if CHRDFI > 0
100        surf(X_ht+XH,Y_flap,Z_ht+ZH+ih+di_ht+df,'EdgeColor','r','DisplayName','Elev. Upper-Stb.')
101        surf(X_ht+XH,Y_flap,-1.*Z_ht+ZH+ih+di_ht+df,'EdgeColor','r','DisplayName','Elev. Lower-Stb.')
102        surf(X_ht+XH,-1.*Y_flap,Z_ht+ZH+ih+di_ht+df,'EdgeColor','r','DisplayName','Elev. Upper-Port')
103        surf(X_ht+XH,-1.*Y_flap,-1.*Z_ht+ZH+ih+di_ht+df,'EdgeColor','r','DisplayName','Elev. Lower-Port')
104    end
105 else
106    wgreso=int32(SSPNOPHT/SSPNHT*wgres);
107    wgresi=wgres-wgreso;
108
109    ARi_ht=2*((SSPNHT-SSPNOPHT)*2)/(CHRDRHT+CHRDBPHT);
110    ARo_ht=2*(SSPNOPHT*2)/(CHRDBPHT+CHRDTPHT);
111
112    TRi_ht=CHRDBPHT/CHRDRHT;
113    TRo_ht=CHRDTPTH/CHRDBPHT;
114
115    Xi_ht=[];
116    Xo_ht=[];
117    Yi_ht=[];
118    Yo_ht=[];
119    Zi_ht=[];
120    Zo_ht=[];
121    dii_ht=[];
122    dio_ht=[];
123    iwi=[];
124    iwo=[];
125
126    sweepi = atand(tand(SAVSIHT)-(4*(0-CHSTATHT)*(1-TRi_ht))/(ARi_ht*(1+TRi_ht)));
127    sweepo = atand(tand(SAVSOHT)-(4*(0-CHSTATHT)*(1-TRo_ht))/(ARo_ht*(1+TRo_ht)));
128
129    for i=1:1:(wgres+1)
130        sw=(i-1)*(SSPNHT-SSPNOPHT)/wgres*tand(sweepi);
131        for j=1:1:(wgres+1)
132            Xi_ht(i,j) = (((j-1)*CHRDRHT/wgres)*(1-(i-1)*TRi_ht/(wgres*TRi_ht/(1-TRi_ht)))+sw;
133            Yi_ht(j,i) = (j-1)*(SSPNHT-SSPNOPHT)/wgres;
134        end
135    end
136
137    for i=1:1:(wgres+1)
138        for j=1:1:(wgres+1)
139            c=Xi_ht(j,wgres+1)-Xi_ht(j,1);

```

```

140     xval=Xi_ht(j,i)-Xi_ht(j,1);
141     Zi_ht(j,i)=TC_HT*5*c*(0.2969.* (xval/c).^0.5-0.126.* (xval/c) -...
142         0.3516.* (xval/c).^2+0.2843.* (xval/c).^3-0.1015.* (xval/c).^4);
143     dii_ht(j,i)=(j-1)*(SSPNHT-SSPNOPHT)/wgres*tand(DHDADIHT);
144     iwi(j,i)=(i-1)*c/wgres*tand(-1*ALIH);
145   end
146 end
147
148 for i=1:1:(wgres+1)
149   sw=(i-1)*(SSPNOPHT)/wgres*tand(sweepto);
150   for j=1:1:(wgres+1)
151     Xo_ht(i,j) = (((j-1)*CHRDBPHT/wgres)*(1-(i-1)*TRoht/(wgres*TRoht/(1-TRoht)))+sw+Xi_ht(wgres
152     +1,1);
153     Yo_ht(j,i) = (j-1)*(SSPNOPHT)/wgres + (SSPNHT-SSPNOPHT);
154   end
155
156 for i=1:1:(wgres+1)
157   for j=1:1:(wgres+1)
158     c=Xo_ht(j,wgres+1)-Xo_ht(j,1);
159     xval=Xo_ht(j,i)-Xo_ht(j,1);
160     Zo_ht(j,i)=TCHT*5*c*(0.2969.* (xval/c).^0.5-0.126.* (xval/c) -...
161         0.3516.* (xval/c).^2+0.2843.* (xval/c).^3-0.1015.* (xval/c).^4);
162     dio_ht(j,i)=(j-1)*(SSPNOPHT)/wgres*tand(DHDADOTH);
163     iwo(j,i)=(i-1)*c/wgres*tand(-1*ALIH);
164   end
165 end
166
167 surf(Xi_ht+XH,Yi_ht,Zi_ht+ZH+iwi+dii_ht,'EdgeColor','k','DisplayName','HT In. Upper-Stb.')
168 surf(Xi_ht+XH,Yi_ht,-1.*Zi_ht+ZH+iwi+dii_ht,'EdgeColor','k','DisplayName','HT In. Lower-Stb.')
169 surf(Xi_ht+XH,-1.*Yi_ht,Zi_ht+ZH+iwi+dii_ht,'EdgeColor','k','DisplayName','HT In. Upper-Port')
170 surf(Xi_ht+XH,-1.*Yi_ht,-1.*Zi_ht+ZH+iwi+dii_ht,'EdgeColor','k','DisplayName','HT In. Lower-Port')
171
172
173
174
175 surf(Xo_ht+XH,Yo_ht,Zo_ht+ZH+iwo+(SSPNHT-SSPNOPHT)*tand(DHDADIHT)+dio_ht,'EdgeColor','k','DisplayName','
176 HT Out. Upper-Stb.')
177 surf(Xo_ht+XH,Yo_ht,-1.*Zo_ht+ZH+iwo+(SSPNHT-SSPNOPHT)*tand(DHDADIHT)+dio_ht,'EdgeColor','k','
178 DisplayName','HT Out. Lower-Stb.')
179 surf(Xo_ht+XH,-1.*Yo_ht,Zo_ht+ZH+iwo+(SSPNHT-SSPNOPHT)*tand(DHDADIHT)+dio_ht,'EdgeColor','k','
180 DisplayName','HT Out. Upper-Port')
181 surf(Xo_ht+XH,-1.*Yo_ht,-1.*Zo_ht+ZH+iwo+(SSPNHT-SSPNOPHT)*tand(DHDADIHT)+dio_ht,'EdgeColor','k','
182 DisplayName','HT Out. Lower-Port')
183 end

```

```

1 %%%%%%%%%%%%%%%%
2 % datcom3d Version 1.2 %
3 % March 10, 2008 %
4 % Created By: Jafar Mohammed %
5 % File: plotVT.m %
6 %%%%%%%%%%%%%%%%
7
8 % Please refer to plotWing.m for details of this code
9 % Special parameter VERTUP:
10 % If VERTUP = 1, then vertical tail is above centerline
11 % If VERTUP = 0, then vertical tail is below centerline
12 % Special parameter YV:
13 % YV = distance from fuselage reference line to vertex of VT
14 function plotVT(XV,YV,ZV,CHRDR_VT,CHRDBP_VT,CHRDTP_VT,SSPN_VT,SSPNOP_VT,SAVSI_VT,SAVSO_VT,CHSTAT_VT,VERTUP,
15 TC_VT,wgres)
16
16 if SSPNOP_VT == 0
17   AR_vt=2*(SSPN_VT*2)/(CHRDR_VT+CHRDTP_VT);
18   TR_vt=CHRDTP_VT/CHRDR_VT;
19
20   X_vt=[];
21   Y_vt=[];
22   Z_vt=[];
23
24   sweep = atan(tand(SAVSI_VT)-(4*(0-CHSTAT_VT)*(1-TR_vt))/(AR_vt*(1+TR_vt)));
25
26   for i=1:1:(wgres+1)
27     sw=(i-1)*SSPN_VT/wgres*tand(sweep);
28     for j=1:1:(wgres+1)
29       X_vt(i,j) = (((j-1)*CHRDR_VT/wgres)*(1-(i-1)*TR_vt/(wgres*TR_vt/(1-TR_vt)))+sw;
30       Z_vt(j,i) = (j-1)*SSPN_VT/wgres;
31     end
32   end
33
34   for i=1:1:(wgres+1)
35     for j=1:1:(wgres+1)
36       c=X_vt(j,wgres+1)-X_vt(j,1);
37       xval=X_vt(j,i)-X_vt(j,1);
38       Y_vt(j,i)=TC_VT*5*c*(0.2969.* (xval/c).^0.5-0.126.* (xval/c) -...
39           0.3516.* (xval/c).^2+0.2843.* (xval/c).^3-0.1015.* (xval/c).^4);
40     end
41   end

```

```

43 if VERTUP > 0
44     surf(X_vt+XV,Y_vt+YV,Z_vt+ZV,'EdgeColor','k','DisplayName','VT Stb.')
45     surf(X_vt+XV,-1.*Y_vt+YV,Z_vt+ZV,'EdgeColor','k','DisplayName','VT Port')
46
47 %surf2stl('vt1.stl',X_vt+XV,Y_vt+YV,Z_vt+ZV);
48 else
49     surf(X_vt+XV,Y_vt+YV,-1.*Z_vt+ZV,'EdgeColor','k','DisplayName','VT Stb.')
50     surf(X_vt+XV,-1.*Y_vt+YV,-1.*Z_vt+ZV,'EdgeColor','k','DisplayName','VT Port')
51
52 %surf2stl('vt2.stl',X_vt+XV,Y_vt+YV,Z_vt+ZV);
53 end
54 else
55 wgreso=int32(SSPNOP_VT/SSPN_VT*wgres);
56 wgresi=wgres-wgreso;
57
58 ARI_vt=2*((SSPN_VT-SSPNOP_VT)*2)/(CHRDR_VT+CHRDBP_VT);
59 ARo_vt=2*(SSPNOP_VT*2)/(CHRDBP_VT+CHRDTP_VT);
60
61 TRi_vt=CHRDBP_VT/CHRDR_VT;
62 TRo_vt=CHRDTP_VT/CHRDBP_VT;
63
64 Xi_vt=[];
65 Xo_vt=[];
66 Yi_vt=[];
67 Yo_vt=[];
68 Zi_vt=[];
69 Zo_vt=[];
70
71 sweepi = atand(tand(SAVSI_VT)-(4*(0-CHSTAT_VT)*(1-TRi_vt))/(ARI_vt*(1+TRi_vt)));
72 sweepo = atand(tand(SAVSO_VT)-(4*(0-CHSTAT_VT)*(1-TRo_vt))/(ARo_vt*(1+TRo_vt)));
73
74 for i=1:1:(wgres+1)
75     sw=(i-1)*(SSPN_VT-SSPNOP_VT)/wgres*tand(sweepi);
76     for j=1:1:(wgres+1)
77         Xi_vt(i,j) = (((j-1)*CHRDR_VT/wgres)*(1-(i-1)*TRi_vt/(wgres*TRi_vt/(1-TRi_vt)))+sw;
78         Zi_vt(j,i) = (j-1)*(SSPN_VT-SSPNOP_VT)/wgres;
79     end
80 end
81
82 for i=1:1:(wgres+1)
83     for j=1:1:(wgres+1)
84         c=Xi_vt(j,wgres+1)-Xi_vt(j,1);
85         xval=Xi_vt(j,i)-Xi_vt(j,1);
86         Yi_vt(j,i)=TC_VT*5*c*(0.2969.*((xval/c).^0.5-0.126.*((xval/c)-...
87             0.3516.*((xval/c).^2+0.2843.*((xval/c).^3-0.1015.*((xval/c).^4));
88     end
89 end
90
91 for i=1:1:(wgres+1)
92     sw=(i-1)*(SSPNOP_VT)/wgres*tand(sweepo);
93     for j=1:1:(wgres+1)
94         Xo_vt(i,j) = (((j-1)*CHRDBP_VT/wgres)*(1-(i-1)*TRo_vt/(wgres*TRo_vt/(1-TRo_vt)))+sw+Xi_vt(wgres+1,1);
95         Zo_vt(j,i) = (j-1)*(SSPNOP_VT)/wgres + (SSPN_VT-SSPNOP_VT);
96     end
97 end
98
99 for i=1:1:(wgres+1)
100    for j=1:1:(wgres+1)
101        c=Xo_vt(j,wgres+1)-Xo_vt(j,1);
102        xval=Xo_vt(j,i)-Xo_vt(j,1);
103        Yo_vt(j,i)=TC_VT*5*c*(0.2969.*((xval/c).^0.5-0.126.*((xval/c)-...
104            0.3516.*((xval/c).^2+0.2843.*((xval/c).^3-0.1015.*((xval/c).^4));
105    end
106 end
107
108 if VERTUP > 0
109     surf(Xi_vt+XV,Yi_vt+YV,Zi_vt+ZV,'EdgeColor','k','DisplayName','VT In. Stb.')
110     surf(Xi_vt+XV,-1.*Yi_vt+YV,Zi_vt+ZV,'EdgeColor','k','DisplayName','VT In. Port')
111     surf(Xo_vt+XV,Yo_vt+YV,Zo_vt+ZV,'EdgeColor','k','DisplayName','VT Out. Stb.')
112     surf(Xo_vt+XV,-1.*Yo_vt+YV,Zo_vt+ZV,'EdgeColor','k','DisplayName','VT Out. Port')
113
114
115 else
116     surf(Xi_vt+XV,Yi_vt+YV,-1.*Zi_vt+ZV,'EdgeColor','k','DisplayName','VT In. Stb.')
117     surf(Xi_vt+XV,-1.*Yi_vt+YV,-1.*Zi_vt+ZV,'EdgeColor','k','DisplayName','VT In. Port')
118     surf(Xo_vt+XV,Yo_vt+YV,-1.*Zo_vt+ZV,'EdgeColor','k','DisplayName','VT Out. Stb.')
119     surf(Xo_vt+XV,-1.*Yo_vt+YV,-1.*Zo_vt+ZV,'EdgeColor','k','DisplayName','VT Out. Port')
120 end
121
122 end
123
124 %%%
125 % datcom3d Version 1.2 %
126 % March 10, 2008 %
127 % Created By: Jafar Mohammed %
128 % File: plotVT.m %
129 %%%

```

```

8 % Please refer to plotWing.m for details of this code
9 % Special parameter VERTUP:
10 %   If VERTUP = 1, then vertical tail is above centerline
11 %   If VERTUP = 0, then vertical tail is below centerline
12 % Special parameter YV:
13 %   YV = distance from fuselage reference line to vertex of VT
14 function plotVT(XV,YV,ZV,CHRDR_VT,CHRDBP_VT,CHRDTP_VT,SSPN_VT,SSPNOP_VT,SAVSI_VT,SAVSO_VT,CHSTAT_VT,VERTUP,
15 TC_VT,wgres)
16
17 if SSPNOP_VT == 0
18     AR_vt=2*(SSPN_VT*2)/(CHRDR_VT+CHRDTP_VT);
19     TR_vt=CHRDTP_VT/CHRDR_VT;
20
21 X_vt=[];
22 Y_vt=[];
23 Z_vt=[];
24
25 sweep = atand(tand(SAVSI_VT)-(4*(0-CHSTAT_VT)*(1-TR_vt))/(AR_vt*(1+TR_vt)));
26
27 for i=1:1:(wgres+1)
28     sw=(i-1)*SSPN_VT/wgres*tand(sweep);
29     for j=1:1:(wgres+1)
30         X_vt(i,j) = (((j-1)*CHRDR_VT/wgres)*(1-(i-1)*TR_vt/(wgres*TR_vt/(1-TR_vt))))+sw;
31         Z_vt(j,i) = (j-1)*SSPN_VT/wgres;
32     end
33 end
34
35 for i=1:1:(wgres+1)
36     for j=1:1:(wgres+1)
37         c=X_vt(j,wgres+1)-X_vt(j,1);
38         xval=X_vt(j,1)-X_vt(j,1);
39         Y_vt(j,i)=TC_VT*5*c*(0.2969.*((xval/c).^0.5-0.126.*((xval/c)-...
40             0.3516.*((xval/c).^2+0.2843.*((xval/c).^3-0.1015.*((xval/c).^4));
41     end
42 end
43
44 if VERTUP > 0
45     surf(X_vt+XV,Y_vt+YV,Z_vt+ZV,'EdgeColor','k','DisplayName','VT Stb.')
46     surf(X_vt+XV,-1.*Y_vt+YV,Z_vt+ZV,'EdgeColor','k','DisplayName','VT Port')
47
48 %surf2stl('vt1.stl',X_vt+XV,Y_vt+YV,Z_vt+ZV);
49 else
50     surf(X_vt+XV,Y_vt+YV,-1.*Z_vt+ZV,'EdgeColor','k','DisplayName','VT Stb.')
51     surf(X_vt+XV,-1.*Y_vt+YV,-1.*Z_vt+ZV,'EdgeColor','k','DisplayName','VT Port')
52
53 %surf2stl('vt2.stl',X_vt+XV,Y_vt+YV,Z_vt+ZV);
54 end
55 else
56     wgreso=int32(SSPNOP_VT/SSPN_VT*wgres);
57     wgresi=wgres-wgreso;
58
59 ARI_vt=2*((SSPN_VT-SSPNOP_VT)*2)/(CHRDR_VT+CHRDBP_VT);
60 ARo_vt=2*(SSPNOP_VT*2)/(CHRDTP_VT+CHRDTP_VT);
61
62 TRi_vt=CHRDBP_VT/CHRDR_VT;
63 TRo_vt=CHRDTP_VT/CHRDBP_VT;
64
65 Xi_vt=[];
66 Xo_vt=[];
67 Yi_vt=[];
68 Yo_vt=[];
69 Zi_vt=[];
70 Zo_vt=[];
71
72 sweepi = atand(tand(SAVSI_VT)-(4*(0-CHSTAT_VT)*(1-TRi_vt))/(ARI_vt*(1+TRi_vt)));
73 sweepo = atand(tand(SAVSO_VT)-(4*(0-CHSTAT_VT)*(1-TRo_vt))/(ARo_vt*(1+TRo_vt)));
74
75 for i=1:1:(wgres+1)
76     sw=(i-1)*(SSPN_VT-SSPNOP_VT)/wgres*tand(sweepi);
77     for j=1:1:(wgres+1)
78         Xi_vt(i,j) = (((j-1)*CHRDR_VT/wgres)*(1-(i-1)*TRi_vt/(wgres*TRi_vt/(1-TRi_vt))))+sw;
79         Zi_vt(j,i) = (j-1)*(SSPN_VT-SSPNOP_VT)/wgres;
80     end
81 end
82
83 for i=1:1:(wgres+1)
84     for j=1:1:(wgres+1)
85         c=Xi_vt(j,wgres+1)-Xi_vt(j,1);
86         xval=Xi_vt(j,1)-Xi_vt(j,1);
87         Yi_vt(j,i)=TC_VT*5*c*(0.2969.*((xval/c).^0.5-0.126.*((xval/c)-...
88             0.3516.*((xval/c).^2+0.2843.*((xval/c).^3-0.1015.*((xval/c).^4));
89     end
90 end
91
92 for i=1:1:(wgres+1)
93     sw=(i-1)*(SSPNOP_VT)/wgres*tand(sweepo);
94     for j=1:1:(wgres+1)
95         Xo_vt(i,j) = (((j-1)*CHRDBP_VT/wgres)*(1-(i-1)*TRo_vt/(wgres*TRo_vt/(1-TRo_vt))))+sw+Xi_vt(wgres
+1,1);

```

```

95         Zo_vt(j,i) = (j-1)*(SSPNOP_VT)/wgres + (SSPN_VT-SSPNOP_VT);
96     end
97 end
98
99 for i=1:1:(wgres+1)
100    for j=1:1:(wgres+1)
101        c=Xo_vt(j,wgres+1)-Xo_vt(j,1);
102        xval=Xo_vt(j,i)-Xo_vt(j,1);
103        Yo_vt(j,i)=TC_VT*5*c*(0.2969.*xval/c).^0.5-0.126.*xval/c)-...
104            0.3516.*xval/c.^2+0.2843.*xval/c.^3-0.1015.*xval/c.^4);
105    end
106 end
107
108 if VERTUP > 0
109    surf(Xi_vt+XV,Yi_vt+YV,Zi_vt+ZV,'EdgeColor','k','DisplayName','VT In. Stb.')
110    surf(Xi_vt+XV,-1.*Yi_vt+YV,Zi_vt+ZV,'EdgeColor','k','DisplayName','VT In. Port')
111    surf(Xo_vt+XV,Yo_vt+YV,Zo_vt+ZV,'EdgeColor','k','DisplayName','VT Out. Stb.')
112    surf(Xo_vt+XV,-1.*Yo_vt+YV,Zo_vt+ZV,'EdgeColor','k','DisplayName','VT Out. Port')
113
114
115 else
116    surf(Xi_vt+XV,Yi_vt+YV,-1.*Zi_vt+ZV,'EdgeColor','k','DisplayName','VT In. Stb.')
117    surf(Xi_vt+XV,-1.*Yi_vt+YV,-1.*Zi_vt+ZV,'EdgeColor','k','DisplayName','VT In. Port')
118    surf(Xo_vt+XV,Yo_vt+YV,-1.*Zo_vt+ZV,'EdgeColor','k','DisplayName','VT Out. Stb.')
119    surf(Xo_vt+XV,-1.*Yo_vt+YV,-1.*Zo_vt+ZV,'EdgeColor','k','DisplayName','VT Out. Port')
120 end
121
122 end
123
124 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
125 % datcom3d Version 1.2 %
126 % March 27, 2008 %
127 % Created By: Jafar Mohammed %
128 % File: plotWing.m %
129 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
130
131 % Inputs:
132 % XW - x-coordinate of theoretical wing apex
133 % ZW - z-coordinate of theoretical wing apex
134 % ALIW - angle of incidence of the wing
135 % CHRDR - root chord (at centerline)
136 % CHRDBP - chord at breakpoint if outer and inner planforms exist
137 % CHRDTP - chord at tip
138 % SSPN - the semi-span of the wing (i.e. b/2)
139 % SSPNOP - the span of the outer planform measured from CHRDBP to CHRDTP
140 % SAVSI - the sweep of the inner planform
141 % SAVSO - the sweep of the outer planform
142 % CHSTAT - fraction of chord where sweep is measured from (i.e. 0.25)
143 % DHADADI - dihedral of the inner planform
144 % DHADADO - dihedral of the outer planform
145 % SPANFI - inner span position of flap
146 % SPANFO - outer span position of flap
147 % CHRDFI - chord position (wrt to LE) of flap at SPANFI
148 % CHRDFO - chord position (wrt to LE) of flap at SPANFO
149 % TC - t/c of the airfoil (i.e. 0.09, 0.12, 0.18, etc.)
150 % wgres - resolution of the planform(s) (meshgrid)
151
152 function plotWing(XW,ZW,ALIW,CHRDR_WG,CHRDBP_WG,CHRDTP_WG,SSPN_WG,SSPNOP_WG,SAVSI_WG,SAVSO_WG,CHSTAT_WG,
153 DHADADI_WG,DHADADO_WG,...
154     SPANFI_F,SPANFO_F,CHRDFI_F,CHRDFO_F,DELTA,SPANFI_A,SPANFO_A,CHRDFI_A,CHRDFO_A,DELTAR,DELTAL,
155     TC_WG,wgres)
156
157 % Determine if there is an outer planform
158 if SSPNOP_WG == 0
159     % Compute aspect ratio and taper ratio of wing
160     AR_wg=2*(SSPN_WG*2)/(CHRDR_WG+CHRDTP_WG);
161     TR_wg=CHRDTP_WG/CHRDR_WG;
162
163     % Allocate memory for arrays
164     X_wg=[];
165     Y_wg=[];
166     Z_wg=[];
167     di_wg=[]; % Dihedral spanwise
168     iw=[]; % Wing incidence chordwise
169
170     % Determine the leading edge sweep using CHSTAT
171     sweep = atan2d(tan(SAVSI_WG)-(4*(0-CHSTAT_WG)*(1-TR_wg))/(AR_wg*(1+TR_wg)));
172
173     % Determine the X and Y coordinates of the wing using taper ratio
174     % and sweep
175     for i=1:1:(wgres+1)
176         sw=(i-1)*SSPN_WG/wgres*tand(sweep);
177         for j=1:1:(wgres+1)
178             X_wg(i,j) = (((j-1)*CHRDR_WG/wgres)*(1-(i-1)*TR_wg/(wgres*TR_wg/(1-TR_wg))))+sw;
179             Y_wg(j,i) = (j-1)*SSPN_WG/wgres;
180         end
181     end
182
183     % Determine the shape of the airfoil given a TC ratio
184     % Determine the dihedral spanwise and wing incidence chordwise

```

```

59 % Determine flap deflection
60 for i=1:(wgres+1)
61     for j=1:1:(wgres+1)
62         c=X_wg(j,wgres+1)-X_wg(j,1);
63         xval=X_wg(j,i)-X_wg(j,1);
64         Z_wg(j,i)=TC_WG*5*c*(0.2969.* (xval/c).^0.5-0.126.* (xval/c) -...
65             0.3516.* (xval/c).^2+0.2843.* (xval/c).^3-0.1015.* (xval/c).^4);
66         di_wg(j,i)=(i-1)*c/wgres*tand(DHDADI_WG);
67         iw(j,i)=(i-1)*c/wgres*tand(-1*ALIW);
68         %df(j,i)=(i-1)*CHRDR_WG/wgres*tand(-1*DELTA);
69         df(j,i)=(i-1)*c/wgres*tand(-1*DELTA);
70         dal(j,i)=(i-1)*c/wgres*tand(-1*DELTAL);
71         dar(j,i)=(i-1)*c/wgres*tand(-1*DELTAR);
72         l(j)=j;
73     end
74 end
75
76 % FLAP CALCULATIONS ****
77 %Y_wg2 = Y_wg;
78
79 % Allocate memory for arrays
80 I = []; % Intermediate array
81 Iy = []; % Overflow array
82 Ii = []; % Indices of inner span
83 Io = []; % Indices of outer span
84 l1 = []; % Indices of Flap chord length
85 l2 = []; % Indices of Chord length - flap chord length
86 ind = []; % Indices where flap exists
87
88
89 % Correct flap span for MATLAB plotting
90 if SPANFO_F > (SSPN_WG-SSPN_WG/(wgres+1))
91     SPANFO_F = SPANFO_F + SSPN_WG/(wgres+1);
92 end
93
94 % Correct for outboard control surfaces
95 if ((SPANFI_F-SSPN_WG/(wgres+1)) < SPANFO_A) && (SPANFO_F > SPANFO_A)
96     SPANFI_F = SPANFI_F - SSPN_WG/(wgres+1);
97 end
98
99 % Obtain indices of wing where flap exists (along span)
100 [I,Iy] = find((Y_wg >= SPANFI_F) & (Y_wg <= SPANFO_F));
101
102 Iymin = min(I); % Inboard index value of flap (inside)
103 Iymax = max(I); % Outboard index value of flap (inside)
104 Iyminf = Iymin+1; % Inboard index value of flap (outside)
105 Iymaxf = Iymax-1; % Outboard index value of flap (outside)
106
107 % Clean up array I
108 for i=Iyminf:Iymaxf, ind((i)-(Iyminf-1))=i;, end
109
110 %Obtain Ii and Io
111 for i=1:Iymin-1, Ii(i)=i;, end
112 for i=Iymax+1:(wgres+1), Io((i-1)-(Iymax-1))=i;, end
113
114 % Chordlength at flap inboard location (used for outboard location)
115 ci=X_wg(Iymin,wgres+1)-X_wg(Iymin,1);
116
117 % Number of indices that the flaps spans across (chordwise)
118 fc=int32(CHRDFI_F/ci*wgres);
119
120 % Obtain L2 and l1
121 for i=1:fc, l2(i)=(wgres+1)-(i-1);, end
122 for i=1:(wgres)-fc, l1(i)=i;, end
123
124 Y_flap = Y_wg; % Initialize flap to the wing matrix
125
126 % The following statements core out the flap matrix to leave
127 % only the flap area
128 Y_flap(Ii,l2) = NaN;
129 Y_flap(Io,l2) = NaN;
130 Y_flap(l1,l1) = NaN;
131
132 Y_wg(ind,l2) = NaN; % Core out the flap area in the wing
133
134 % Adjust the flap deflection matrix to line up LE of flap with wing
135 %df = df - df(1,(wgres+1)-fc);
136 for i=1:(wgres+1)
137     dft = df(i,(wgres+1)-fc);
138     for j=1:(wgres+1)
139         df(i,j) = df(i,j) - dft;
140     end
141 end
142
143 % AILERON CALCULATIONS ****
144 clear I Iy Ii Io l1 l2 ind Iymin Iymax Iyminf Iymaxf ci fc
145 I = []; % Intermediate array
146 Iy = []; % Overflow array
147 Ii = []; % Indices of inner span

```

```

148 Io = [] ; % Indices of outer span
149 I1 = [] ; % Indices of aileron chord length
150 I2 = [] ; % Indices of Chord length - aileron chord length
151 ind = [] ; % Indices where aileron exists
152
153 % Correct aileron span for MATLAB plotting
154 if SPANFO_A > (SSPN_WG-SSPN_WG/(wgres+1))
155 SPANFO_A = SPANFO_A + SSPN_WG/(wgres+1);
156 end
157
158 % Correct for outboard control surfaces
159 if ((SPANFI_A-SSPN_WG/(wgres+1)) < SPANFO_F) && (SPANFO_A > SPANFO_F)
160 SPANFI_A = SPANFI_A - SSPN_WG/(wgres+1);
161 %SPANFI_A = SPANFO_F - SSPN_WG/(wgres+1);
162 end
163
164 % Obtain indices of wing where aileron exists (along span)
165 [I,Iy] = find((Y_wg >= SPANFI_A) & (Y_wg <= SPANFO_A));
166
167 Iymin = min(I); % Inboard index value of aileron (inside)
168 Iymax = max(I); % Outboard index value of aileron (inside)
169 Iyminf = Iymin+1; % Inboard index value of aileron (outside)
170 Iymaxf = Iymax-1; % Outboard index value of aileron (outside)
171
172 % Clean up array I
173 for i=Iyminf:Iymaxf, ind((i)-(Iyminf-1))=i;, end
174
175 %Obtain Ii and Io
176 for i=1:Iymin-1, Ii(i)=i;, end
177 for i=Iymax+1:(wgres+1), Io((i-1)-(Iymax-1))=i;, end
178
179 % Chordlength at aileron inboard location (used for outboard location)
180 ci=X_wg(Iymin,wgres+1)-X_wg(Iymin,1);
181
182 % Number of indices that the aileron spans across (chordwise)
183 fc=int32(CHRDFI_A/ci*wgres);
184
185 % Obtain L2 and l1
186 for i=1:fc, l2(i)=(wgres+1)-(i-1);, end
187 for i=1:(wgres)-fc, l1(i)=i;, end
188
189 Y_aileron = Y_wg; % Initialize flap to the wing matrix
190
191 % The following statements core out the aileron matrix to leave
192 % only the aileron area
193 Y_aileron(Ii,l2) = NaN;
194 Y_aileron(Io,l2) = NaN;
195 Y_aileron(1,11) = NaN;
196
197 Y_wg(ind,l2) = NaN; % Core out the aileron area in the wing
198
199 % Adjust the flap deflection matrix to line up LE of flap with wing
200 %df = df - df(1,(wgres+1)-fc);
201 for i=1:(wgres+1)
202 daLT = daL(i,(wgres+1)-fc);
203 daRT = daR(i,(wgres+1)-fc);
204 for j=1:(wgres+1)
205 daL(i,j) = daL(i,j) - daLT;
206 daR(i,j) = daR(i,j) - daRT;
207 end
208 end
209
210 % Plot the surfaces of the wing
211 % Upper and lower right, upper and lower left
212 surf(X_wg+XW,Y_wg,Z_wg+ZW+iw+di_wg,'EdgeColor','k','DisplayName','Wing Upper-Stb.')
213 surf(X_wg+XW,Y_wg,-1*Z_wg+ZW+iw+di_wg,'EdgeColor','k','DisplayName','Wing Lower-Stb.')
214 surf(X_wg+XW,-1.*Y_wg,Z_wg+ZW+iw+di_wg,'EdgeColor','k','DisplayName','Wing Upper-Port')
215 surf(X_wg+XW,-1.*Y_wg,-1*Z_wg+ZW+iw+di_wg,'EdgeColor','k','DisplayName','Wing Lower-Port')
216
217
218
219 if CHRDFI_F > 0
220 % Plot the surfaces of the flap
221 % Upper and lower right, upper and lower left
222 surf(X_wg+XW,Y_flap,Z_wg+ZW+iw+di_wg+df,'EdgeColor','r','DisplayName','Flap Upper-Stb.')
223 surf(X_wg+XW,Y_flap,-1*Z_wg+ZW+iw+di_wg+df,'EdgeColor','r','DisplayName','Flap Lower-Stb.')
224 surf(X_wg+XW,-1.*Y_flap,Z_wg+iw+di_wg+ZW+df,'EdgeColor','r','DisplayName','Flap Upper-Port')
225 surf(X_wg+XW,-1.*Y_flap,-1*Z_wg+iw+di_wg+ZW+df,'EdgeColor','r','DisplayName','Flap Lower-Port')
226
227 end
228
229 if CHRDFI_A > 0
230 % Plot the surfaces of the flap
231 % Upper and lower right, upper and lower left
232 surf(X_wg+XW,Y_aileron,Z_wg+ZW+iw+di_wg+daR,'EdgeColor','b','DisplayName','Aileron Upper-Stb.')
233 surf(X_wg+XW,Y_aileron,-1*Z_wg+ZW+iw+di_wg+daR,'EdgeColor','b','DisplayName','Aileron Lower-Stb.')
234 surf(X_wg+XW,-1.*Y_aileron,Z_wg+iw+di_wg+ZW+dal,'EdgeColor','b','DisplayName','Aileron Upper-Port')
235 surf(X_wg+XW,-1.*Y_aileron,-1*Z_wg+iw+di_wg+ZW+dal,'EdgeColor','b','DisplayName','Aileron Lower-Port')
236 end

```

```

237 else
238 % Inboard and outboard panel geometry
239
240 wgreso=int32(SSPNOP_WG/SSPN_WG*wgres);
241 wgresi=wgres-wgreso;
242
243 % Inboard and outboard aspect ratios
244 ARI_wg=2*((SSPN_WG-SSPNOP_WG)*2)/(CHRDR_WG+CHRDBP_WG);
245 ARo_wg=2*(SSPNOP_WG*2)/(CHRDBP_WG+CHRDTP_WG);
246
247 % Inboard and outboard taper ratios
248 TRI_wg=CHRDBP_WG/CHRDR_WG;
249 TRo_wg=CHRDTP_WG/CHRDBP_WG;
250
251 % Allocate memory for arrays
252 Xi_wg=[];
253 Xo_wg=[];
254 Yi_wg=[];
255 Yo_wg=[];
256 Zi_wg=[];
257 Zo_wg=[];
258 dii_wg=[];
259 dio_wg=[];
260 iwi=[];
261 iwo=[];
262
263 %Inboard and outboard leading edge sweep
264 sweepi = atand(tand(SAVSI_WG)-(4*(0-CHSTAT_WG)*(1-TRI_wg))/(ARI_wg*(1+TRI_wg)));
265 sweepo = atand(tand(SAVSO_WG)-(4*(0-CHSTAT_WG)*(1-TRo_wg))/(ARo_wg*(1+TRo_wg)));
266
267 % Determine the X and Y coordinates of the inboard planform using
268 % taper ratio and sweep
269 for i=1:1:(wgres+1)
270     sw=(i-1)*(SSPN_WG-SSPNOP_WG)/wgres*tand(sweepi);
271     for j=1:1:(wgres+1)
272         Xi_wg(i,j) = (((j-1)*CHRDR_WG/wgres)*(1-(i-1)*TRI_wg/(wgres*TRI_wg/(1-TRI_wg)))+sw;
273         Yi_wg(j,i) = (j-1)*(SSPN_WG-SSPNOP_WG)/wgres;
274     end
275 end
276
277 % Determine the shape of the airfoil given a TC ratio
278 % Determine the dihedral spanwise and wing incidence chordwise
279 % For inboard planform
280 for i=1:1:(wgres+1)
281     for j=1:1:(wgres+1)
282         c=Xi_wg(j,wgres+1)-Xi_wg(j,1);
283         xval=Xi_wg(j,i)-Xi_wg(j,1);
284         Zi_wg(j,i)=TC_WG*5*c*(0.2969.*((xval/c).^0.5-0.126.*((xval/c)-...
285             0.3516.*((xval/c).^2+0.2843.*((xval/c).^3-0.1015.*((xval/c).^4));
286         dii_wg(j,i)=(j-1)*(SSPN_WG-SSPNOP_WG)/wgres*tand(DHDADI_WG);
287         iwi(j,i)=(i-1)*c/wgres*tand(-1*ALIW);
288     end
289 end
290
291 % Determine the X and Y coordinates of the outboard planform using
292 % taper ratio and sweep
293 for i=1:1:(wgres+1)
294     sw=(i-1)*(SSPNOP_WG)/wgres*tand(sweepo);
295     for j=1:1:(wgres+1)
296         Xo_wg(i,j) = (((j-1)*CHRDBP_WG/wgres)*(1-(i-1)*TRo_wg/(wgres*TRo_wg/(1-TRo_wg)))+sw+Xi_wg(wgres+1,1);
297         Yo_wg(j,i) = (j-1)*(SSPNOP_WG)/wgres + (SSPN_WG-SSPNOP_WG);
298     end
299 end
300
301 % Determine the shape of the airfoil given a TC ratio
302 % Determine the dihedral spanwise and wing incidence chordwise
303 % For outboard planform
304 for i=1:1:(wgres+1)
305     for j=1:1:(wgres+1)
306         c=Xo_wg(j,wgres+1)-Xo_wg(j,1);
307         xval=Xo_wg(j,i)-Xo_wg(j,1);
308         Zo_wg(j,i)=TC_WG*5*c*(0.2969.*((xval/c).^0.5-0.126.*((xval/c)-...
309             0.3516.*((xval/c).^2+0.2843.*((xval/c).^3-0.1015.*((xval/c).^4));
310         dio_wg(j,i)=(j-1)*(SSPNOP_WG)/wgres*tand(DHDADO_WG);
311         iwo(j,i)=(i-1)*c/wgres*tand(-1*ALIW);
312     end
313 end
314
315 % Plot the surfaces of the inboard planform
316 % Upper and lower right, upper and lower left
317 surf(Xi_wg*XW,Yi_wg,Zi_wg+ZW+iwi+dii_wg,'EdgeColor','k','DisplayName','Wing In. Upper-Stb.')
318 surf(Xi_wg*XW,Yi_wg,-1.*Zi_wg+ZW+iwi+dii_wg,'EdgeColor','k','DisplayName','Wing In. Lower-Stb.')
319 surf(Xi_wg*XW,-1.*Yi_wg,Zi_wg+ZW+iwi+dii_wg,'EdgeColor','k','DisplayName','Wing In. Upper-Port')
320 surf(Xi_wg*XW,-1.*Yi_wg,-1.*Zi_wg+ZW+iwi+dii_wg,'EdgeColor','k','DisplayName','Wing In. Lower-Port')
321
322 %surf2stl('ala1.stl',Xi_wg*XW,Yi_wg,Zi_wg+ZW+iwi+dii_wg);
323 %surf2stl('ala2.stl',Xi_wg*XW,Yi_wg,-1.*Zi_wg+ZW+iwi+dii_wg);
324 %surf2stl('ala3.stl',Xi_wg*XW,-1.*Yi_wg,Zi_wg+ZW+iwi+dii_wg);

```

```

325 %surf2stl('ala4.stl',Xi_wg+XW,-1.*Yi_wg,-1.*Zi_wg+ZW+iwi+dii_wg);
326 % Plot the surfaces of the outboard planform
327 % Upper and lower right, upper and lower left
328 surf(Xo_wg+XW,Yo_wg,Zo_wg+ZW+iwo+(SSPN_WG-SSPNOP_WG)*tand(DHDADI_WG)+dio_wg,'EdgeColor','k','DisplayName','
329 Wing Out. Upper-Stb.')
330 surf(Xo_wg+XW,Yo_wg,-1.*Zo_wg+ZW+iwo+(SSPN_WG-SSPNOP_WG)*tand(DHDADI_WG)+dio_wg,'EdgeColor','k','
331 DisplayName','Wing Out. Lower-Stb.')
332 surf(Xo_wg+XW,-1.*Yo_wg,Zo_wg+ZW+iwo+(SSPN_WG-SSPNOP_WG)*tand(DHDADI_WG)+dio_wg,'EdgeColor','k','
333 DisplayName','Wing Out. Upper-Port')
334 surf(Xo_wg+XW,-1.*Yo_wg,-1.*Zo_wg+ZW+iwo+(SSPN_WG-SSPNOP_WG)*tand(DHDADI_WG)+dio_wg,'EdgeColor','k','
335 DisplayName','Wing Out. Lower-Port')
336 %surf2stl('ala5.stl',Xo_wg+XW,Yo_wg,Zo_wg+ZW+iwo+(SSPN_WG-SSPNOP_WG)*tand(DHDADI_WG)+dio_wg);
337 %surf2stl('ala6.stl',Xo_wg+XW,Yo_wg,-1.*Zo_wg+ZW+iwo+(SSPN_WG-SSPNOP_WG)*tand(DHDADI_WG)+dio_wg);
338 %surf2stl('ala7.stl',Xo_wg+XW,-1.*Yo_wg,Zo_wg+ZW+iwo+(SSPN_WG-SSPNOP_WG)*tand(DHDADI_WG)+dio_wg);
339 %surf2stl('ala8.stl',Xo_wg+XW,-1.*Yo_wg,-1.*Zo_wg+ZW+iwo+(SSPN_WG-SSPNOP_WG)*tand(DHDADI_WG)+dio_wg);
337 end

```

Tali funzioni sono state modificate da quelle originali trovate in rete con l'aggiunta della funzione `surf2stl`, necessaria a trasformare i plot tridimensionali di MATLAB (definiti come superfici `surf`), in file `.stl` (stereolitografici) separati.

Si è poi utilizzato il software di modellazione 3D "Blender" per unire i vari pezzi dell'aereo. E' stato così possibile produrre un modello 3D in formato `.stl` utilizzabile per i codici di simulazione della traiettoria del velivolo, presenti nei successivi capitoli.

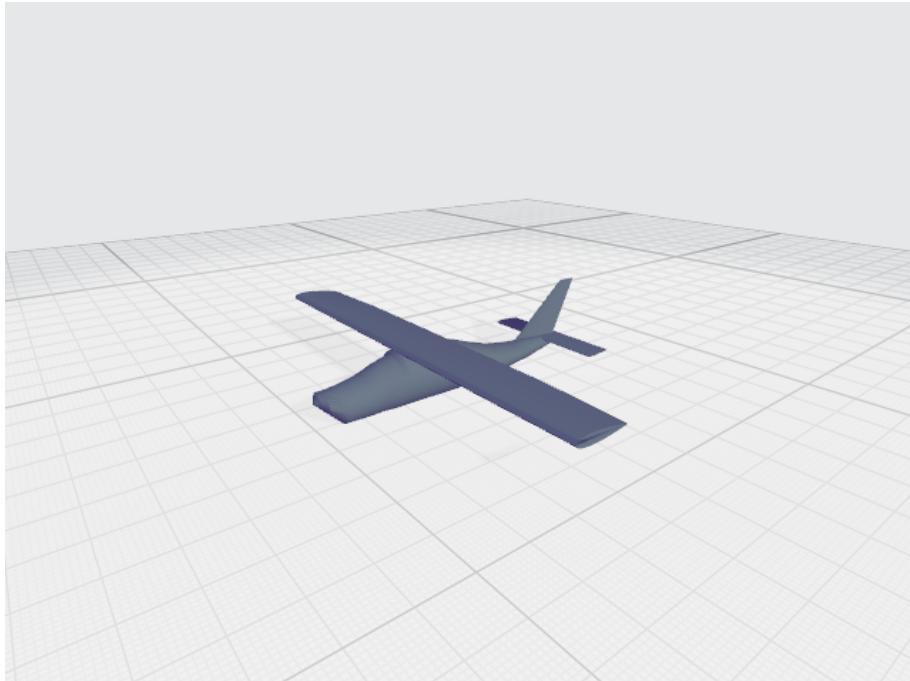


Figura 1.9: Modello 3D

Capitolo 2

Equazioni generali del moto di un velivolo

2.1 Intro

In questo capitolo si pone come obiettivo la simulazione non lineare delle prestazioni di un velivolo rigido. Per tale scopo osserviamo alcuni aspetti dei moti relativi. Nella Figura 2.1 è mostrato il moto di un velivolo rispetto a una terna di riferimento inerziale. Tale terna fissa viene indicata come $\mathcal{T}_E = (O, x, y, z)_E$, supponendo le terne solidali alla Terra (Earth) come terne fisse. Consideriamo anche una terna mobile $\mathcal{T}_m = (C, x, y, z)_m$. Nello studio di un aereo come in Figura 2.1 sceglieremo una terna $\mathcal{T}_B = (C, x, y, z)_B$ solidale al velivolo. Parliamo quindi di terna degli assi velivolo (body-fixed reference frame).

Per la simulazione utilizziamo un modello matematico ridotto rispetto alle equazioni generali del moto. Le equazioni a cui si fa riferimento sono le 2.1.

$$\begin{aligned} \frac{W}{g} \left(\begin{Bmatrix} \dot{V}_{Gx_m} \\ \dot{V}_{Gy_m} \\ \dot{V}_{Gz_m} \end{Bmatrix} + \begin{bmatrix} 0 & -\Omega_{z_m} & \Omega_{y_m} \\ \Omega_{z_m} & 0 & -\Omega_{x_m} \\ -\Omega_{y_m} & \Omega_{x_m} & 0 \end{bmatrix} \begin{Bmatrix} V_{Gx_m} \\ V_{Gy_m} \\ V_{Gz_m} \end{Bmatrix} \right) &= \begin{Bmatrix} F_{x_m} \\ F_{y_m} \\ F_{z_m} \end{Bmatrix} \\ \begin{Bmatrix} \dot{K}_{r_{x_m}} \\ \dot{K}_{r_{y_m}} \\ \dot{K}_{r_{z_m}} \end{Bmatrix} + \begin{bmatrix} 0 & -\Omega_{z_m} & \Omega_{y_m} \\ \Omega_{z_m} & 0 & -\Omega_{x_m} \\ -\Omega_{y_m} & \Omega_{x_m} & 0 \end{bmatrix} \begin{Bmatrix} K_{r_{x_m}} \\ K_{r_{y_m}} \\ K_{r_{z_m}} \end{Bmatrix} + \frac{W}{g} \begin{bmatrix} 0 & -z_G & y_G \\ z_G & 0 & x_G \\ y_G & x_G & 0 \end{bmatrix} \begin{Bmatrix} a_{C_{x_m}} \\ a_{C_{y_m}} \\ a_{C_{z_m}} \end{Bmatrix} &= \begin{Bmatrix} \mathcal{M}_{x_m} \\ \mathcal{M}_{y_m} \\ \mathcal{M}_{z_m} \end{Bmatrix} \end{aligned} \quad (2.1)$$

Che possono essere riscritte come le eq. 2.2.

$$\begin{aligned} \frac{W}{g} \left(\{\dot{V}_G\}_m + [\tilde{\Omega}_m]_m \{V_G\}_m \right) &= \{F\}_m \\ \{\dot{K}_r\}_m + [\tilde{\Omega}_m]_m \{K_r\} + \frac{W}{g} [\tilde{r}_G]_m \{a_C\}_m &= \{\mathcal{M}\}_m \end{aligned} \quad (2.2)$$

Quest'ultima, assumendo come origine della terna mobile e come polo di riferimento dei momenti il baricentro G, si semplifica come la

$$\{\dot{K}_r\}_m + [\tilde{\Omega}_m]_m \{K_r\} = \{\mathcal{M}\}_m \quad (2.3)$$

Inoltre si ipotizzerà che l'aereo possa volare anche in presenza di vento costante presumendo che la massa d'aria sia in moto di traslazione uniforme rispetto alla terra con velocità \mathbf{V}_w . La presenza di vento rende la velocità della *corrente relativa* pari a $\mathbf{V}_\infty = \mathbf{V}_w - \mathbf{V}_G$, in cui \mathbf{V}_G è la velocità vera del velivolo rispetto alla Terra. Verra così particolarizzata la prima delle 2.2 come nella 2.4.

$$\frac{W}{g} \left(\{\dot{V}_G\}_A + [\tilde{\Omega}_A]_A \{V_G\}_A \right) = \{F\}_A \quad (2.4)$$

Per particolarizzare l'equazione 2.4, si faranno le seguente ipotesi semplificative:

- angolo di derapata costantemente nullo $\beta = 0$

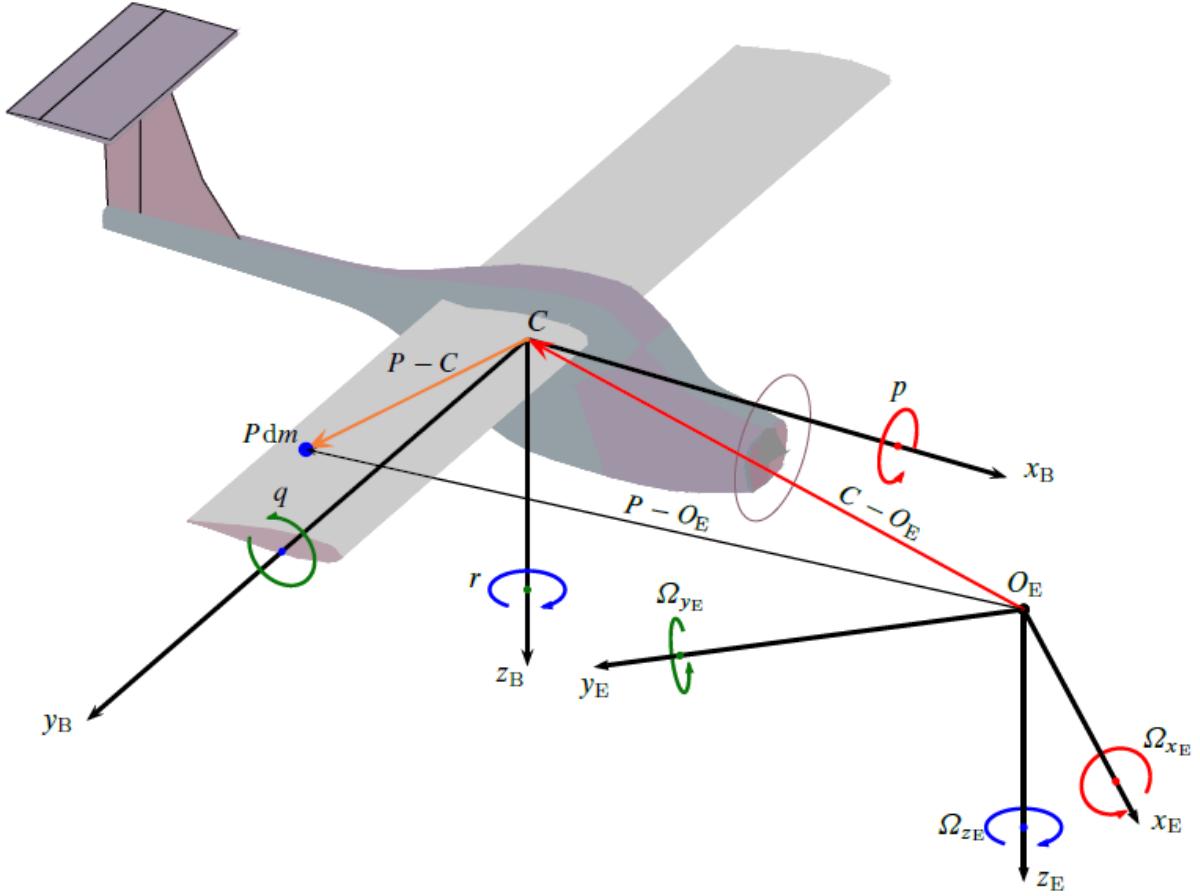


Figura 2.1: Terna

- vettore di spinta allineato costantemente con la velocità
- angolo d'attacco α_B piccolo

In questo modo la trasformazione dagli assi Terra (Earth) agli assi Velivolo (Body) passano da:

$$\mathcal{T}_E \xrightarrow{\delta, \gamma} \mathcal{T}_W \xrightarrow{\nu, -\beta} \mathcal{T}_A \xrightarrow{\alpha_B} \mathcal{T}_B$$

Alla seguente:

$$\mathcal{T}_E \xrightarrow{\delta, \gamma} \mathcal{T}_W \xrightarrow{\nu, -\beta=0} \mathcal{T}_A \approx \mathcal{T}_B$$

Per cui si confonde la terna di assi velivolo con quella di assi aerodinamici. E' ora possibile ottenere delle semplificazioni, difatti si possono esprimere le componenti della velocità, della forza aerodinamica e della forza propulsiva come nell'equazione (2.5)

$$\{V_G\}_A = \begin{Bmatrix} V \\ 0 \\ 0 \end{Bmatrix}, \quad \{F_A\}_A = \begin{Bmatrix} -D \\ 0 \\ -L \end{Bmatrix}, \quad \{F_T\}_A = \begin{Bmatrix} T \\ 0 \\ 0 \end{Bmatrix} \quad (2.5)$$

Dall'ipotesi di $\beta = 0$ segue che la forza laterale aerodinamica è nulla $Y_A = 0$, mentre dall'ipotesi per cui $\alpha_B \approx 0$ si può dire che $V_{infty} \approx V$. Calcolando il vettore velocità angolare degli assi aerodinamici mediante la definizione divettore di evoluzione $\Omega_W \equiv \mathbf{E} = \dot{\delta} \sin \gamma \mathbf{i}_W + \dot{\gamma} \mathbf{j}_W + \dot{\delta} \cos \gamma \mathbf{k}_W$ e data l'approssimazione $\mathcal{T}_A = \mathcal{T}_B$. Dunque la velocità angolare del velivolo è:

$$\Omega_A \approx \Omega_B \quad (2.6)$$

Effettuando alcuni passaggi matematici si ottengono le tre componenti della velocità istantanea nella terna \mathcal{T}_A :

$$\begin{cases} p^A \triangleq \Omega_{B,x_A} = \dot{\nu} - \dot{\delta} \sin\gamma \\ q^A \triangleq \Omega_{B,y_A} = \dot{\gamma} \cos\nu - \dot{\delta} \sin\nu \cos\gamma \\ r^A \triangleq \Omega_{B,z_A} = \dot{\delta} \cos\nu \cos\gamma - \dot{\gamma} \sin\nu \end{cases} \quad (2.7)$$

A questo punto la prima delle (2.1) diventa la (2.8).

$$m \left(\begin{Bmatrix} \dot{V} \\ 0 \\ 0 \end{Bmatrix} + \begin{bmatrix} 0 & -r^A & q^A \\ r^A & 0 & -p^A \\ -q^A & p^A & 0 \end{bmatrix} \begin{Bmatrix} V \\ 0 \\ 0 \end{Bmatrix} \right) = [R(1, \nu)][R(2, \gamma)][R(3, \delta)] \begin{Bmatrix} 0 \\ 0 \\ mg \end{Bmatrix} + \begin{Bmatrix} -D \\ 0 \\ -L \end{Bmatrix} + \begin{Bmatrix} T \\ 0 \\ 0 \end{Bmatrix} \quad (2.8)$$

In cui:

$$[T_{AE}] = [R(1, \nu)][R(2, \gamma)][R(3, \delta)] \quad (2.9)$$

Sostituendo i valori dell'equazione (2.8) ed esplicitando questa matrice di rotazione si ottiene un importante risultato, le **Nonlinear Aircraft Performance Equations (NAPE)**, che rappresentano le leggi della dinamica traslazionale del velivolo che mantiene un angolo di derapata costantemente nullo e un vettore di spinta sempre allineato con il vettore velocità e con angolo di attacco piccolo:

$$\begin{cases} \dot{V} = \frac{T-D}{m} - g \sin\gamma \\ \dot{\gamma} = \frac{1}{mV} L \cos\nu - mg \cos\gamma \\ \dot{\delta} = \frac{L \sin\nu}{mV \cos\gamma} \end{cases} \quad (2.10)$$

In cui con T-D idnichiamo l'esubero di spinta. Possiamo anche tenere di una massa variabile del velivolo dovuta al consumo di combustibile tramite la seguente equazione

$$\dot{m} = \frac{W_{fuel}}{g} = -K_W T \quad (2.11)$$

In cui K_W è lo *specific fuel consumption*, dato dal rapporto tra libbre di combustibile utilizzato e libbre di spinta, moltiplicate per le ore considerate. K_W dipende dal particolare sistema propulsivo installato e lo considereremo costante. Le equazioni dinamiche ricavate vanno accoppiate a delle equazioni cinematiche di navigazione che descrivono l'evoluzione della traiettoria del baricentro del velivolo in assi Terra. Detta $[T_{EA}] = [T_{EA}]^{-1}$ la matrice di trasformazione delle componenti del vettore velocità da assi aerodinamici ad assi terra, le equazioni di navigazione sono le seguenti.

$$\{V_G\}_E = \begin{Bmatrix} \dot{x}_{E,G} \\ \dot{y}_{E,G} \\ \dot{z}_{E,G} = -\dot{h} \end{Bmatrix} = [T_{EA}]\{V_G\}_E = [T_{EA}] \begin{Bmatrix} V \\ 0 \\ 0 \end{Bmatrix} \quad (2.12)$$

Che dopo le dovute sostituzioni diventa la (2.13).

$$\begin{cases} \dot{x}_{E,G} = V \cos\gamma \cos\delta \\ \dot{y}_{E,G} = V \cos\gamma \sin\delta \\ \dot{h} = V \sin\gamma \end{cases} \quad (2.13)$$

Nel sistema di equazioni di evoluzione si riconoscono le variabili di stato

$$[V, \gamma, \delta, m, x_{E,G}, y_{E,G}, h]$$

le cui storie temporali sono determinate dalle funzioni di ingresso $T(t)$, $L(t)$, $\nu(t)$. La resistenza può considerarsi funzione della portanza:

$$D = K_{D_0} \rho V_\infty^2 + K_{D_1} \frac{L^2}{\rho V_\infty^2} \quad (2.14)$$

In cui:

$$K_{D_0} = \frac{1}{2} S C_{D_0}, \quad K_{D_1} = \frac{2}{S \pi A R e_{eff}} \quad (2.15)$$

La (2.14) discende dall'assunzione di una *polare di resistenza parabolica*:

$$D \triangleq \frac{1}{2} S C_D = \frac{1}{2} \rho V_\infty^2 S \left(C_{D_0} + \frac{C_L^2}{\pi A R e_{eff}} \right) \quad (2.16)$$

In cui C_{D_0} è il coefficiente di resistenza parassita e e_{eff} è il fattore di Oswald del velivolo.

Il coefficiente di portanza è:

$$C_L \triangleq \frac{L}{\frac{1}{2} \rho V_\infty^2 S} = C_{L_\alpha} (\alpha_B - \alpha_{0L}) \quad (2.17)$$

, dove C_{L_α} è la pendenza della retta di portanza e α_{0L} è l'angolo d'attacco di portanza nulla.

Si noti che $V_\infty \neq V$, cioè portanza e resistenza dipendono dalla velocità del vento relativo. Nel moto in presenza di vento assegnato:

$$\{V_W\}_E = [V_{x_{E,W}}, V_{y_{E,W}}, V_{z_{E,W}}]^T \quad (2.18)$$

In cui:

$$\begin{aligned} V_{x_{E,\infty}} &= V_{x_{E,W}} - \dot{x}_{E,W} \\ V_{y_{E,\infty}} &= V_{y_{E,W}} - \dot{y}_{E,W} \\ V_{z_{E,\infty}} &= V_{z_{E,W}} + \dot{h} \end{aligned} \quad (2.19)$$

Quindi:

$$\begin{aligned} V_{x_{E,\infty}}^2 &= V_{x_{E,W}}^2 + V_{y_{E,W}}^2 + V_{z_{E,W}}^2 = \\ &= (V_{x_{E,W}} - V \cos\gamma \cos\delta)^2 + (V_{y_{E,W}} - V \cos\gamma \sin\delta)^2 + (V_{z_{E,W}} + V \sin\gamma)^2 \end{aligned} \quad (2.20)$$

Questo modello matematica si completa con la formula di calcolo dell'angolo di attacco che si ricava esprimendo α_B in termini della funzione di controllo $L(t)$ e del valore istantaneo di V_∞^2 . Invertendo la (2.17) si ottiene:

$$\alpha_B(t) = K_L \frac{L}{\rho V_\infty^2} + \alpha_{0L} \quad \text{con } K_L = \frac{2}{S C_{L_\alpha}} \quad (2.21)$$

L'equazione (2.21) rappresenta l'angolo d'attacco necessario per determinare l'intensità voluta della portanza per il controllo della pendenza della traiettoria nonché, per effetto del suo legame con la resistenza, per il controllo della velocità.

2.2 Esercizio 6.5

Il velivolo considerato è un Tecnam P2010, mostrato nella Figura 2.2, i dati utilizzati sono in Tabella 2.2. I dati riguardanti l'atmosfera sono presenti in Tabella 2.1, mentre i valori iniziali e comandati sono riportati nella Tabella 2.3. Si è anche in presenza di vento, le sue componenti sono le seguenti:

$$V_{x_{E,W}} = 15 \text{ mph}, \quad V_{x_{E,W}} = 23 \text{ mph}, \quad V_{x_{E,W}} = 6 \text{ mph}$$

L'obiettivo è quello di assegnare opportune leggi di comando $T(t)$, $L(t)$, $\nu(t)$ affinché il velivolo in volo livellato iniziale diretto verso Nord alla quota di 10000 ft e alla velocità di 100 mph effettui una serie di manovre, seguendo i comandi mostrato nella Tabella 2.3. La simulazione viene eseguita tramite il software Simulink, i grafici e la traiettoria in 3D del velivolo vengono eseguiti utilizzando codici MATLAB.



Figura 2.2: Tecnam P2010

Modello Atmosferico ISA			
Quota di volo (ASL)	h_0	3048 m	10000 ft
Densità dell'aria	ρ_0	0.653 kg/m ³	0.00127 slug/ft ³

Tabella 2.1: Modello Atmosferico ISA

Velivolo			
Velocità minima	V_{Min}	26.8 m/s	60 mph
Velocità massima	V_{Max}	71.5 m/s	160 mph
Altitudine operativa massima	h_{Max}	3600 m	12000 ft
Superficie Alare	S	13.9 m	150 ft ²
Apertura Alare	b	10.3 m	33.8 ft
Allungamento Alare	$\mathcal{A}R$	7.63	
Massa	m_0	915 kg	2000 lbf

Tabella 2.2: Dati Velivolo - Tecnam P2010

Ai fini dell'esercizio sono state ipotizzate valide le seguenti funzioni di trasferimento tra la spinta comandata e la spinta istantanea:

$$\frac{\mathcal{L}[T(t)]}{\mathcal{L}[T_c(t)]} \triangleq \frac{T(s)}{T_c(s)} = \frac{p_T}{s + p_T} \quad o \text{ anche} \quad \dot{T} = p_T(T_c - T) \quad (2.22)$$

(dove $\mathcal{L}[\cdot]$ è la trasformata di Laplace), tra la portanza comandata e la portanza istantanea:

$$\frac{\mathcal{L}[L(t)]}{\mathcal{L}[L_c(t)]} \triangleq \frac{L(s)}{L_c(s)} = \frac{p_L}{s + p_L} \quad o \text{ anche} \quad \dot{L} = p_L(L_c - L) \quad (2.23)$$

e tra l'angolo di *bank* comandato e quello effettivo:

$$\frac{\mathcal{L}[\nu(t)]}{\mathcal{L}[\nu_c(t)]} \triangleq \frac{\nu(s)}{\nu_c(s)} = \frac{p_\nu}{s + p_\nu} \quad o \text{ anche} \quad \dot{\nu} = p_\nu(\nu_c - \nu) \quad (2.24)$$

Comandi iniziali				
Velocità	V_0	45 m/s	100 mph	
Angolo di salita (Flight Path)	γ_0	0 deg	0 rad	
Angolo di prua (Heading)	δ_0	0 deg	0 rad	
Comandi al tempo t=5s				
Velocità	$V_{c,0}$	67 m/s	150 mph	
Angolo di salita (Flight Path)	$\gamma_{c,0}$	3 deg	0.052 rad	
Angolo di prua (Heading)	$\delta_{c,0}$	15 deg	0.0262 rad	
Rateo di salita (Rate of Climb)	$\dot{h}_{c,0}$	3.5 m/s	7.85 mph	
Comandi al tempo t=90s				
Velocità	$V_{c,1}$	54 m/s	120 mph	
Angolo di salita (Flight Path)	$\gamma_{c,1}$	0 deg	0 rad	
Angolo di prua (Heading)	$\delta_{c,1}$	15 deg	0.0262 rad	
Rateo di salita (Rate of Climb)	$\dot{h}_{c,1}$	0 m/s	0 mph	
Comandi al tempo t=150s				
Velocità	$V_{c,2}$	45 m/s	100 mph	
Angolo di salita (Flight Path)	$\gamma_{c,2}$	-5 deg	-0.087 rad	
Angolo di prua (Heading)	$\delta_{c,2}$	0 deg	0 rad	
Rateo di salita (Rate of Climb)	$\dot{h}_{c,2}$	-5.8 m/s	-13 mph	
Comandi al tempo t=240s				
Velocità	$V_{c,3}$	45 m/s	100 mph	
Angolo di salita (Flight Path)	$\gamma_{c,3}$	0 deg	0 rad	
Angolo di prua (Heading)	$\delta_{c,3}$	0 deg	0 rad	
Rateo di salita (Rate of Climb)	$\dot{h}_{c,3}$	0 m/s	0 mph	

Tabella 2.3: Valori comandati in volo

In tali equazioni p_T , p_L , p_ν sono delle costanti di tempo caratteristiche del motore e dell'aerodinamica del velivolo. Inoltre, le riposte ai rispettivi segnali comandati possono essere limitate come segue:

$$0 \leq T \leq T_{max}, \quad L \leq \rho V_\infty^2 \left(\frac{\alpha_B - \alpha_{0L}}{K_L} \right)_{max}, \quad -\nu_{max} \leq \nu \leq \nu_{max} \quad (2.25)$$

Le leggi di comando che controllano la navigazione sono le funzioni $T_c(t)$, $L_c(t)$, $\nu_c(t)$ ottenute con una logica di controllo proporzionale-integrale (PI), retroazionando i segnali $V(t)$, $\gamma(t)$ e $\delta(t)$ e inseguendo l'annullarsi delle funzioni di errore:

$$E_V(t) = V_c - V(t), \quad E_{\dot{h}}(t) = V_c [\sin \gamma_c - \sin \gamma(t)], \quad E_\delta(t) = \delta_c - \delta(t) \quad (2.26)$$

Per la spinta comandata si ipotizza la seguente funzione di trasferimento:

$$\frac{T_c(s)}{E_V(s)} = \frac{m K_{T_P} (s + K_{T_1}/K_{T_P})}{s} \quad (2.27)$$

Per la portanza comandata invece:

$$\frac{L_c(s)}{E_{\dot{h}}(s)} = \frac{m K_{L_P} (s + K_{L_1}/K_{L_P})}{s} \quad (2.28)$$

in cui $E_{\dot{h}}(s) = \mathcal{L}[E_{\dot{h}}(t)]$. Analogamente per l'angolo di bank:

$$\frac{\nu(s)}{E_\delta(s)} = K_{\nu_P} \frac{V_c}{g} \quad (2.29)$$

Le leggi di controllo della navigazione (equazioni 2.27, 2.28, 2.29) completano il modello matematico. I guadagni K_{T_P} , K_{T_1} , K_{L_P} , K_{L_1} , K_{ν_P} sono costanti caratteristiche del velivolo considerato. Di seguito viene mostrato lo schema del progetto Simulink.

2.3 Modello Simulink

Per inizializzare il modello è necessario il seguente listato, basato sulle considerazione fatte nella scorsa sezione.

```

1 wingSurface = 13.9 * 10.76 * str2u('ft^2');
2 wingSpan    = 10.3 * 3.28 * str2u('ft');
3 aspectRatio = wingSpan^2/wingSurface;
4 oswald      = 0.95;
5
6 aspectRatioE = aspectRatio*oswald;
7 piARE        = pi*aspectRatioE;
8 cDO          = 0.02;
9 weightMax    = 2557 * str2u('lbf');
10 weightEmpty  = 1687 * str2u('lbf');
11 weightMaxFuel = 440 * str2u('lbf');
12 weightNom   = weightEmpty + 0.75*weightMaxFuel;
13 kWdot       = 4e-6 * str2u('slug/(lbf*s)');
14 speedMin    = 60 * str2u('mph');
15 speedMax    = 160 * str2u('mph');
16 thrustMax   = 363 * str2u('lbf');
17 bankAngleMax = 30 * str2u('deg');
18 altitude_0  = 10000 * str2u('ft');
19 [temp_0, a_0, p_0, rho_0] = atmosisa(altitude_0/u.m);
20 airDensity_0 = rho_0 * str2u('kg/m^3');
21 kD0          = 0.5*cDO*wingSurface;
22 kD1          = 2/(piARE*wingSurface);
23 cLalpha      = 0.1 / str2u('deg');
24 alphaOL      = -2.86 * str2u('deg');
25 kL = 2.0/(wingSurface * cLalpha);
26 alphaBody_max = 8 * str2u('deg');
27 limitLMaxOverRhoV2 = (alphaBody_max - alphaOL)/kL;
28 wind_0       = [15, 23, 6] * str2u('mph');
29 weight_0     = weightNom;
30 mass_0       = weight_0 / (9.81 * str2u('m/s^2'));
31 speed_0      = 100 * str2u('mph');
32 flightPathAngle_0 = 0 * str2u('deg');
33 headingAngle_0 = 0 * str2u('deg');
34 xE_0         = 0 * str2u('m');
35 yE_0         = 0 * str2u('m');
36 altitude_0  = 10000 * str2u('ft');
37 bankAngle_0  = 0 * str2u('deg');
38 lift_0       = weight_0*cos(flightPathAngle_0)/cos(bankAngle_0);
39 drag_0       = kD0 * airDensity_0 * speed_0^2 ...
40           + kD1 * (lift_0^2)/(airDensity_0 * speed_0^2);
41
42 thrust_0     = drag_0 + weight_0*sin(flightPathAngle_0);
43 pT = 2.0 * str2u('rad/s');
44 pL = 2.5 * str2u('rad/s');
45 pNu = 1.0 * str2u('rad/s');
46 kTI = 0.002 / str2u('s^2');
47 kTP = 0.080 / str2u('s');
48 kLI = 0.010 / str2u('s^2');
49 kLP = 0.500 / str2u('s');
50 kNu = 0.075 / str2u('s');
51 t0            = 5*str2u('s');
52 speed_c       = 150 * str2u('mph');
53 flightPathAngle_c = 3 * str2u('deg');
54 hDot_c        = speed_c * sin(flightPathAngle_c);
55 headingAngle_c = 15 * str2u('deg');
56
57 t1            = 90*str2u('s');
58 speed_c_1     = 120 * str2u('mph');
59 flightPathAngle_c_1 = 0 * str2u('deg');
60 hDot_c_1      = speed_c * sin(flightPathAngle_c_1);
61 headingAngle_c_1 = 15 * str2u('deg');
62
63 t2            = 150*str2u('s');
64 speed_c_2     = 100 * str2u('mph');
65 flightPathAngle_c_2 = -5 * str2u('deg');
66 hDot_c_2      = speed_c * sin(flightPathAngle_c_2);
67 headingAngle_c_2 = 0 * str2u('deg');
68
69 t3            = 240*str2u('s');
70 speed_c_3     = 100 * str2u('mph');
71 flightPathAngle_c_3 = 0 * str2u('deg');
72 hDot_c_3      = speed_c * sin(flightPathAngle_c_3);
73 headingAngle_c_3 = 0 * str2u('deg');
```

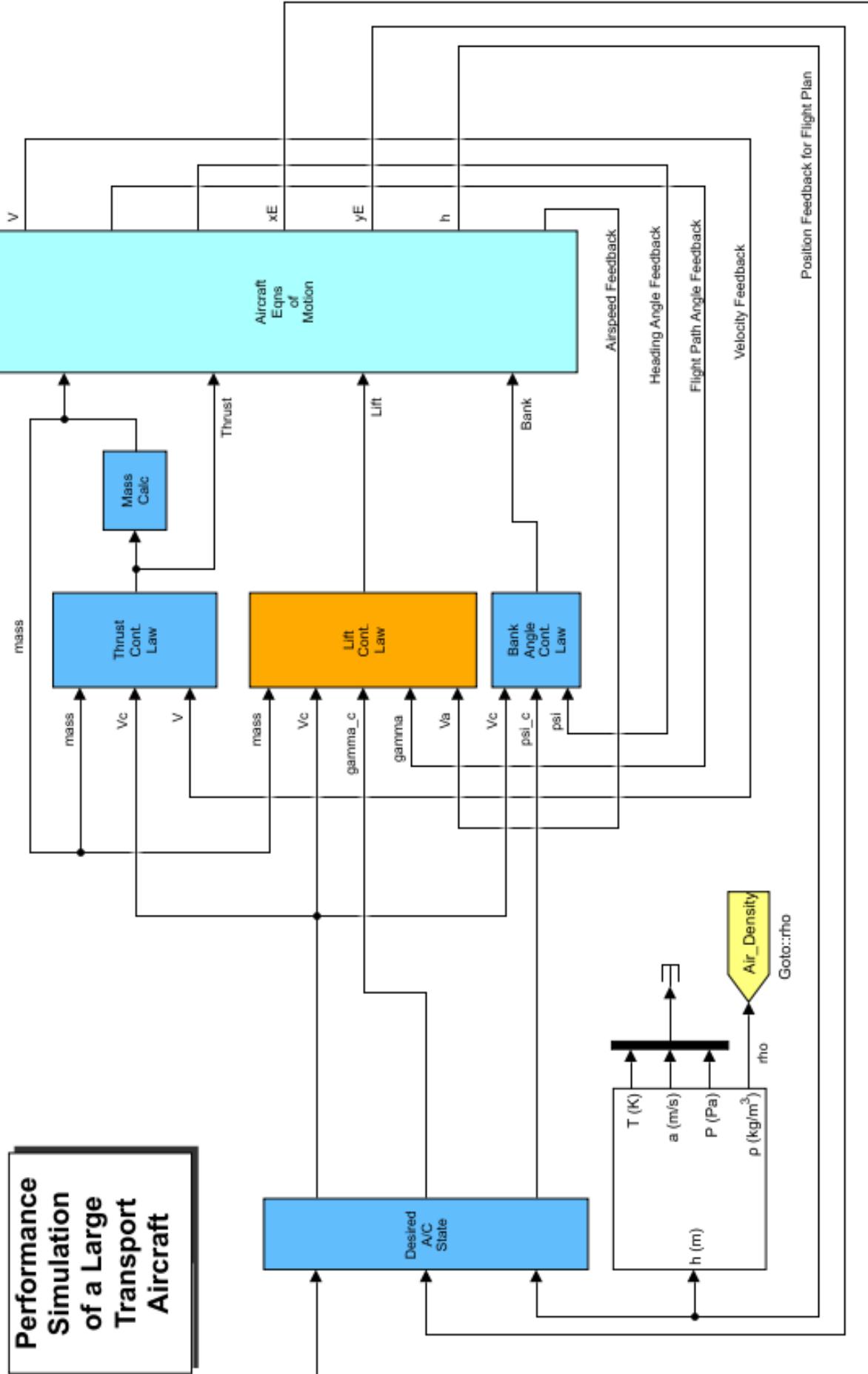


Figura 2.3: Modello Simulink

Aircraft Equations of Motion

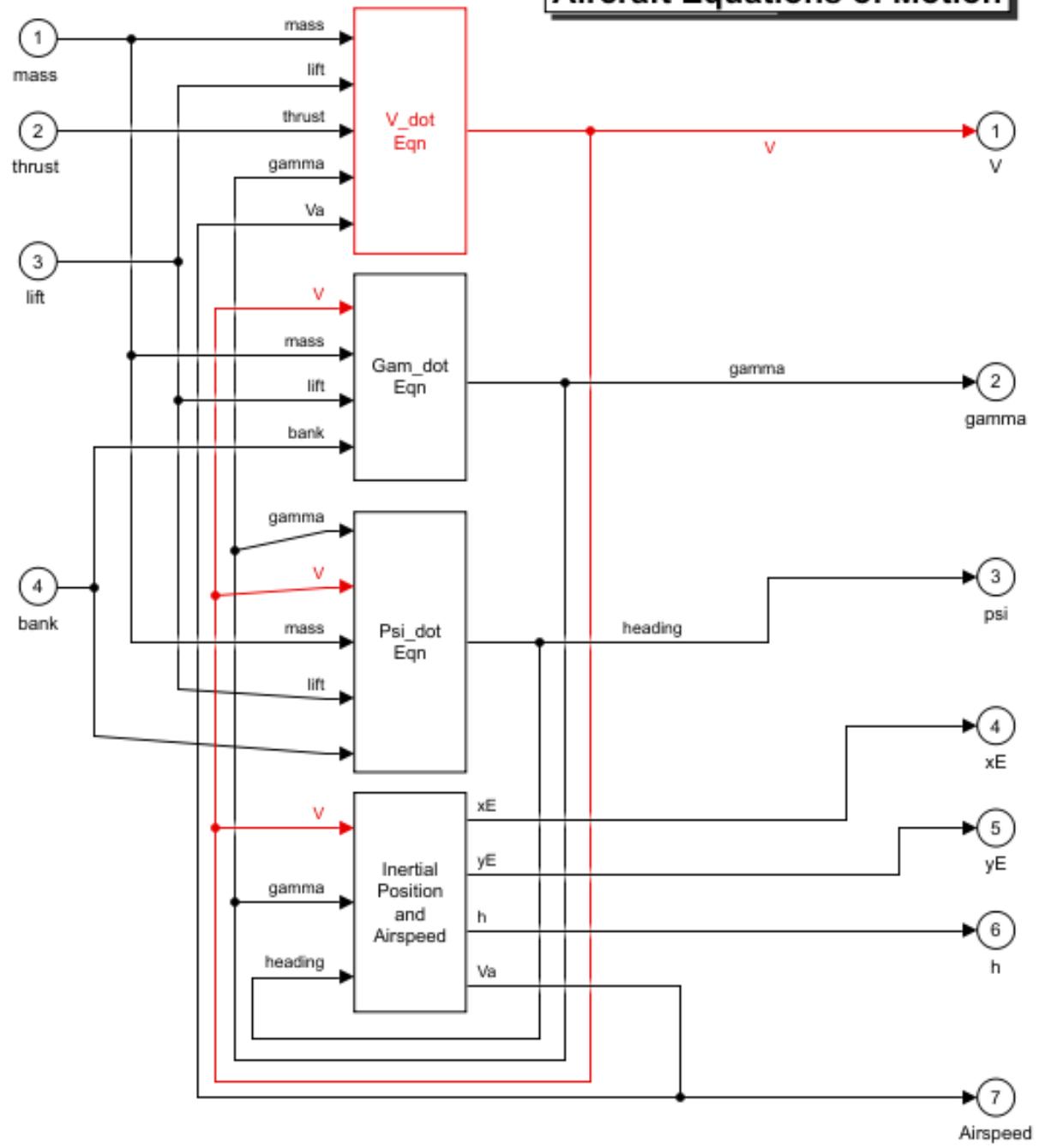


Figura 2.4: Sottosistema Simulink 1

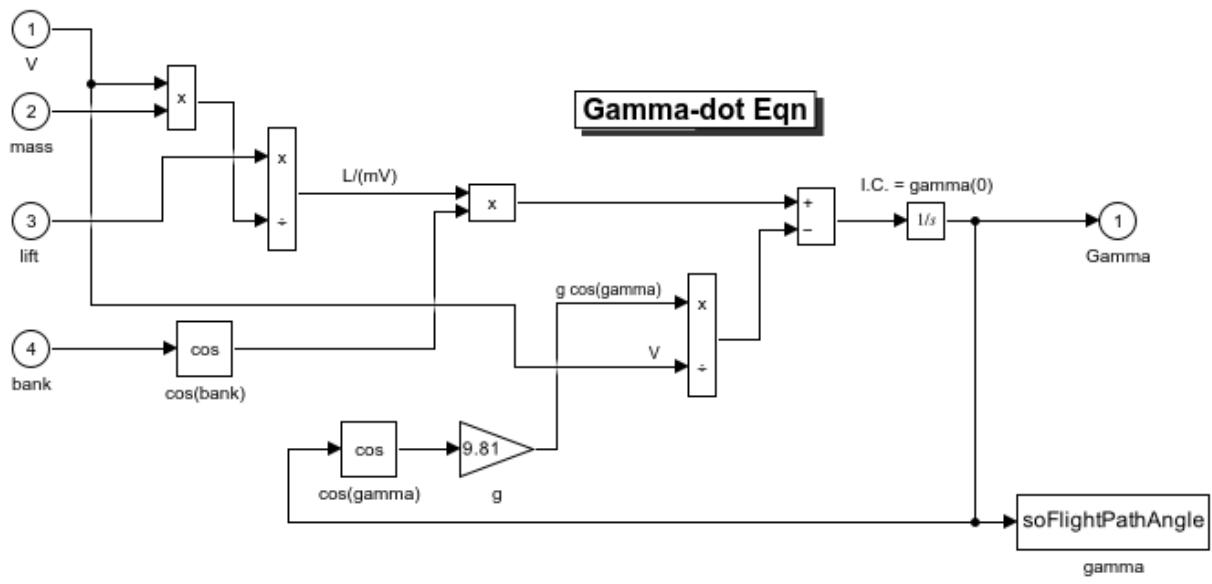


Figura 2.5: Sottosistema Simulink 2

Inertial Position

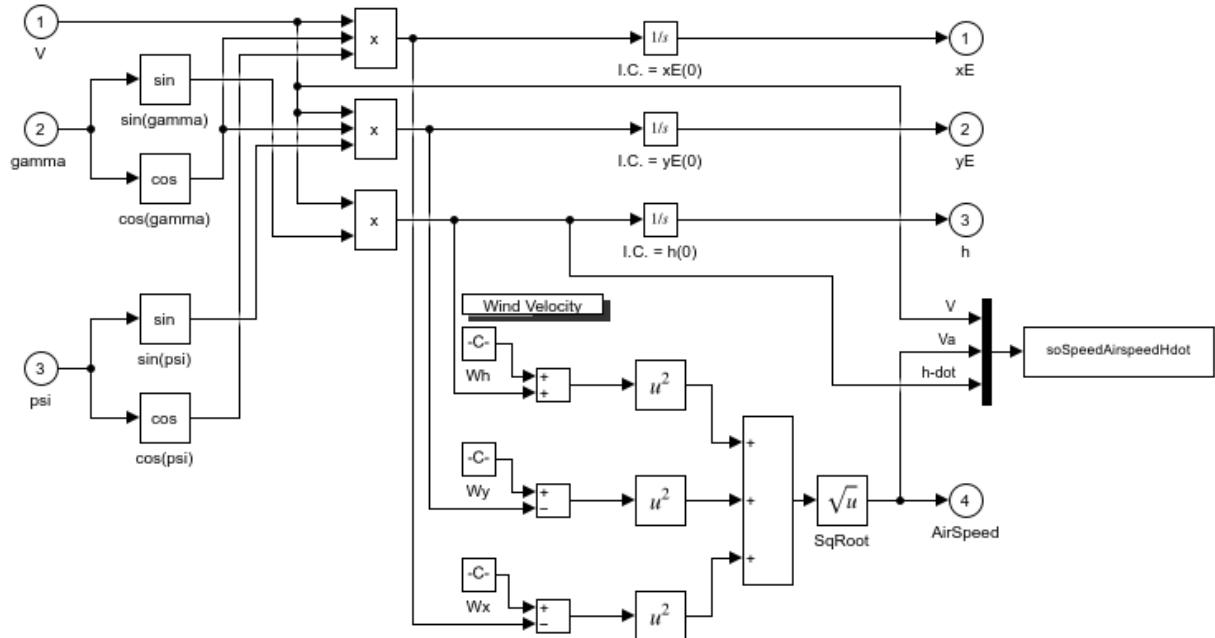


Figura 2.6: Sottosistema Simulink 3

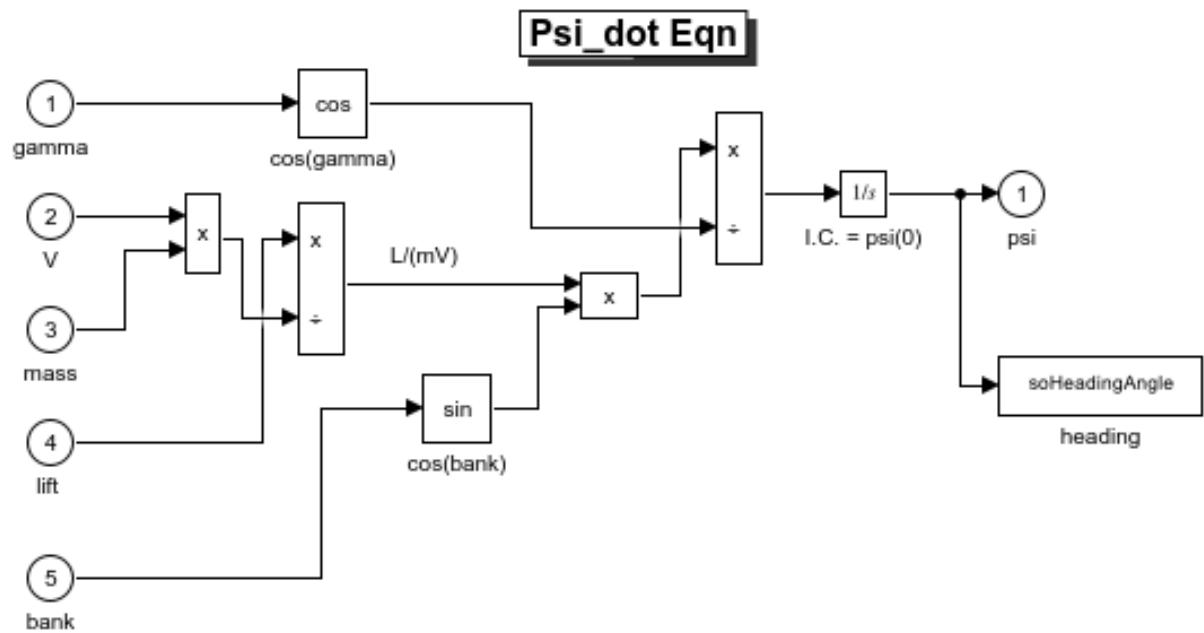


Figura 2.7: Sottosistema Simulink 4

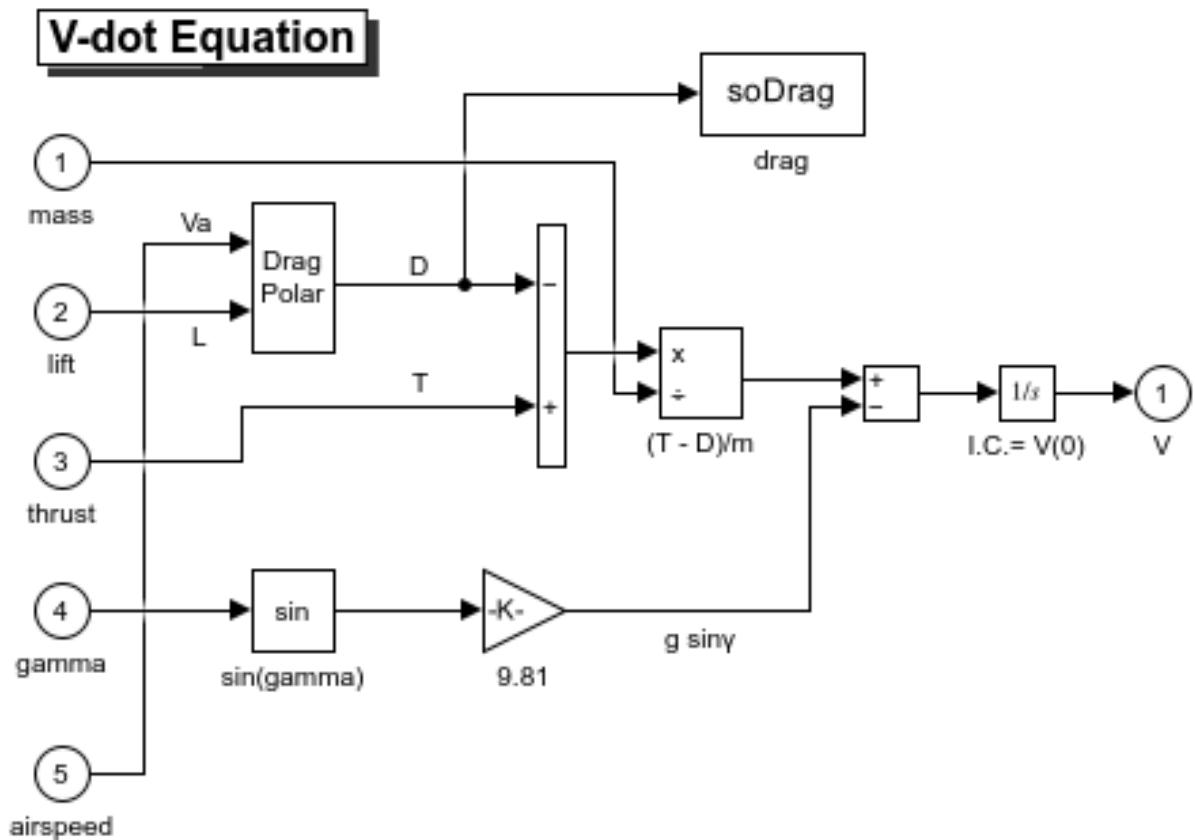


Figura 2.8: Sottosistema Simulink 5

Parabolic Drag Polar

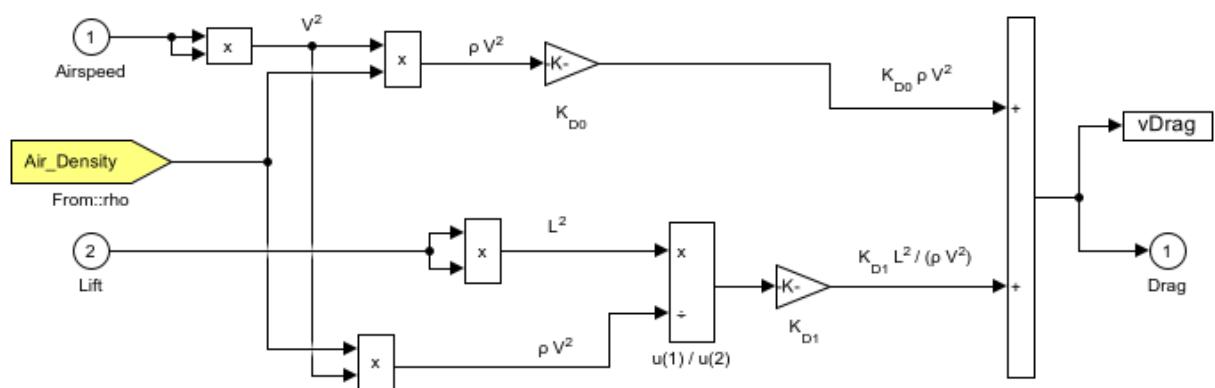


Figura 2.9: Sottosistema Simulink 6

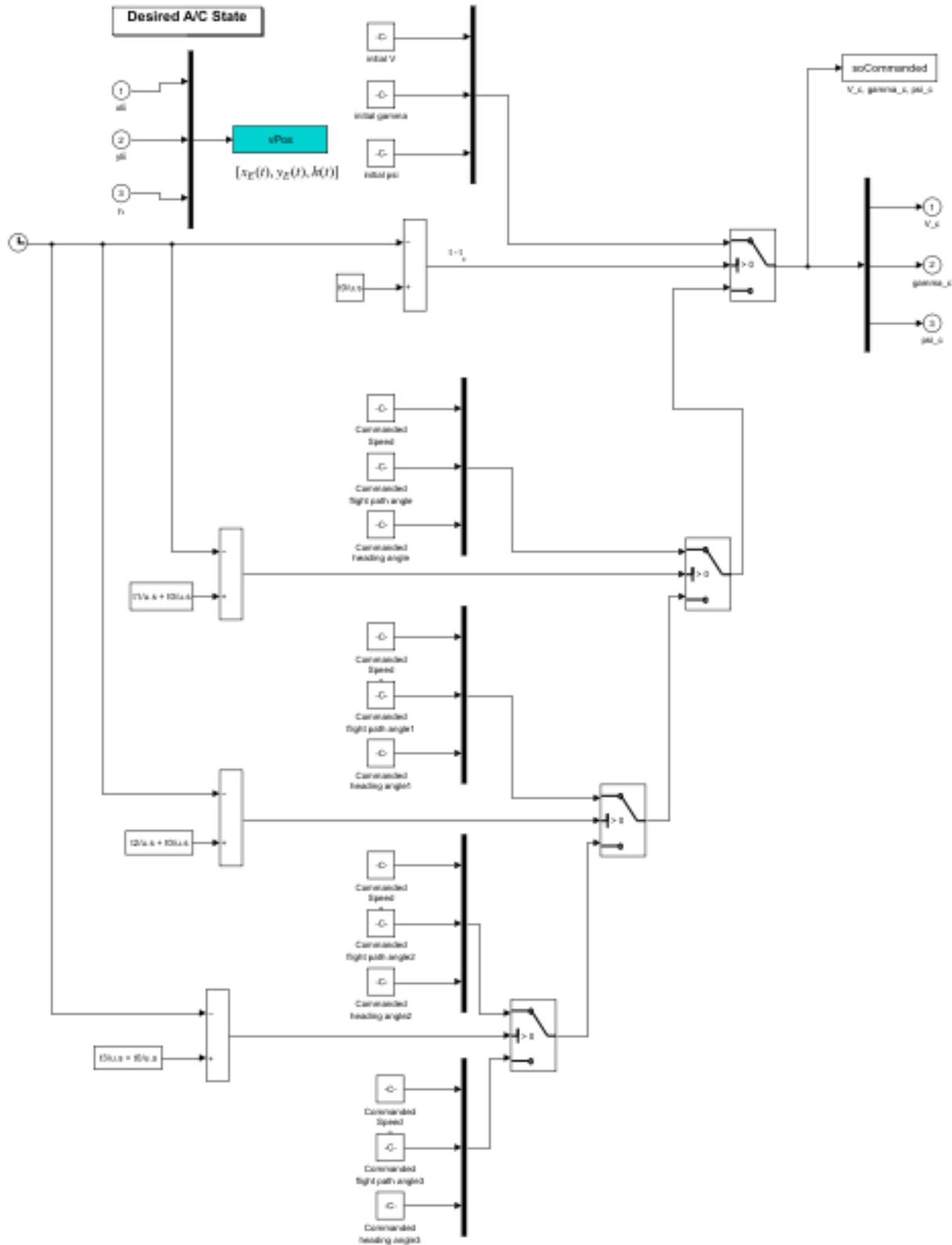


Figura 2.10: Sottosistema Simulink 7

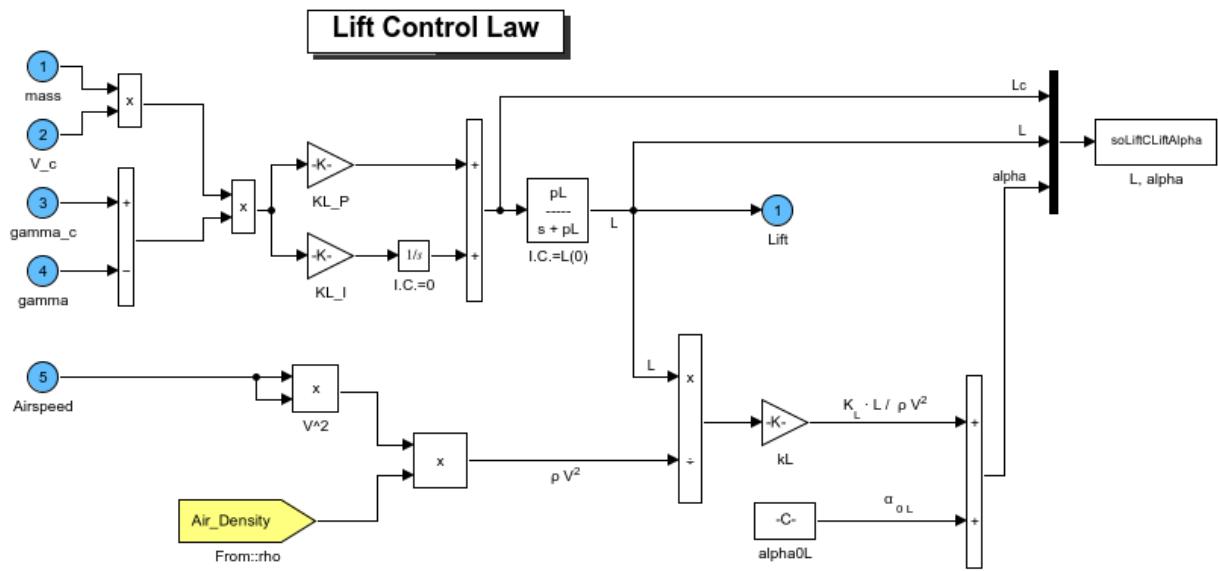


Figura 2.11: Sottosistema Simulink 8

$$x = L(t), u = L_{\alpha}(t)$$

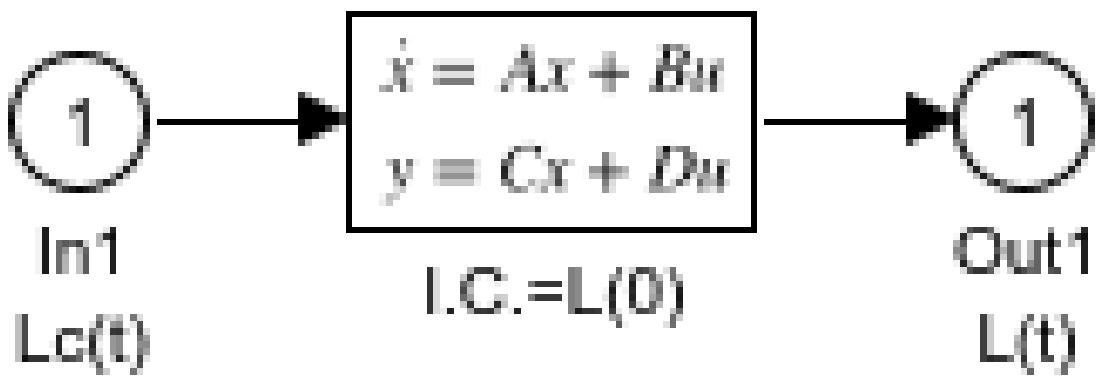


Figura 2.12: Sottosistema Simulink 9

Mass Calculation

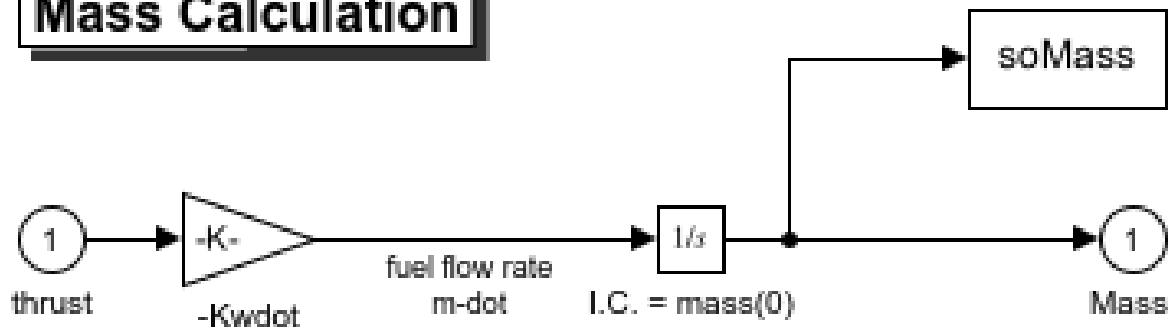


Figura 2.13: Sottosistema Simulink 10

Bank Angle Control Law

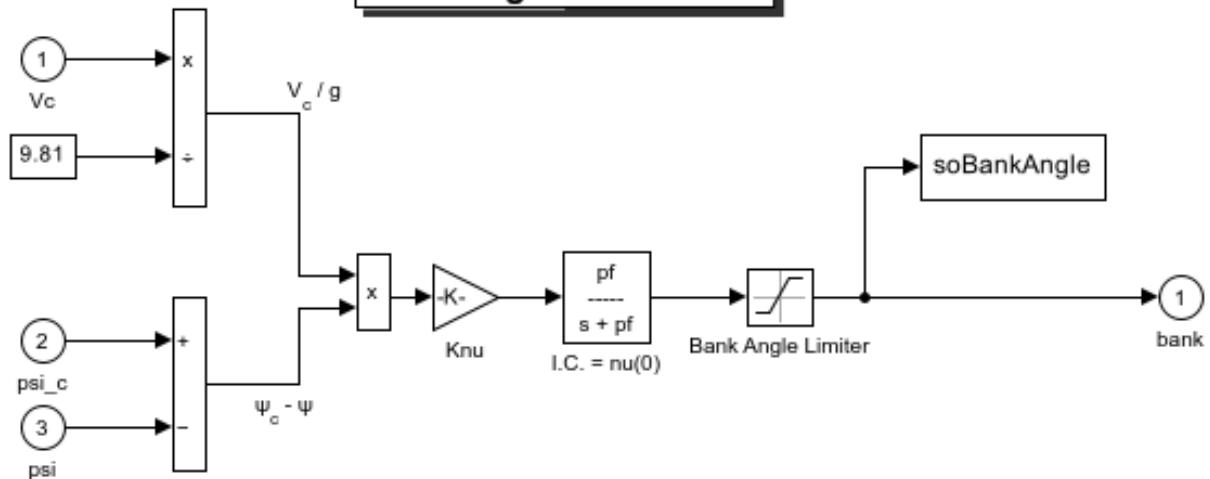


Figura 2.14: Sottosistema Simulink 11

$$x(t) = \nu u(t), \quad u(t) = \nu u_{\text{ref}}(t)$$

$$A = -pNu, \quad B = pNu$$

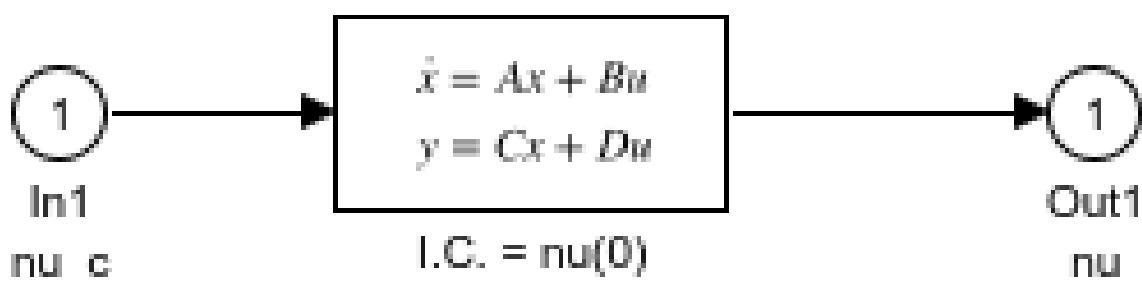


Figura 2.15: Sottosistema Simulink 12

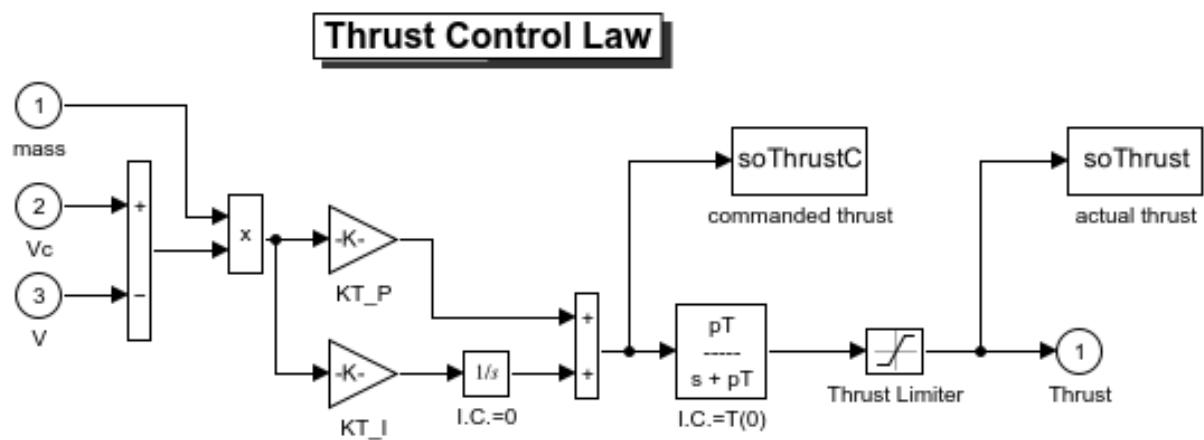


Figura 2.16: Sottosistema Simulink 13

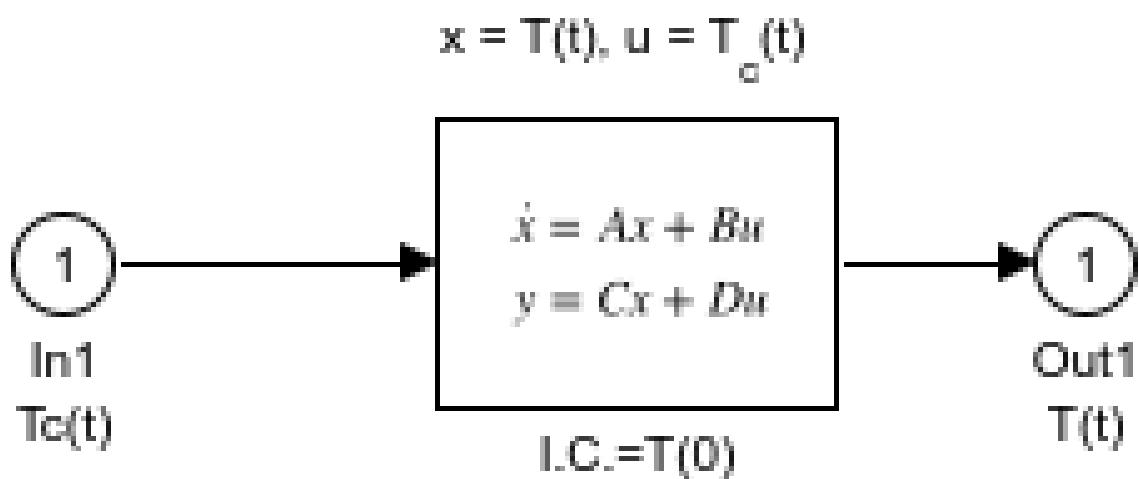


Figura 2.17: Sottosistema Simulink 14

2.4 Risultati del modello

Di seguito sono presentati i risultati del modello analizzato.

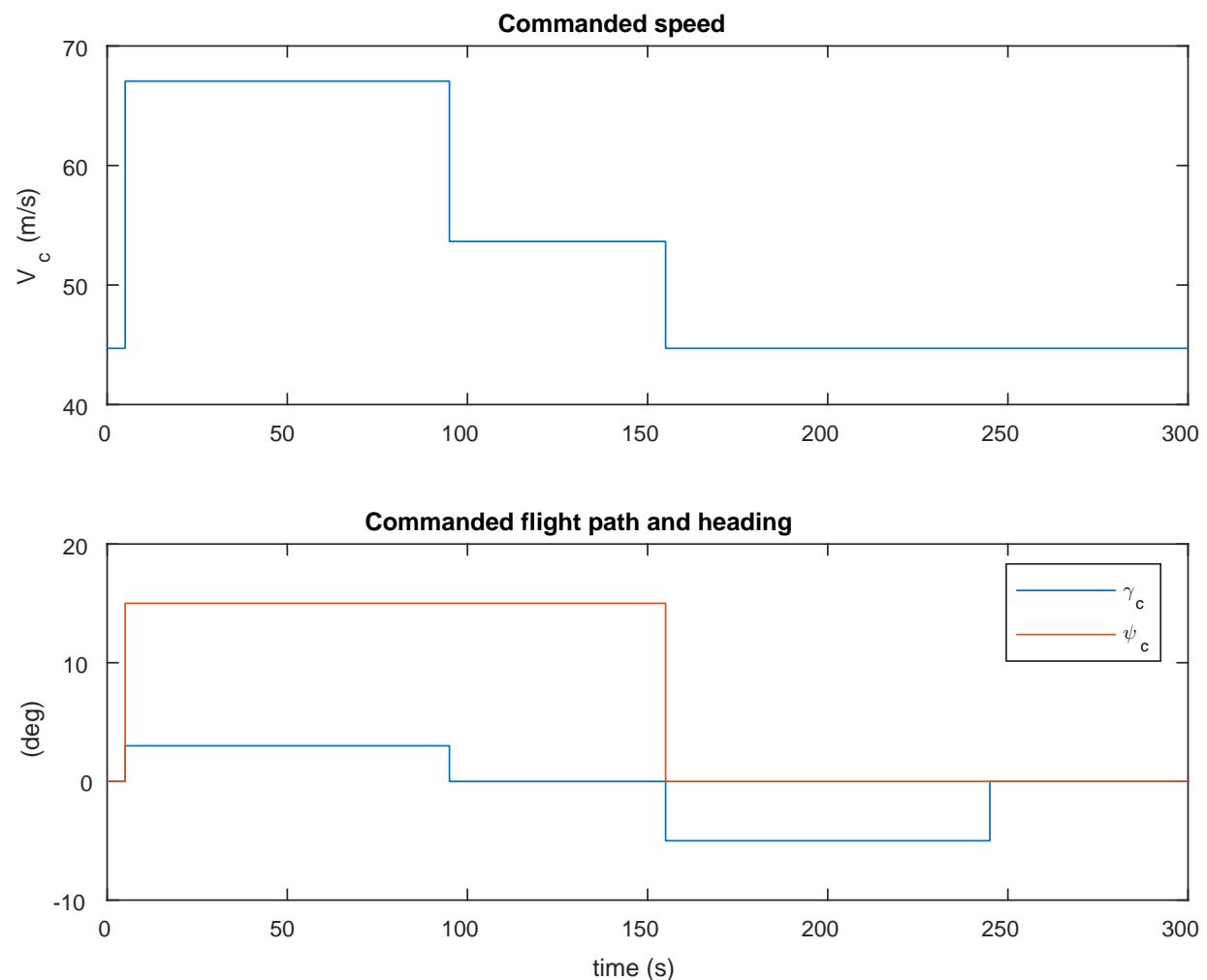


Figura 2.18: Velocità, γ e ψ comandate

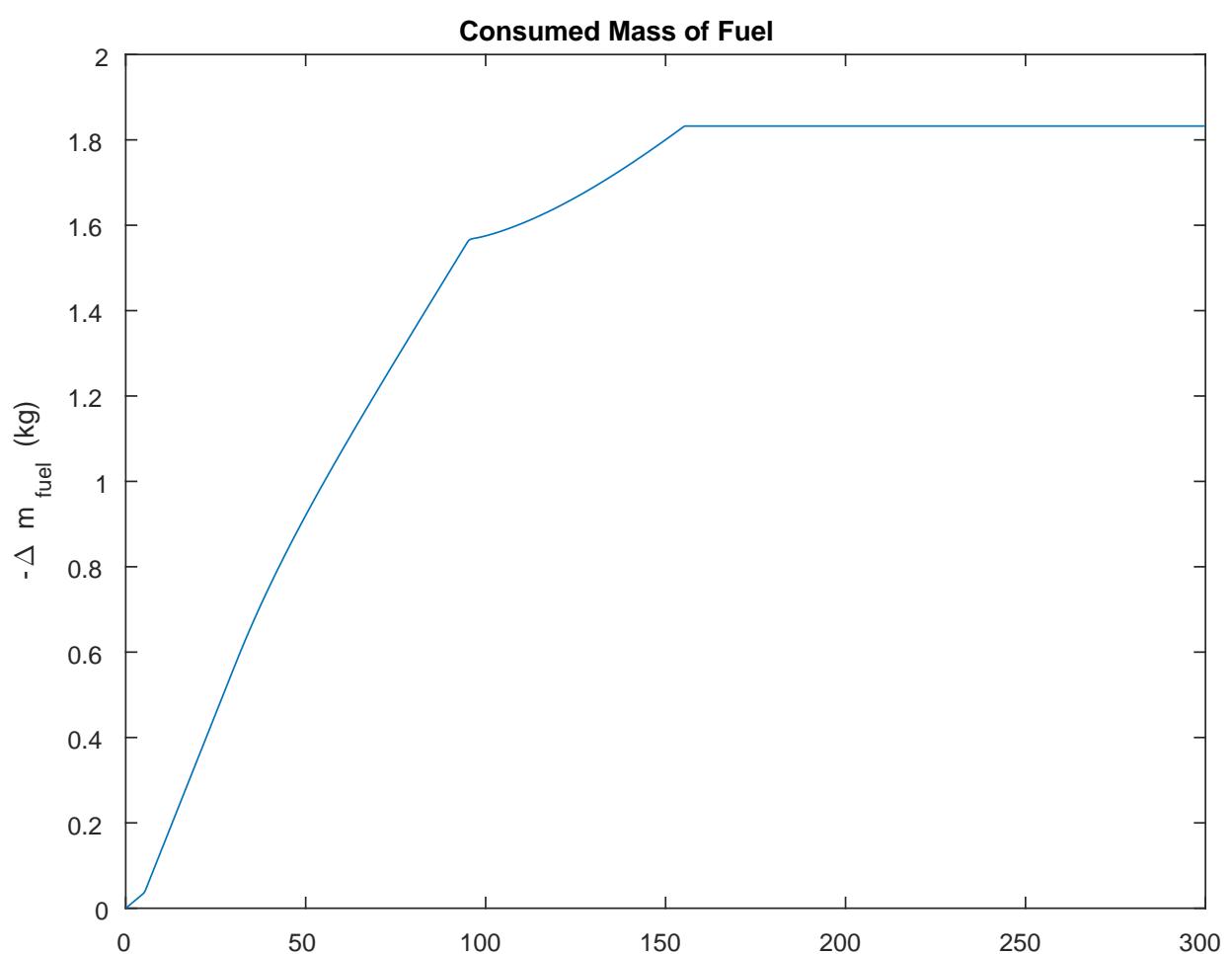


Figura 2.19: Massa di carburante consumato

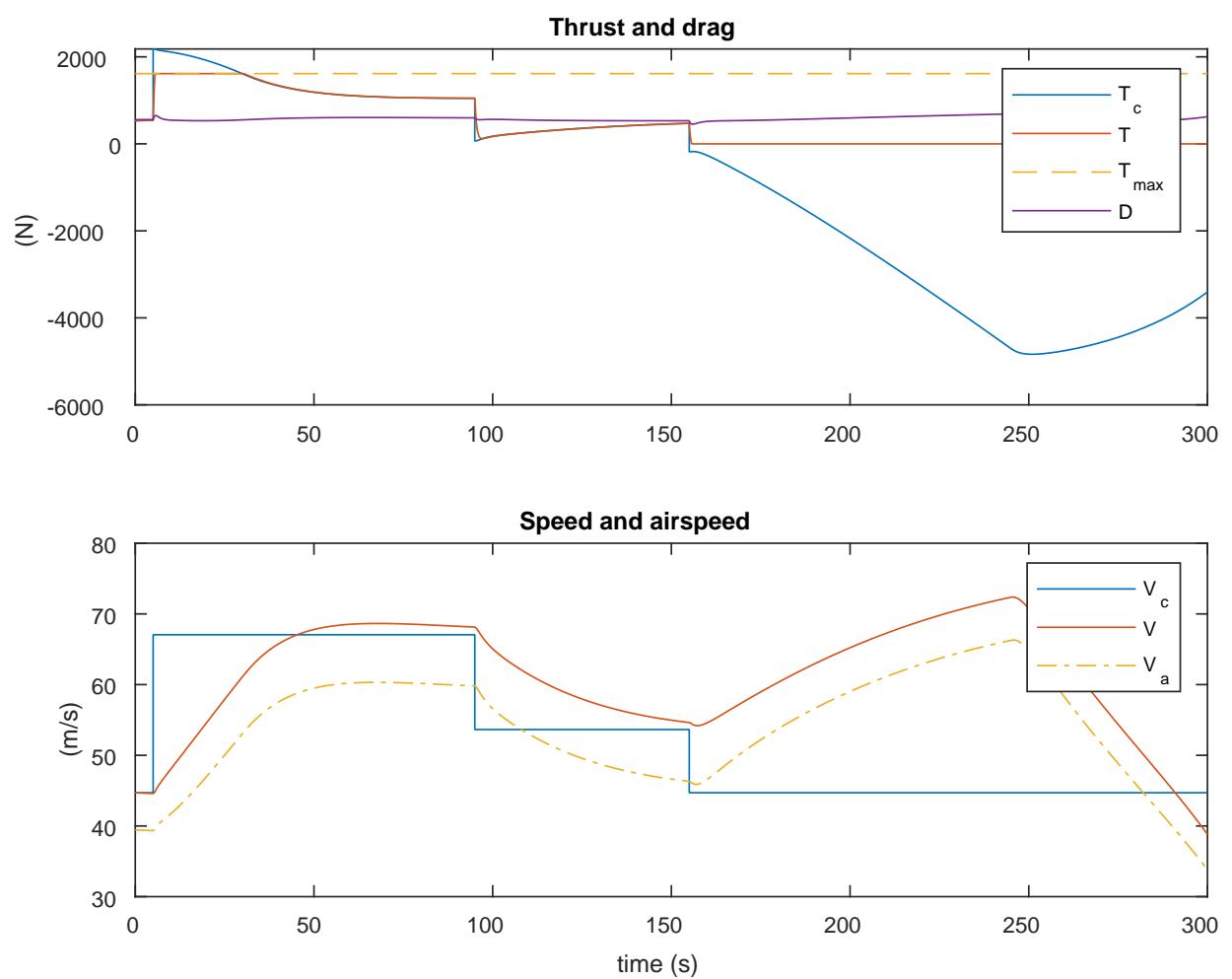


Figura 2.20: Spinta, Resistenza e Velocità

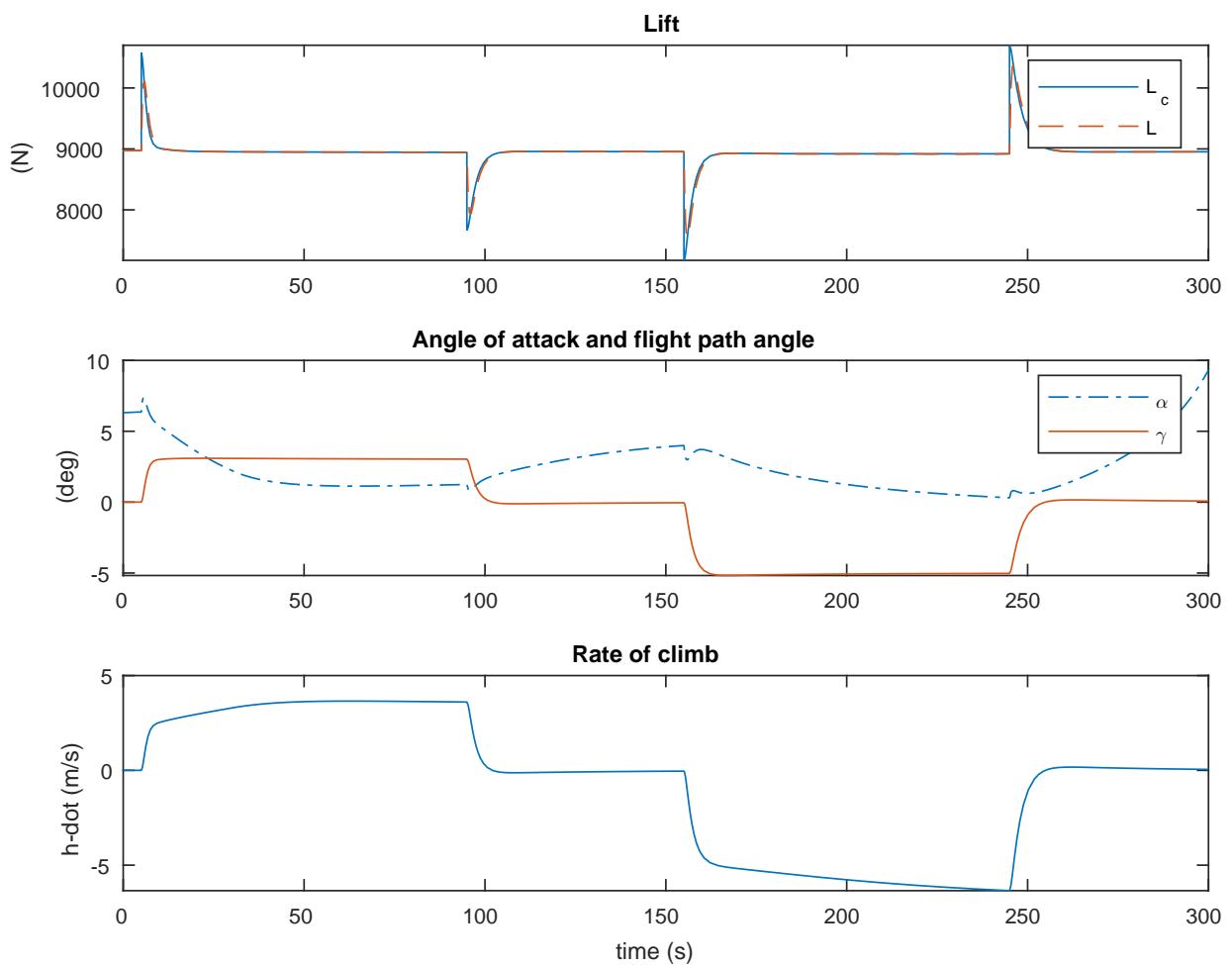


Figura 2.21: Portanza, α e γ , rateo di salita

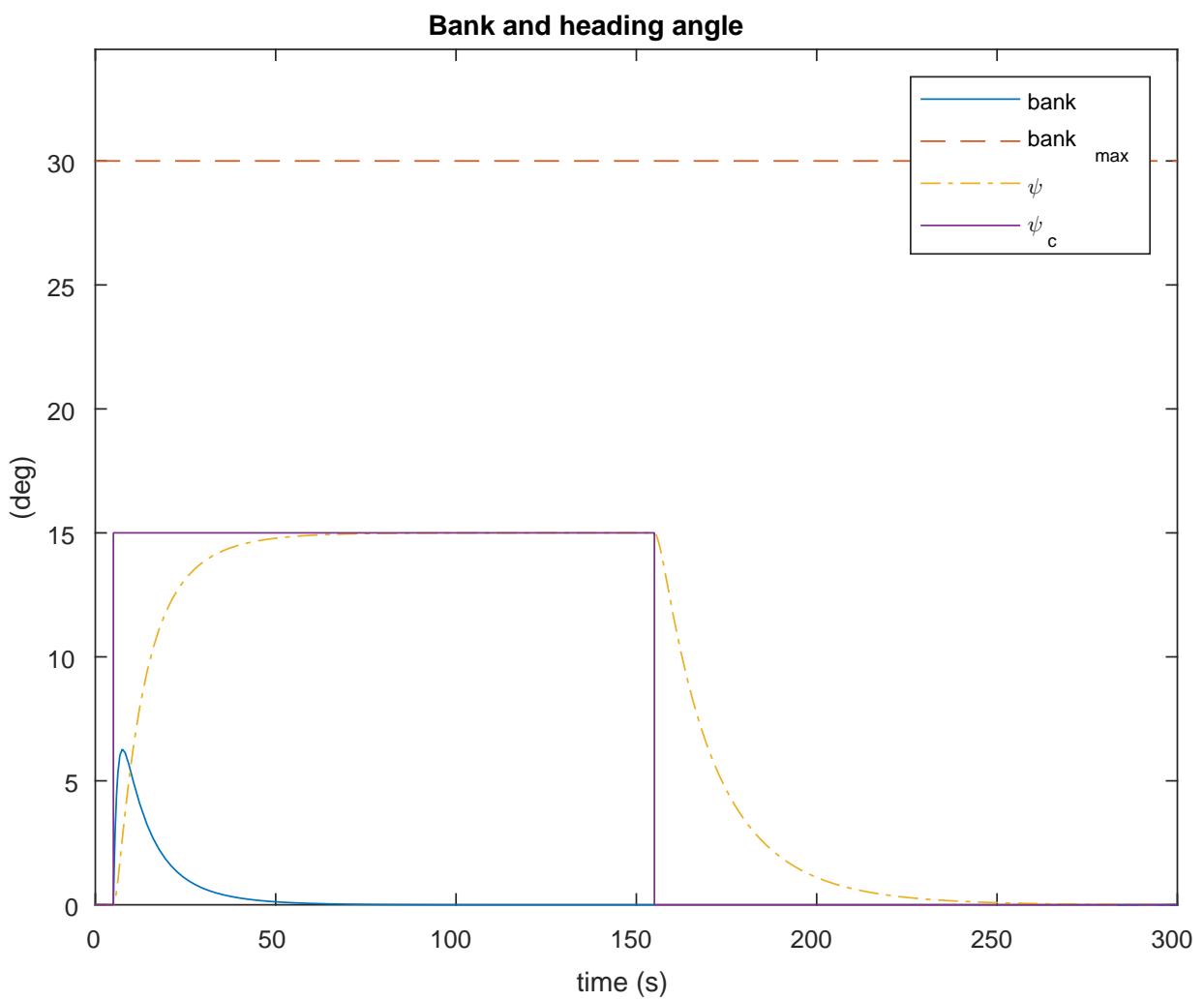


Figura 2.22: Angolo di bank e heading

2.4.1 Traiettoria del Centro di Massa

Col listato riportato di seguito viene mostrata l'evoluzione temporale delle coordinate del centro di massa. In Figura 2.23 queste ultime sono state graficate insieme, mentre nella 2.24 viene mostrato il moto in uno spazio tridimensionale, mediante la funzione *myplotTrajectoryAndBody_Nape3D*. Di seguito i listati relativi a questi ultimi grafici.

```

1 clc; clear; close all;
2 load("Workspace3DFlight.mat","soLiftCLiftAlpha","soBankAngle","soHeadingAngle","soFlightPathAngle","vPos");
3 out.Lc_L_AlphaB = soLiftCLiftAlpha;
4 out.vNu = soBankAngle;
5 out.vDelta = soHeadingAngle;
6 out.vGamma = soFlightPathAngle;
7 out.vPos = vPos;
8
9
10 t_trajectory = 300; %Tempo di osservazione della manovra [s]
11
12 vTime = linspace(0,t_trajectory,300);
13
14 vXe = interp1(out.vPos.Time,out.vPos.Data(:,1),vTime,'pchip');
15 vYe = interp1(out.vPos.Time,out.vPos.Data(:,2),vTime,'pchip');
16 vh = interp1(out.vPos.Time,out.vPos.Data(:,3),vTime,'pchip');
17 vZe = -vh;
18
19 vDelta = interp1(out.vDelta.Time,out.vDelta.Data,vTime,'pchip');
20 vGamma = interp1(out.vGamma.Time,out.vGamma.Data,vTime,'pchip');
21 vNu = interp1(out.vNu.Time,out.vNu.Data,vTime,'pchip');
22 vBeta = zeros(length(vTime),1);
23 vAlphaB = interp1(out.Lc_L_AlphaB.Time,out.Lc_L_AlphaB.Data(:,3),vTime,'pchip');
24
25
26 %% Ricostruzione delle storie temporali degli angoli di Eulero
27 Rot3 = @(angle) [cos(angle) sin(angle) 0;
28                  -sin(angle) cos(angle) 0;
29                  0 0 1];
30 Rot2 = @(angle) [cos(angle) 0 -sin(angle);
31                  0 1 0;
32                  sin(angle) 0 cos(angle)];
33 Rot1 = @(angle) [1 0 0;
34                  0 cos(angle) sin(angle);
35                  0 -sin(angle) cos(angle)];
36
37 vPsi = zeros(length(vTime),1);
38 vTheta = zeros(length(vTime),1);
39 vPhi = zeros(length(vTime),1);
40 vQuat = zeros(length(vTime),4);
41
42 for i = 1:length(vTime)
43
44     TBE = Rot2(vAlphaB(i))*Rot3(-vBeta(i))*Rot1(vNu(i))*Rot2(vGamma(i))*Rot3(vDelta(i));
45     [psi,theta,phi] = dcm2angle(TBE,'ZYX');
46     quat = angle2quat(psi,theta,phi,'ZYX');
47     vPsi(i) = psi;
48     vTheta(i) = theta;
49     vPhi(i) = phi;
50     vQuat(i,1) = quat(1);
51     vQuat(i,2) = quat(2);
52     vQuat(i,3) = quat(3);
53     vQuat(i,4) = quat(4);
54
55 end
56
57 %% grafica
58 h_fig = figure;
59 theView = [62,29];
60 scale_factor = 0.001;
61 step = [1 50 100 150 200 250 300];
62 myplotTrajectoryAndBody_Nape3D(h_fig,vXe,vYe,vZe,vQuat,scale_factor,step,theView);
63
64 [X,Y] = meshgrid(-600:50:20000,-1200:50:4000);
65 Z = sin(X) + cos(Y);
66 Z = Z*10;
67 C = X.*Y;
68 surf(X,Y,Z,C,'FaceColor','blue','EdgeColor','none")
69 light('Position',[1500 0 -2000],'Style','infinite');
70
71 exportgraphics(gca,'cap6(CG1.eps')
72
73
74 figure;
75 plot(vTime,vXe); hold on;
76 plot(vTime,vYe); hold on;
77 plot(vTime,vZe); hold on;
78 xlabel('time(s)')
79 ylabel('(m)')
80
```

```

81 grid on;
82 legend('\{x_cg\}_E','\{y_cg\}_E','\{z_cg\}_E')
83
84 exportgraphics(gca,'cap6_cg2.eps')
```

Il codice relativo alla funzione per il plot 3D è il seguente:

```

1 function myplotTrajectoryAndBody_Nape(h_fig,vXe,vYe,vZe,vQuat,scale_factor,step,theView,varargin)
2 %
3 %   function plotTrajectoryAndBody(vx,vy,vz,vtheta,vphi,vpsi,scale_factor,step,varargin)
4 %   INPUT:
5 %     vXe, vYe, vXe      vectors of CG coordinates (m)
6 %     vQuat              vector of orientation's quaternion (-)
7 %     scale_factor        normalization factor (scalar)
8 %                           (related to body aircraft dimension, <1 magnifies the body)
9 %     step                attitude sampling factor (scalar)
10 %                           (the body is shown each <step> points along the flight path)
11 %
12 %   OPTIONAL INPUT:
13 %     ???                 ???
14 %
15 % ****
16 % Author: Agostino De Marco, Università di Napoli Federico II
17 %
18 % Function inspired by "trajectory3" function on Matlab Central File Exchange:
19 % https://www.mathworks.com/matlabcentral/fileexchange/5656-trajectory-and-attitude-plot-version-3
20 %

21 %% Sanity checks
22 if (length(vXe)~=length(vYe))||(length(vXe)~=length(vZe))||(length(vYe)~=length(vZe))
23     disp(' Error:');
24     disp('      Wrong dimensions of CG coordinate vectors. Check the size.');
25     M = 0;
26     return;
27 end
28 % if ((length(vtheta)~=length(vphi))||(length(vtheta)~=length(vpsi))||(length(vphi)~=length(vpsi)))
29 %     disp(' Error:');
30 %     disp('      Wrong dimensions of the Euler''s angle vectors. Check the size.');
31 %     M = 0;
32 %     return;
33 % end
34 % if length(vtheta)~=length(vx)
35 if size(vQuat,1)~=length(vXe)
36     disp(' Error:');
37     disp('      Size mismatch between Euler''s angle vectors and CG coordinate vectors.');
38     M=0;
39     return
40 end
41 if isnumeric(step)
42     if step>=length(vXe)
43         disp(' Error:');
44         disp('      Attitude sampling factor out of range. Reduce step.');
45         M=0;
46         return
47     end
48     if step<1
49         step = 1;
50     end
51 end
52 if isvector(step)
53     if ~isempty(find(step > length(vXe)))
54         disp(' Error:');
55         disp('      Some indices out of range. Check.');
56         M=0;
57         return
58     end
59 end
60
61
62 % % Trick: to visualize correctly the coordinates %%%%%%% l'ho ricommentato io
63 % vXe = -vXe - min(-vXe);
64 % vYe = -vYe - min(-vYe);
65 %
66 %% From quaternion components to Euler angles
67 % for kk=1:size(vQuat,1)
68 %     dcm = quat2dcm(vQuat(kk,:));
69 %     for ir=1:3
70 %         for ic=1:3
71 %             vDCM(kk,ir,ic) = dcm(ir,ic);
72 %         end
73 %     end
74 % [vpsi(kk), vtheta(kk), vphi(kk)] = quat2angle(vQuat(kk,:));
75 % end
76 %
77 %% Set the camera view angle
78 % theView = [1,1,0.5]; % 3
79 %
80 %% Misc.
81 movie = nargout;
```

```

83 cur_dir = pwd;
84
85 %% Read aircraft data file 'aircraft.mat' and prepare vertices
86 % see: http://www.mathworks.com/matlabcentral/fileexchange/3642
87 % for functions that translate STL files into .mat with Vertices,
88 % Faces and Connectivity infos
89
90 load aircraft_P2010.mat; % Coords of vertices matche with body-axes definitions
91 V=shape.V;
92 F=shape.F;
93 C=shape.C;
94
95 V(:,1) = V(:,1)-round(sum(V(:,1))/size(V,1));
96 V(:,2) = V(:,2)-round(sum(V(:,2))/size(V,1));
97 V(:,3) = V(:,3)-round(sum(V(:,3))/size(V,1));
98
99 Xb_nose_tip = max(abs(V(:,1)));
100 V = V./(scale_factor*Xb_nose_tip);
101
102 %% How many body visualizations along the flight path
103 if isscalar(step)
104     usr_modulo = mod(length(vXe),step);
105
106 %% The visualization loop
107 frame = 0;
108 for i=1:step:(length(vXe)-usr_modulo)
109
110     if movie || (i == 1)
111         clf(h_fig);
112         h_fig = plot3(vXe,vYe,vZe);
113         % grid on;
114         hold on;
115         light;
116     end
117
118     % Trick: conjugate quaternion to invert sign of rotations %%%%%%%%%%%%%%%L'ho
119     %commentato io
120     vQuat(i,2) = -vQuat(i,2); vQuat(i,3) = -vQuat(i,3); vQuat(i,4) = -vQuat(i,4);
121
122     % Transf. matrix from Earth- to body-axes
123     Tbe = quat2dcm(vQuat(i,:));
124     % Transf. matrix from Body- to Earth-axes
125     Teb = Tbe';
126
127     %% Vertices in body-axis coordinates
128     Vb = Teb*V;
129     Vb = Vb';
130
131     P_on_traj = [ vXe(i) vYe(i) vZe(i)];
132     X0 = repmat(P_on_traj,size(Vb,1),1);
133     Vb = Vb + X0;
134
135     %% plot body-axes
136     % Xb = transpose( (2/scale_factor)*Tbe*[1;0;0] ); % CG-to-right wing
137     Xb = transpose( ...
138         (2/scale_factor)*quat2dcm( ...
139             vQuat(i,:)' ... % angle2quat( psi, theta, phi ) ...
140             ) * [1;0;0]' ...
141     );
142     % Yb = transpose( (2/scale_factor)*Tbe*[0;1;0] ); % CG-to-fuselage nose
143     Yb = transpose( ...
144         (2/scale_factor)*quat2dcm( ...
145             vQuat(i,:)' ... % angle2quat( -psi, -theta, -phi ) ...
146             ) * [0;1;0]' ...
147     );
148     % Zb = transpose( (1/scale_factor)*Tbe*[0;0;1] ); % Pilot's head-to-feet direction
149     Zb = transpose( ...
150         (2/scale_factor)*quat2dcm( ...
151             vQuat(i,:)' ... % angle2quat( -psi, -theta, -phi ) ...
152             ) * [0;0;1]' ...
153     );
154     quiver3( ...
155         P_on_traj(1),P_on_traj(2),P_on_traj(3), ...
156         Xb(1),Xb(2),Xb(3), ...
157         'color',[1, 0.0780, 0.1840], 'linewidth',2.5 ...
158     ); hold on
159     quiver3( ...
160         P_on_traj(1),P_on_traj(2),P_on_traj(3), ...
161         Yb(1),Yb(2),Yb(3), ...
162         'color',[0, 0.4470, 0.7410], 'linewidth',2.5 ...
163     ); hold on
164     quiver3( ...
165         P_on_traj(1),P_on_traj(2),P_on_traj(3), ...
166         Zb(1),Zb(2),Zb(3), ...
167         'color',[0.1660, 0.8740, 0.1880], 'linewidth',2.5 ...
168     ); hold on
169
170 %% Display aircraft shape
171 p = patch('faces', F, 'vertices' ,Vb);

```

```

171     set(p, 'facec', [0, 0, 1]);
172     set(p, 'EdgeColor','none');
173     if movie | (i == 1)
174         view(theView);
175         axis equal;
176     end
177
178     if movie
179         if i == 1
180             ax = axis;
181         else
182             axis(ax);
183         end
184         lighting phong
185         frame = frame + 1;
186         M(frame) = getframe;
187     end
188 end % end-of-for
189 end % end-of-if step is scalar
190
191 %% given indices
192
193 if isvector(step)
194     usr_modulo = mod(length(vXe),length(step));
195
196 %% The visualization loop
197 frame = 0;
198 for is=1:length(step)
199     i = step(is);
200     if movie || (i == 1)
201         clf(h_fig);
202         h_fig = plot3(vXe,vYe,vZe);
203         % grid on;
204         hold on;
205         light;
206     end
207
208     % Trick: conjugate quaternion to invert sign of rotations %%%%%%%%%%%%%%%L'ho
209     %commentato io
210     vQuat(i,2) = -vQuat(i,2); vQuat(i,3) = -vQuat(i,3); vQuat(i,4) = -vQuat(i,4);
211
212     % Transf. matrix from Earth- to body-axes
213     Tbe = quat2dcm(vQuat(i,:));
214     % Transf. matrix from Body- to Earth-axes
215     Teb = Tbe';
216
217     %% Vertices in body-axis coordinates
218     Vb = Teb*V';
219     Vb = Vb';
220
221     P_on_traj = [vXe(i) vYe(i) vZe(i)];
222     X0 = repmat(P_on_traj,size(Vb,1),1);
223     Vb = Vb + X0;
224
225     %% plot body-axes
226     % Xb = transpose( (2/scale_factor)*Tbe*[1;0;0] ); % CG-to-right wing
227     Xb = transpose( ...
228         (2/scale_factor)*quat2dcm( ...
229             vQuat(i,:)' ... % angle2quat( psi, theta, phi ) ...
230             ) * [1.8;0;0]' ...
231         );
232     % Yb = transpose( (2/scale_factor)*Tbe*[0;1;0] ); % CG-to-fuselage nose
233     Yb = transpose( ...
234         (2/scale_factor)*quat2dcm( ...
235             vQuat(i,:)' ... % angle2quat( -psi, -theta, -phi ) ...
236             ) * [0;1;0]' ...
237         );
238     % Zb = transpose( (1/scale_factor)*Tbe*[0;0;1] ); % Pilot's head-to-feet direction
239     Zb = transpose( ...
240         (2/scale_factor)*quat2dcm( ...
241             vQuat(i,:)' ... % angle2quat( -psi, -theta, -phi ) ...
242             ) * [0;0;1]' ...
243         );
244     quiver3( ...
245         P_on_traj(1),P_on_traj(2),P_on_traj(3), ...
246         0.5*Xb(1),0.5*Xb(2),0.5*Xb(3), ...
247         'color',[0.6350, 0.0780, 0.1840], 'linewidth',1.5 ...
248     ); hold on
249     quiver3( ...
250         P_on_traj(1),P_on_traj(2),P_on_traj(3), ...
251         0.6*Yb(1),0.6*Yb(2),0.6*Yb(3), ...
252         'color',[0, 0.4470, 0.7410], 'linewidth',1.5 ...
253     ); hold on
254     quiver3( ...
255         P_on_traj(1),P_on_traj(2),P_on_traj(3), ...
256         0.5*Zb(1),0.5*Zb(2),0.5*Zb(3), ...
257         'color',[0.4660, 0.6740, 0.1880], 'linewidth',1.5 ...
258     ); hold on

```

```

259 %% Display aircraft shape
260 p = patch('faces', F, 'vertices' ,Vb);
261 set(p, 'facec', [1, 1, 0]);
262 set(p, 'EdgeColor','none');
263 if movie | (i == 1)
264     view(theView);
265     axis equal;
266 end
267
268 if movie
269     if i == 1
270         ax = axis;
271     else
272         axis(ax);
273     end
274     lighting phong
275     frame = frame + 1;
276     M(frame) = getframe;
277 end
278 end % end-of-for
279 end % end-of-if step is vector
280
281 hold on;
282
283 %% Plot the flight path
284 % some useful lines
285 h_fig = plot3([min(vXe) max(vXe)]*1.1, [max(vYe) max(vYe)]*1.1, [0 0], 'color', ones(3,1)*.8);
286 h_fig = plot3([min(vXe) max(vXe)]*1.1, [min(vYe) min(vYe)]*1.1, [0 0], 'color', ones(3,1)*.8);
287
288 h_fig = plot3([max(vXe) max(vXe)]*1.1, [min(vYe) max(vYe)]*1.1, [0 0], 'color', ones(3,1)*.8);
289 h_fig = plot3([min(vXe) min(vXe)]*1.1, [min(vYe) max(vYe)]*1.1, [0 0], 'color', ones(3,1)*.8);
290
291
292 % curves
293 plot3(vXe, vYe, vZe./vZe.*max(vZe), '-', 'color', [.5 .5 .5]) % Ground Track
294 plot3(vXe, vYe./vYe.*min(vYe), vZe, '-', 'color', [.5 .5 .5]) % Trajectory in a vertical plane
295 h_fig = plot3(vXe, vYe, vZe, 'k', 'linewidth', 0.7); % 3D trajectory
296
297 lighting phong;
298 % grid on;
299 % view(theView);
300 daspect([1 1 1]);
301
302 xlabel('$x_{\{E\}} (m)$', 'interpreter', 'latex', 'fontsize', 11);
303 ylabel('$y_{\{E\}} (m)$', 'interpreter', 'latex', 'fontsize', 11);
304 zlabel('$z_{\{E\}} (m)$', 'interpreter', 'latex', 'fontsize', 11);
305
306 %% Plot Earth axes
307 quiver3( ...
308     0,0,0, ...
309     abs(0.6*max(vXe)),0,0, ...
310     'color',[0.6350, 0.0780, 0.1840], 'linewidth',1.5 ...
311 ); hold on;
312 quiver3( ...
313     0,0,0, ...
314     0,abs(1.1*max(vYe)),0, ...
315     'color',[0, 0.4470, 0.7410], 'linewidth',1.5 ...
316 ); hold on;
317
318 quiver3( ...
319     0,0,0, ...
320     0,0,(abs(max(vZe))/6),...
321     'color',[0.4660, 0.6740, 0.1880], 'linewidth',1.5 ...
322 ); hold on;
323
324
325 %xlim([min([-0.05*abs(max(vXe)),1.1*min(vXe)]), 1.1*max(vXe)]);
326 %ylim([min([-0.05*abs(max(vYe)),1.1*min(vYe)]), 1.1*max(vYe)]);
327 %zlim([min([-0.05*abs(max(vZe)),1.1*min(vZe)]), max([abs(max(vZe)),0.2*abs(max(vXe))])]);
328
329 set(gca,'XDir','reverse');
330 set(gca,'ZDir','reverse');
331 grid on
332
333 cd (cur_dir);
334
335 end

```

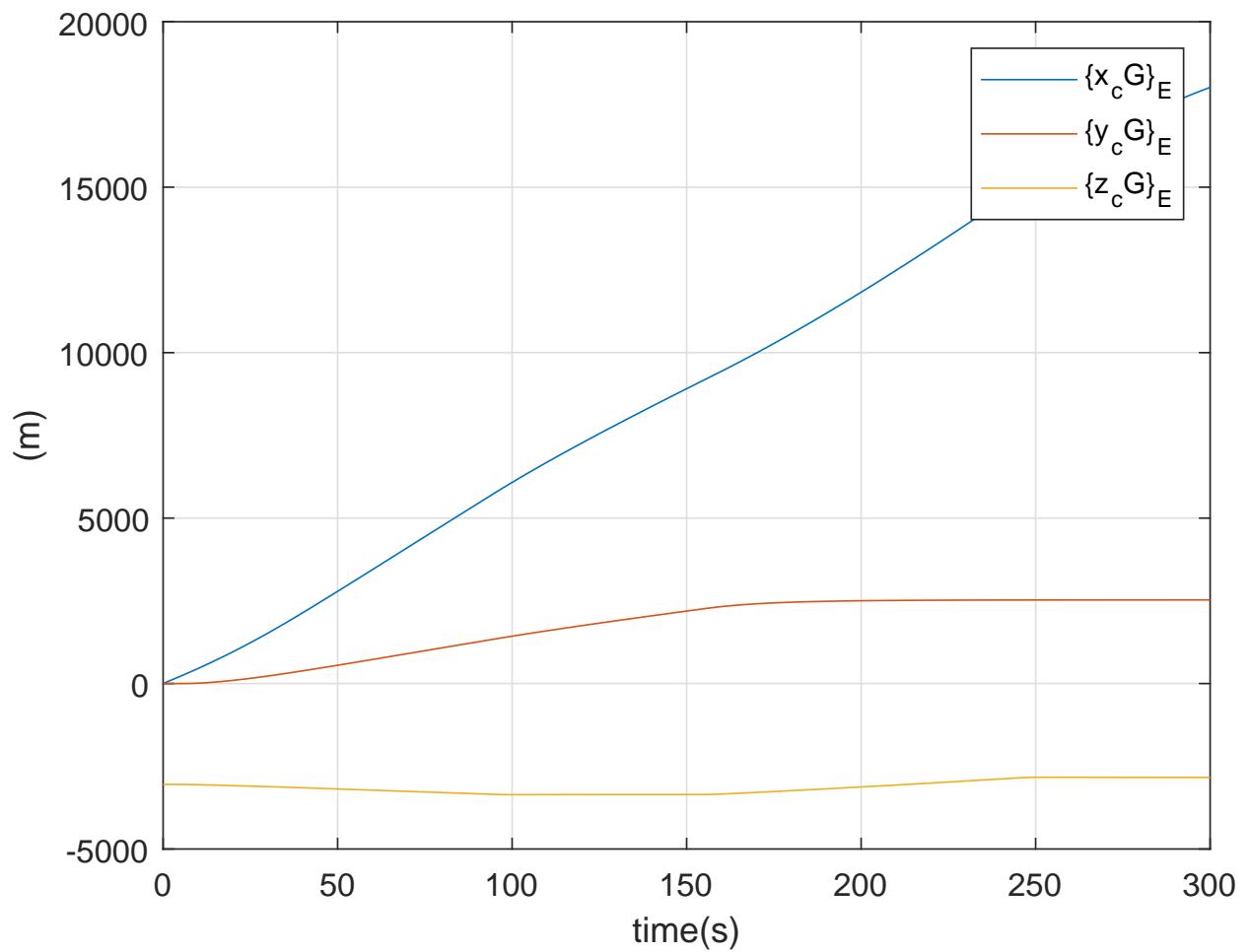


Figura 2.23: Storie temporali delle coordinate del CG

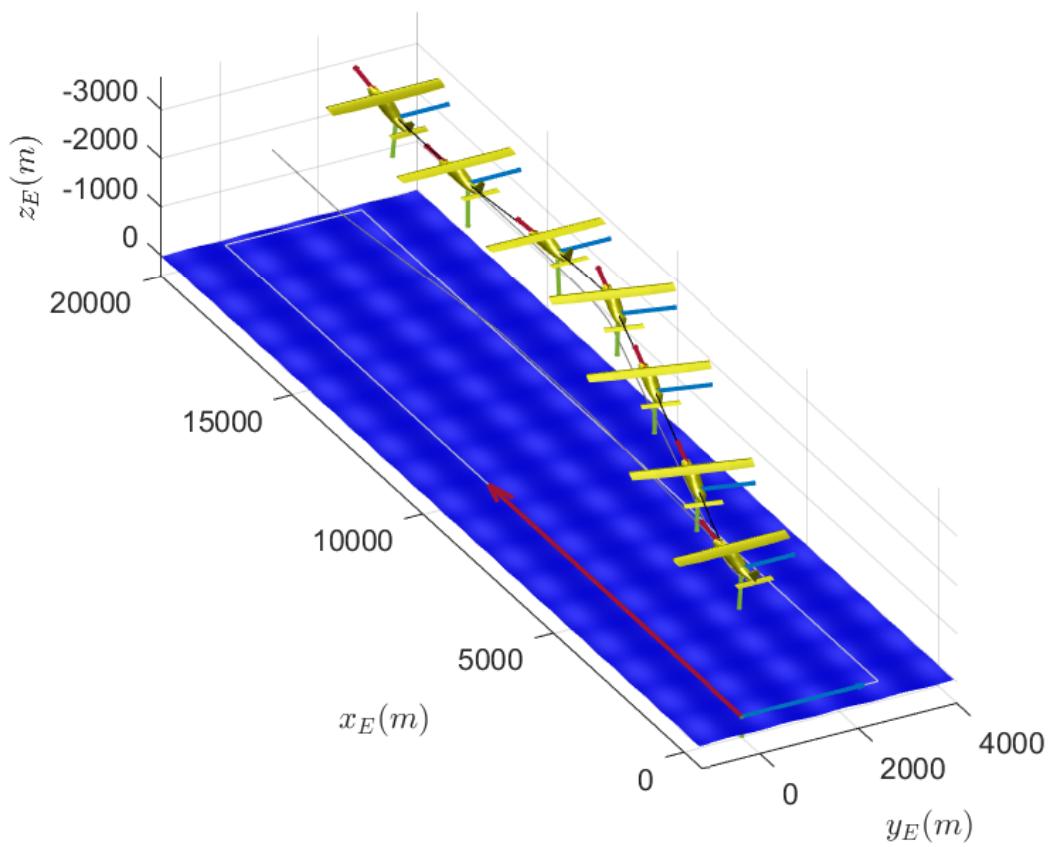


Figura 2.24: Visualizzazione 3D delle storie temporali delle CG

2.5 Risultati di un secondo modello

Nel precedente esercizio la spinta comandata raggiungeva valori negativi (vedi Figura 2.20), per compensarlo vengono fatti alcuni accorgimenti.

- si tiene conto dell'efficienza dell'elica (ipotizzata pari a $\eta_{pr} = 0.8$) e della quota per la potenza
- viene modificata la legge degli angoli comandati

Il listato è il seguente:

```

1 clear all; close all; clc;
2 wingSurface = 13.9 * 10.76 * str2u('ft^2');
3 wingSpan    = 10.3 * 3.28 * str2u('ft');
4 aspectRatio = wingSpan^2/wingSurface;
5 oswald      = 0.95;
6
7 aspectRatioE = aspectRatio*oswald;
8 piARE       = pi*aspectRatioE;
9 cDO          = 0.02;
10 weightMax   = 2557 * str2u('lbf');
11 weightEmpty  = 1687 * str2u('lbf');
12 weightMaxFuel = 440 * str2u('lbf');
13 weightNom   = weightEmpty + 0.75*weightMaxFuel;
14 kWdot        = 4e-6 * str2u('slug/(lbf*s)');
15 speedMin     = 60 * str2u('mph');
16 speedMax     = 160 * str2u('mph');
17
18 [temp_SL, a_SL, p_SL, rho_SL] = atmosisa(0);
19 altitude_0   = 10000 * str2u('ft');
20 [temp_0, a_0, p_0, rho_0] = atmosisa(altitude_0/u.m);
21
22 sigma = rho_0/rho_SL;
23 ShaftHorsePower=180*sigma *str2u('hp'); % Potenza all'albero
24 eta_pr=0.8; % efficienza dell'elica
25 Power_disponibile=ShaftHorsePower*eta_pr; %potenza disponibile
26 thrustMax    = Power_disponibile/speedMax ;
27
28
29 bankAngleMax = 30 * str2u('deg');
30 altitude_0   = 10000 * str2u('ft');
31 airDensity_0 = rho_0 * str2u('kg/m^3');
32 kD0 = 0.5*cDO*wingSurface;
33 kD1 = 2/(piARE*wingSurface);
34 cLalpha = 0.1 / str2u('deg');
35 alphaOL = -2.86 * str2u('deg');
36 kL = 2.0/(wingSurface * cLalpha);
37 alphaBody_max = 8 * str2u('deg');
38 limitMaxOverRhoV2 = (alphaBody_max - alphaOL)/kL;
39 wind_0 = [15, 23, 6] * str2u('mph');
40 weight_0      = weightNom;
41 mass_0        = weight_0 / (9.81 * str2u('m/s^2'));
42 speed_0       = 100 * str2u('mph');
43 flightPathAngle_0 = 0 * str2u('deg');
44 headingAngle_0 = 0 * str2u('deg');
45 xE_0          = 0 * str2u('m');
46 yE_0          = 0 * str2u('m');
47 altitude_0   = 10000 * str2u('ft');
48 bankAngle_0   = 0 * str2u('deg');
49 lift_0         = weight_0*cos(flightPathAngle_0)/cos(bankAngle_0);
50 drag_0         = kD0 * airDensity_0 * speed_0^2 ...
+ kD1 * (lift_0^2)/(airDensity_0 * speed_0^2);
51
52 thrust_0       = drag_0 + weight_0*sin(flightPathAngle_0);
53 pT = 2.0 * str2u('rad/s');
54 pL = 2.5 * str2u('rad/s');
55 pNu = 1.0 * str2u('rad/s');
56 kTI = 0.002 / str2u('s^2');
57 kTP = 0.080 / str2u('s');
58 kLI = 0.010 / str2u('s^2');
59 kLP = 0.500 / str2u('s');
60 kNu = 0.075 / str2u('s');
61 t0             = 5*str2u('s');
62 speed_c        = 150 * str2u('mph');
63 flightPathAngle_c = 3 * str2u('deg');
64 hDot_c         = speed_c * sin(flightPathAngle_c);
65 headingAngle_c = 15 * str2u('deg');
66
67 t1              = 90*str2u('s');
68 speed_c_1       = 120 * str2u('mph');
69 flightPathAngle_c_1 = 2 * str2u('deg');
70 hDot_c_1         = speed_c * sin(flightPathAngle_c_1);
71 headingAngle_c_1 = 15 * str2u('deg');
72
73 t2              = 150*str2u('s');
74 speed_c_2       = 100 * str2u('mph');
75 flightPathAngle_c_2 = 1 * str2u('deg');
76

```

```

77 hDot_c_2      = speed_c * sin(flightPathAngle_c_2);
78 headingAngle_c_2   = 0 * str2u('deg');
79
80 t3           = 240*str2u('s');
81 speed_c_3    = 100 * str2u('mph');
82 flightPathAngle_c_3 = 0 * str2u('deg');
83 hDot_c_3      = speed_c * sin(flightPathAngle_c_3);
84 headingAngle_c_3   = 0 * str2u('deg');

```

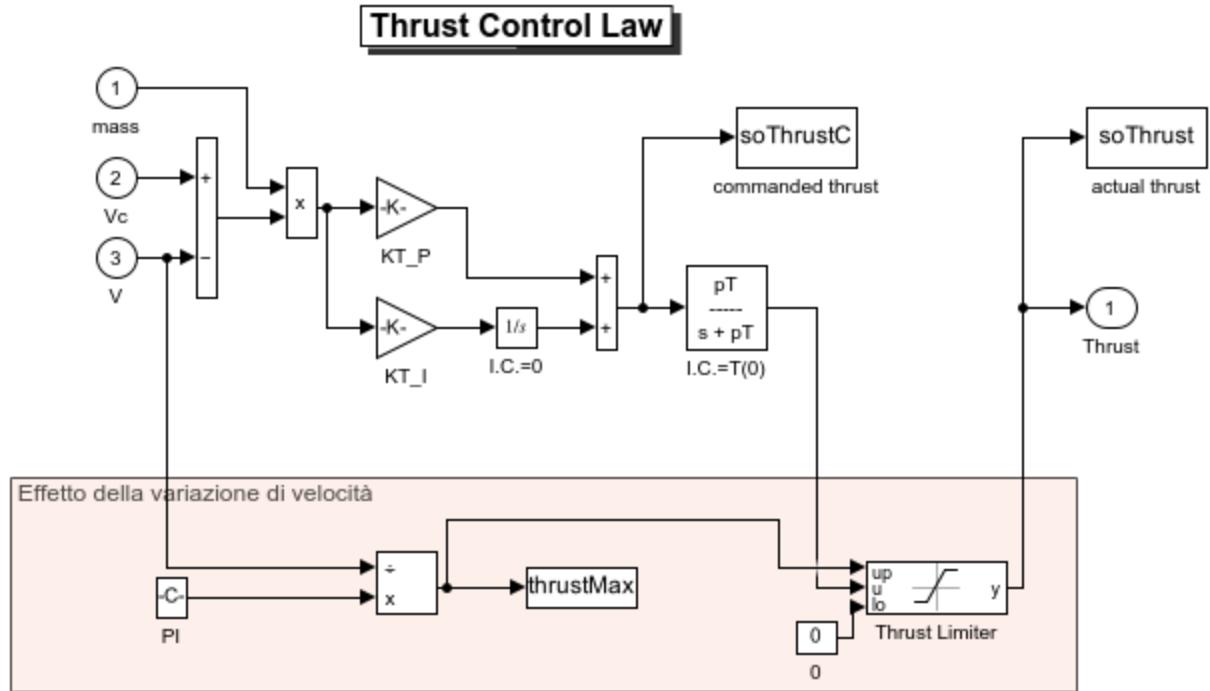


Figura 2.25: Sottosistema Simulink per la spinta variabile con la velocità

2.6 Risultati del modello

Di seguito sono presentati i risultati del modello analizzato.

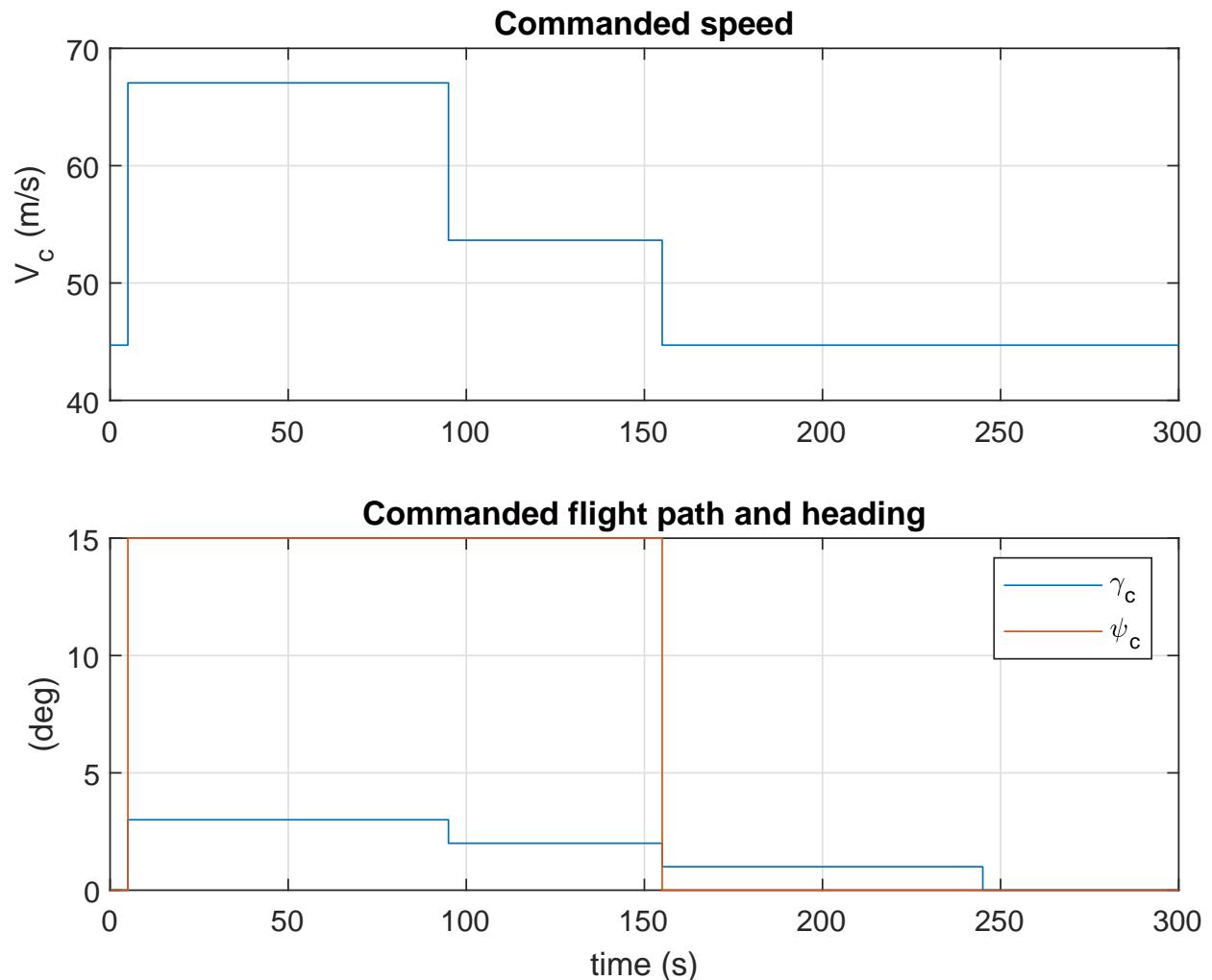


Figura 2.26: Velocità, γ e ψ comandate 2

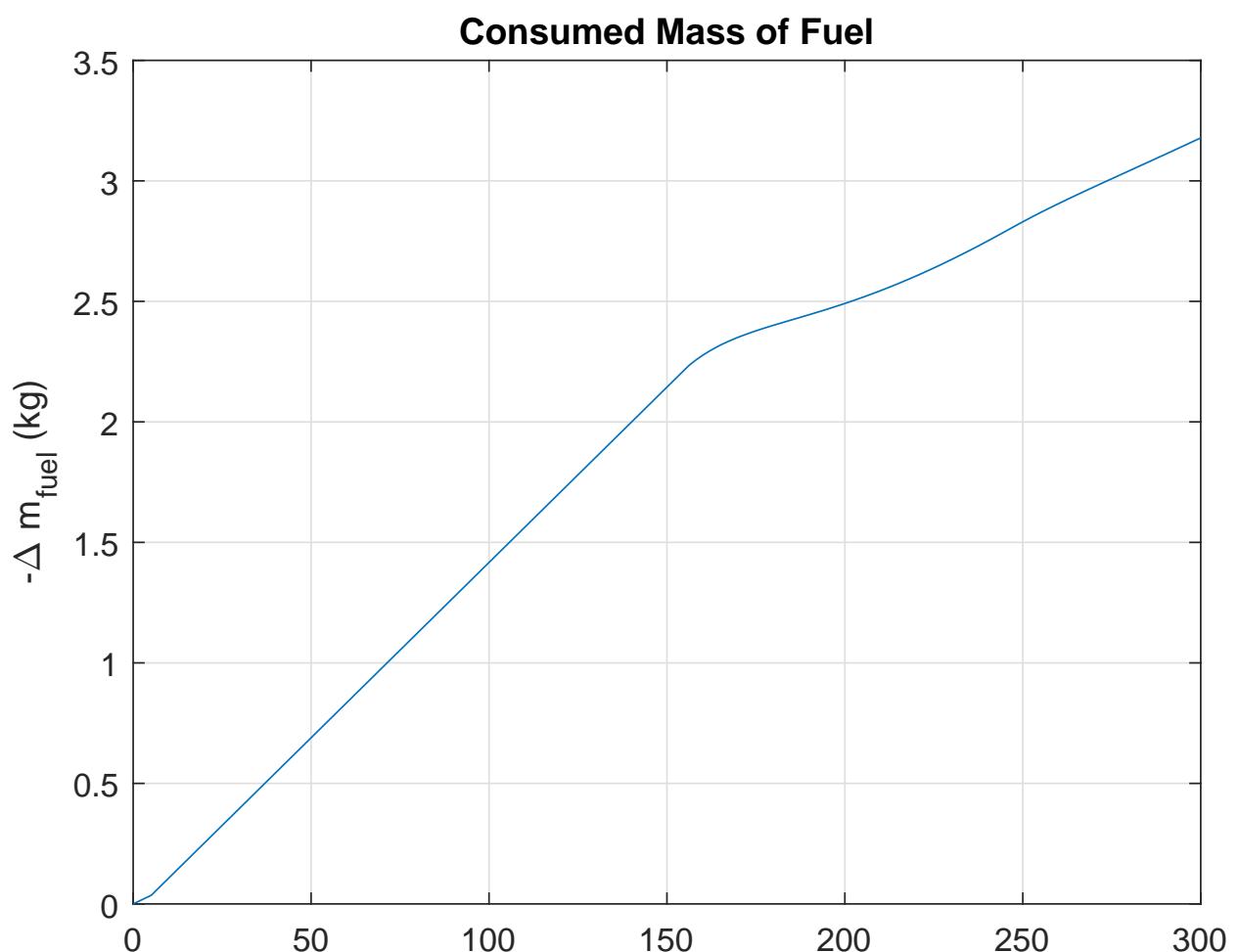


Figura 2.27: Massa di carburante consumato 2

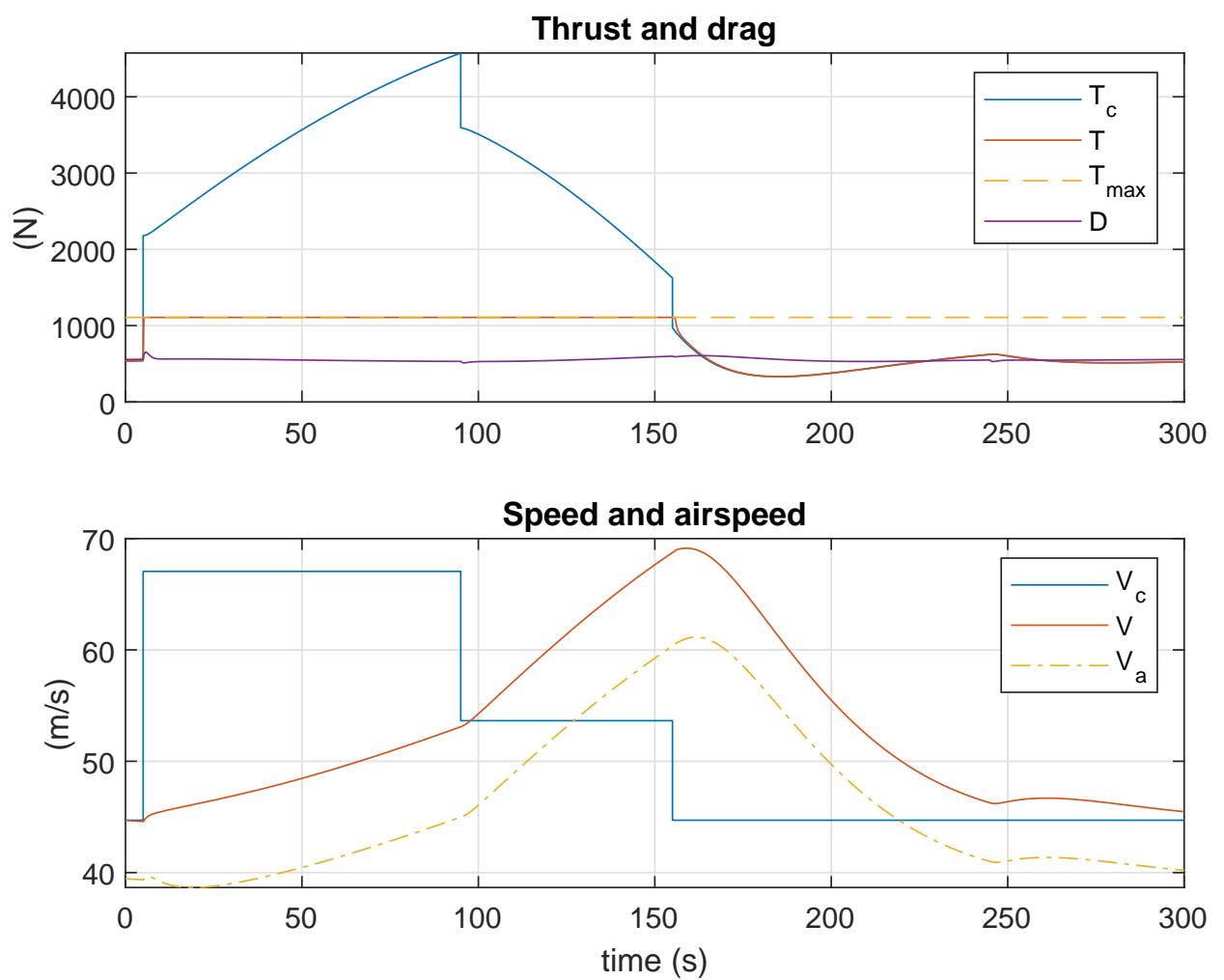


Figura 2.28: Spinta, Resistenza e Velocità 2

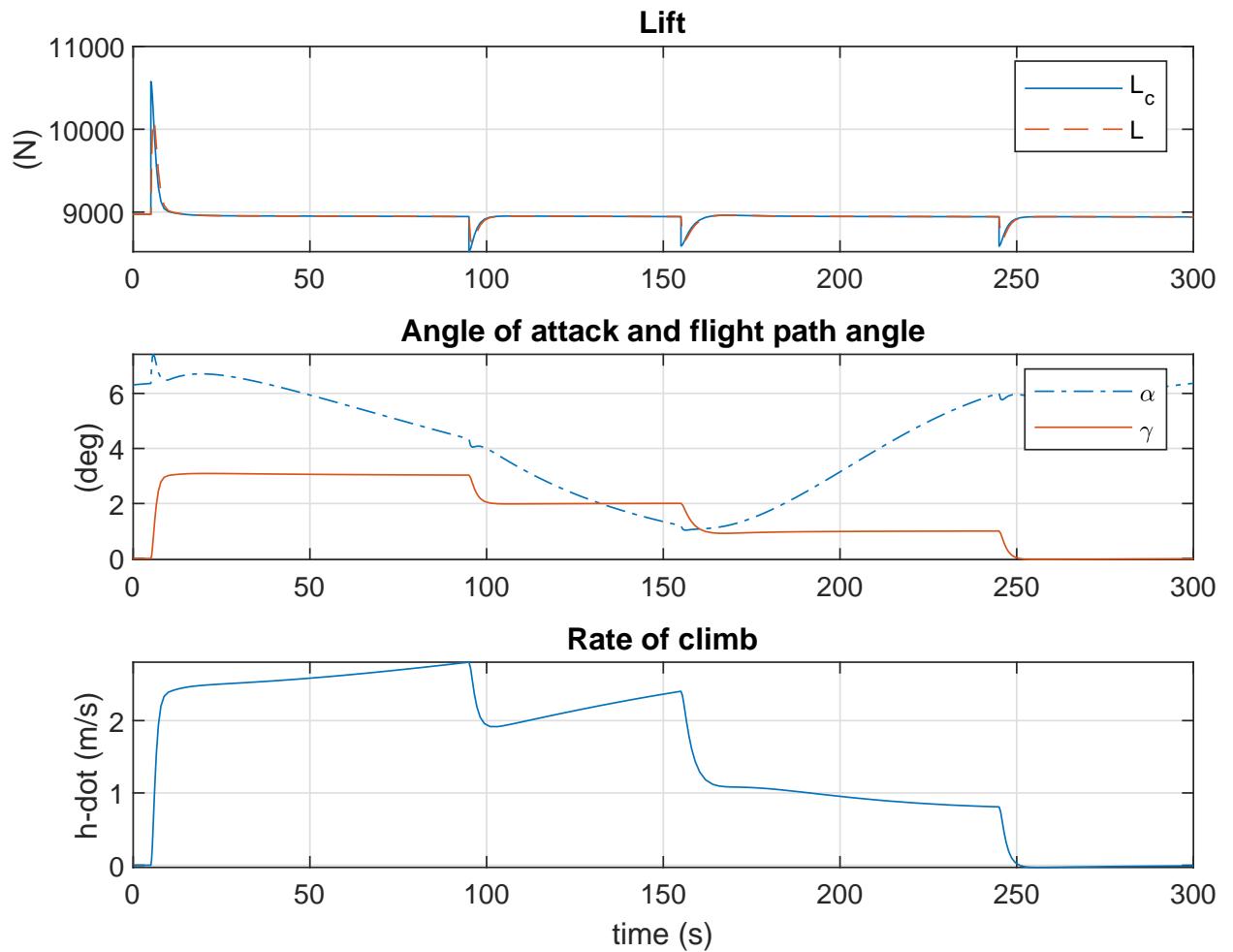


Figura 2.29: Portanza, α e γ , rateo di salita 2

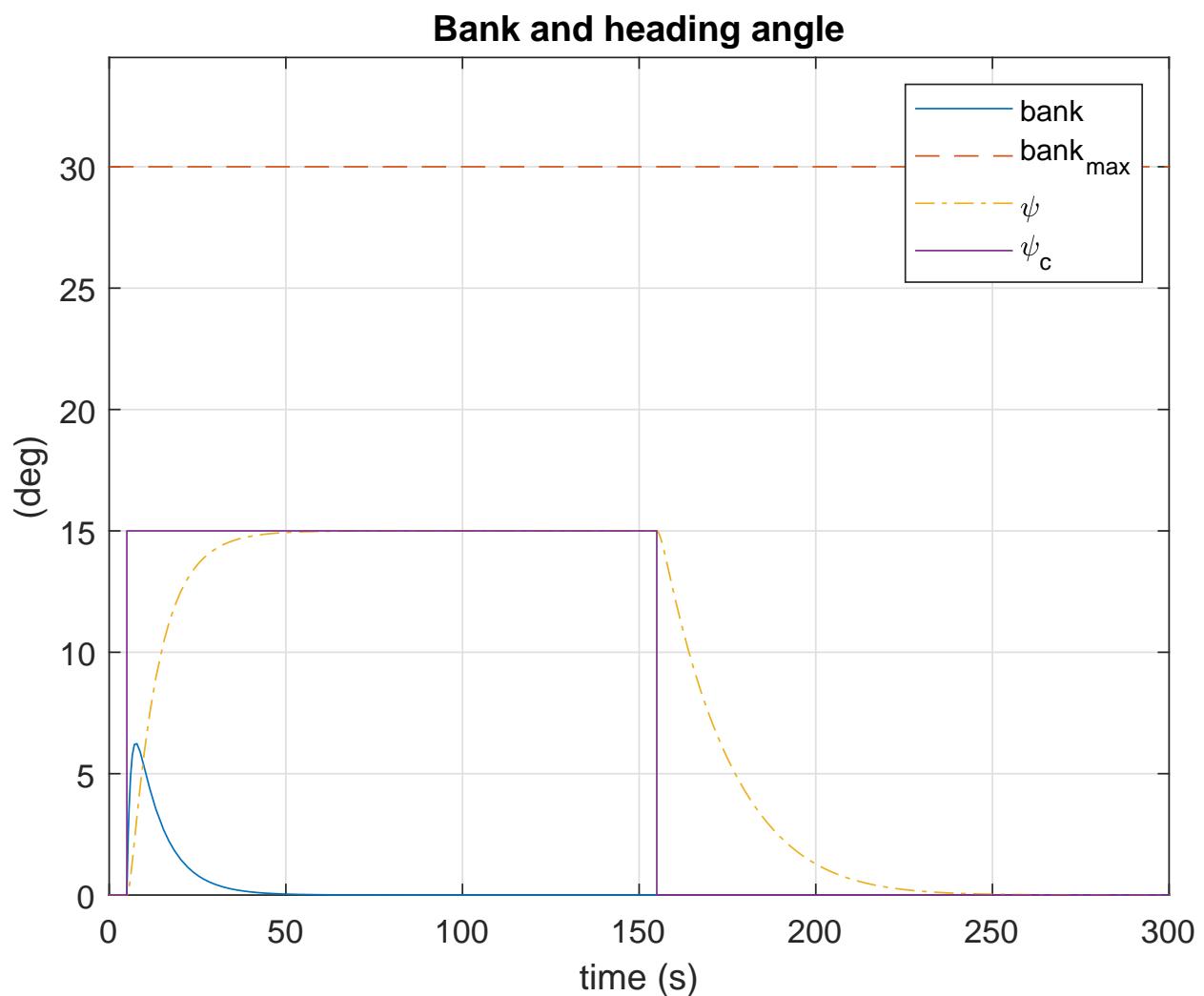


Figura 2.30: Angolo di bank e heading 2

2.6.1 Traiettoria del Centro di Massa

Per il plot del volo in 3D e delle coordinate del CG si usa lo stesso listato della sezione precedente. Il codice relativo alla funzione per il plot 3D è il seguente:

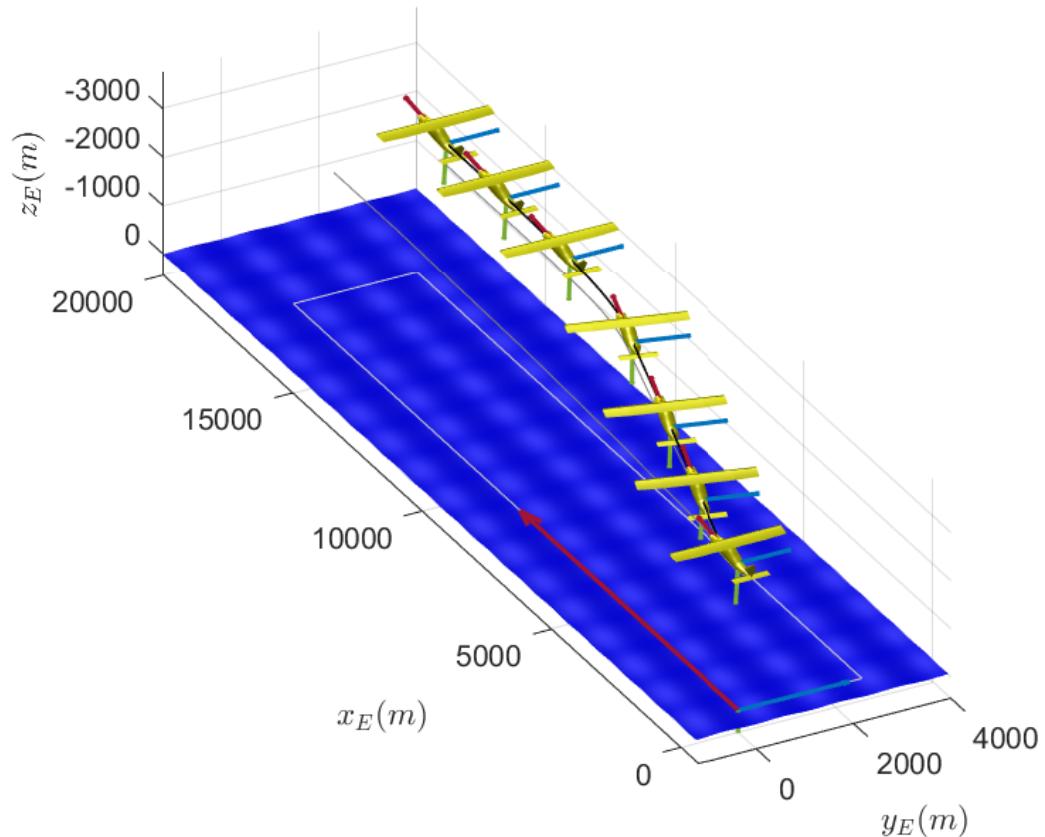


Figura 2.31: Storie temporali delle coordinate del CG 2

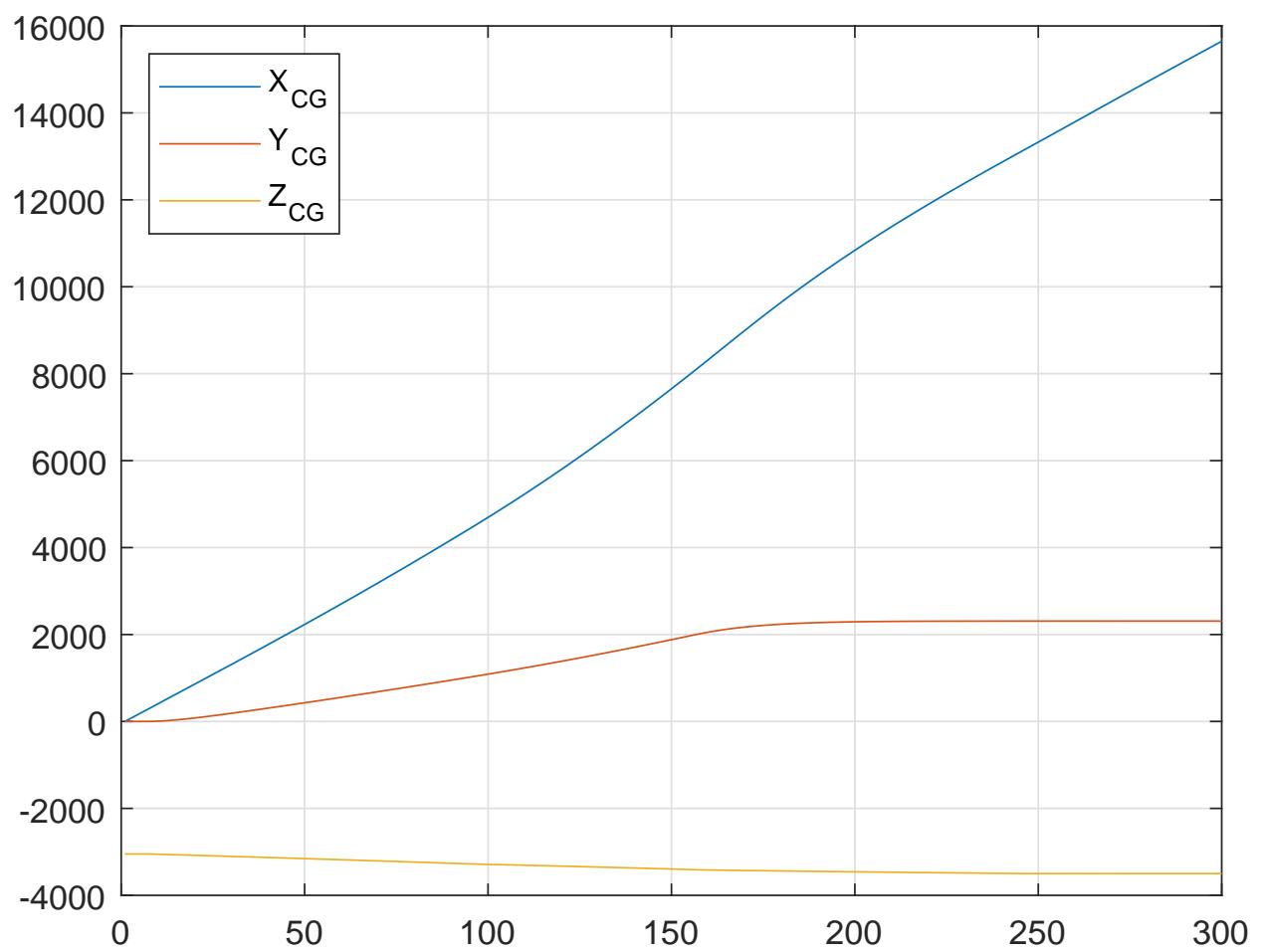


Figura 2.32: Visualizzazione 3D delle storie temporali delle CG 2

Capitolo 3

Moto longitudinal-simmetrico a comandi bloccati

3.1 Intro

Lo scopo di questa esercitazione è l'analisi del moto longitudinal-simmetrico a comandi bloccati di un velivolo rigido di massa costante e a comandi bloccati. L'ipotesi di comandi bloccati si ottiene quando il pilota esercita delle forze sui comandi di volo (sforzi), che attraverso opportune catene cinematiche si traducono in momenti di cerniera di comando. Questi contrastano i momenti aerodinamici dovuti all'interazione con la corrente, in modo da mantenere la superficie in una desiderata posizione angolare. A causa di questo vincolo all'escurzione angolare, il corpo rigido presenta 6 gradi di libertà. Pertanto la caratterizzazione di questo moto avviene mediante l'integrazione di 6 equazioni differenziali a partire da condizioni iniziali assegnate. Poichè le equazioni della dinamica sono proiettate in una terna mobile, non sono sufficienti per la determinazione del moto ma è necessario accoppiare anche delle equazioni ausiliarie, (equazioni della navigazione e gimbal equations). Si giunge così ad un sistema di 12 equazioni in 12 incognite, in cui 6 sono dinamiche (Equazione (3.1) e 6 cinematiche (Equazione (3.2)).

$$\{x_d\} = [u, v, w, p, q, r]^T \quad (3.1)$$

$$\{x_k\} = [x_{E,G}, y_{E,G}, z_{E,G}, \phi, \theta, \psi]^T \quad (3.2)$$

Bisogna ora osservare che il velivolo è caratterizzato da un moto roto-traslatorio ottenuto sovrapponendo:

- un moto traslatorio con velocità V ;
- un moto rotatorio con velocità angolare Ω ;

Riprendendo le Equazioni (2.2) ed esplicitandole rispetto alla terna di assi velivolo, si ottiene:

$$m\{\dot{V}_G\}_B + m[\tilde{\Omega}_B]_B\{V_G\}_B = \{F\}_B \quad (3.3)$$

e:

$$\{\dot{\mathcal{K}}_r\}_B + [\tilde{\Omega}_B]_B\{\mathcal{K}_r\}_B + m[\tilde{r}_G]_B\{a_C\}_B = \{\mathcal{M}\}_B \quad (3.4)$$

entrambe espresse in forma matriciale. Scegliendo come polo il baricentro del velivolo si ha $[r_G]_B = 0$. Si osserva anche che, detta $[I]_B$ la matrice di inerzia del velivolo e data la scelta della terna body rispetto alla quale tale terna è costante, possiamo dire che:

$$\{\mathcal{K}_r\}_B = [I]_B[\Omega_B]_B \quad (3.5)$$

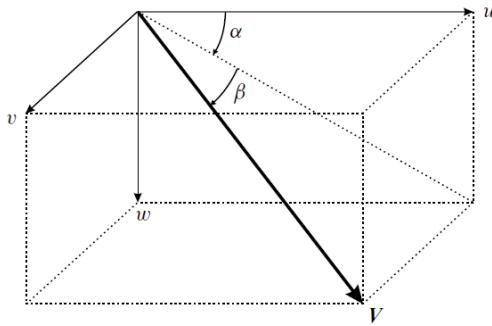
Ricordando che:

$$[I]_B = \begin{bmatrix} I_{xx} & -I_{xy} & -I_{xz} \\ -I_{xy} & I_{yy} & -I_{yz} \\ -I_{xz} & -I_{yz} & I_{zz} \end{bmatrix} \quad (3.6)$$

Quindi l'equazione (3.4) diventa:

$$[I]_B[\dot{\Omega}_B]_B + [\tilde{\Omega}_B]_B([I]_B[\Omega_B]_B) = \{\mathcal{M}\}_B \quad (3.7)$$

Figura 3.1: Legame tra le componenti del vettore velocità del vento relativo negli assi velivolo ed angoli aerodinamici (si è posto per semplicità $\alpha \equiv \alpha_B$)



Le equazioni scalari del moto di un velivolo rigido proiettate nella terna degli assi velivolo sono:

$$\begin{Bmatrix} \dot{u} \\ \dot{v} \\ \dot{w} \end{Bmatrix} = - \begin{bmatrix} 0 & -r & q \\ r & 0 & -p \\ -q & p & 0 \end{bmatrix} \begin{Bmatrix} u \\ v \\ w \end{Bmatrix} + \frac{g}{W} \begin{Bmatrix} X_G + X_A + X_T \\ Y_G + Y_A + Y_T \\ Z_G + Z_A + Z_T \end{Bmatrix} \quad (3.8)$$

$$\begin{Bmatrix} \dot{p} \\ \dot{q} \\ \dot{r} \end{Bmatrix} = \frac{1}{\det[I]_B} \begin{bmatrix} I_1 & I_2 & I_3 \\ I_2 & I_4 & I_5 \\ I_3 & I_5 & I_6 \end{bmatrix} \left(\begin{Bmatrix} \mathcal{L}_A + \mathcal{L}_T \\ \mathcal{M}_A + \mathcal{M}_T \\ \mathcal{N}_A + \mathcal{N}_T \end{Bmatrix} + \begin{bmatrix} 0 & -r & q \\ r & 0 & -p \\ -q & p & 0 \end{bmatrix} \begin{bmatrix} I_{xx} & -I_{xy} & -I_{xz} \\ -I_{xy} & I_{yy} & -I_{yz} \\ -I_{xz} & -I_{yz} & I_{zz} \end{bmatrix} \begin{Bmatrix} p \\ q \\ r \end{Bmatrix} \right) \quad (3.9)$$

Alle equazioni della dinamica proiettata in assi veicolo (3.8) e (3.9) vanno associate:

- le equazioni cinematiche che propagano nel tempo i valori delle coordinate del baricentro (navigation equations)
- quelle che propagano gli angoli di Eulero (gimbal equations)

$$\begin{Bmatrix} \dot{x}_{E,G} \\ \dot{y}_{E,G} \\ \dot{z}_{E,G} \end{Bmatrix} = \begin{bmatrix} C_\theta C_\psi & S_\phi S_\theta C_\psi - C_\phi S_\psi & C_\phi S_\theta C_\psi + S_\phi S_\psi \\ C_\theta S_\psi & S_\phi S_\theta S_\psi + C_\phi C_\psi & C_\phi S_\theta S_\psi - S_\phi C_\psi \\ -S_\theta & S_\phi C_\theta & C_\phi C_\theta \end{bmatrix} \begin{Bmatrix} u \\ v \\ w \end{Bmatrix} \quad (3.10)$$

$$\begin{Bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{Bmatrix} = \begin{bmatrix} 1 & \frac{S_\phi S_\theta}{C_\theta} & \frac{C_\phi S_\theta}{C_\theta} \\ 0 & C_\phi & -S_\phi \\ 0 & \frac{S_\phi}{C_\theta} & \frac{C_\phi}{C_\theta} \end{bmatrix} \begin{Bmatrix} p \\ q \\ r \end{Bmatrix} \quad (3.11)$$

Verranno anche utilizzate le seguenti equazioni:

$$\begin{cases} \dot{V} = \frac{1}{m} [-D \cos\beta + Y_A \sin\beta + X_T \cos\alpha \cos\beta + Z_T \sin\alpha \cos\beta \\ \quad -mg (\cos\alpha \cos\beta \sin\theta - \sin\beta \sin\phi \cos\theta - \sin\alpha \cos\beta \cos\phi \cos\theta)] \\ \dot{\alpha} = \frac{1}{mV \cos\beta} [-L + Z_T \cos\alpha - X_T \sin\alpha + mg (\cos\alpha \cos\phi \cos\theta + \sin\alpha \sin\theta)] \\ \quad +q - \tan\beta(p \cos\alpha + r \sin\alpha) \\ \dot{\beta} = \frac{1}{mV} [D \sin\beta + Y_A \cos\beta - X_T \cos\alpha \sin\beta + Y_T \cos\beta - Z_T \sin\alpha \sin\beta \\ \quad +mg (\cos\alpha \sin\beta \sin\theta + \cos\beta \sin\phi \cos\theta - \sin\alpha \sin\beta \cos\phi \cos\theta)] \\ \quad p \sin\alpha - r \cos\alpha \end{cases} \quad (3.12)$$

Rendendo l'equazione (3.1) la seguente:

$$\{x_d\} = [V, \alpha, \beta, p, q, r]^T \quad (3.13)$$

Sussistendo le seguenti equazioni:

$$\begin{cases} u = V \cos\beta \cos\alpha \\ v = V \sin\beta \\ w = V \cos\beta \sin\alpha \end{cases} \quad (3.14)$$

$$V = \sqrt{u^2 + v^2 + w^2} \quad (3.15)$$

Gli angoli di attacco e di derapata, note le componenti del vettore V, si possono ricavare invertendo le equazioni precedenti. Si ottiene:

$$\begin{cases} \alpha = \tan^{-1} \frac{w}{u} \\ \beta = \sin^{-1} \frac{v}{V} \end{cases} \quad (3.16)$$

Le precedenti considerazioni sono esplicite nella Figura 3.1.

Sono sorgenti di non linearità:

- accoppiamenti cinematici: (rv,qw,... e p senα, p cosα...) ad esempio da β, il rollio ha un effetto di conversione dell'AoA in angolo di derapata. Gli accoppiamenti cinematici rendono interdipendenti le dinamiche di α e β in presenza di velocità angolare;
- accoppiamenti inerziali: (pr,qr,pq);
- gravità: i prodotti di g per le funzioni trigonometriche sono non lineari;
- aerodinamica: le forze e i momenti aerodinamici degli angoli di eulero sono non lineari;
- propulsione: è funzione non lineare del Mach e di z.

3.1.1 Moto a 3 Gradi di Libertà

Un caso particolare è il moto a 3 DoF (scendono da 6 a 3). A tale scopo si ipotizza il velivolo simmetrico, che vola ad ali livellate ($\phi(t) = 0$). Il velivolo mantiene il suo piano di simmetria $x_B z_B$ costantemente verticale. L'angolo di prua è fissato quindi per tutto il moto ($\psi(t) = 0$). Si avrà: $\nu(t) = \beta(t) = 0$, $p(t) = r(t) = 0$, $Y(t) = \mathcal{L}(t) = \mathcal{N}(t) = 0$, ovvero:

$$\dot{\beta} = \dot{p} = \dot{r} = \dot{\psi} = \dot{\theta} = \dot{Y}_{G_E} = 0$$

E' possibile dunque trascurare

- la 2^a equazione dell'equilibrio alla traslazione(u,v,w) o la 3^a se scelgo(V,α,β);
- la 1^a e la 3^a equazione dell'equilibrio alla rotazione;
- la 2^a delle equazioni della navigazione;
- la 1^a e la 3^a delle Gimbal equations.

Quindi:

$$\{x_d\}_{3-DoF} = [V, \alpha, q]^T \quad \{x_k\}_{3-DoF} = [x_{EG}, z_{EG}, \theta]^T \quad (3.17)$$

Per ipotesi di moto longitudinal-simmetrico:

$$\delta_a(t) = \delta_r(t) = 0 \quad (3.18)$$

Se è un bimotore si aggiunge:

$$\delta_{T_{Left}}(t) = \delta_{T_{Right}}(t) = \delta_T(t) \quad (3.19)$$

Quindi si ottengono:

$$\begin{cases} \dot{V} = \frac{1}{m} [-D \cos\beta + X_T \cos\alpha_B + Z_T \sin\alpha_B - mg (\cos\alpha_B \sin\theta - \sin\alpha_B \cos\theta)] \\ \dot{\alpha} = \frac{1}{mV} [-L + Z_T \cos\alpha_B - X_T \sin\alpha_B + mg (\cos\alpha_B \cos\theta + \sin\alpha_B \sin\theta)] + q \\ I_y \dot{q} = \mathcal{M}_A + \mathcal{M}_T \end{cases} \quad (3.20)$$

$$\begin{cases} \dot{x}_{E,G} = (V \cos\alpha_B) \cos\theta + (V \sin\alpha_B) \sin\theta \\ \dot{z}_{E,G} = -(V \cos\alpha_B) \sin\theta + (V \sin\alpha_B) \cos\theta \\ \dot{\theta} = q \end{cases} \quad (3.21)$$

In cui $u = V \cos\alpha_B$ e $w = V \sin\alpha_B$. E' possibile porre:

$$X_T = T \cos\mu_T, \quad Z_T = -T \sin\mu_T, \quad \mathcal{M}_T = Te_T + \mathcal{M}_T^\star \quad (3.22)$$

3.2 Esercizio 1

Nel seguente esercizio si procederà alla risoluzione delle equazioni della dinamica del volo per il velivolo *F/A-18 High Angle-of-attack Research Vehicle* (HARV). Il modello aerodinamico si basa sulle equazioni discusse nella precedente sezione. Per risolvere è necessario esplicitare le forze e i momenti risultanti dalle coppie aerodinamiche e propulsive, di seguito riportati.

$$D = \frac{1}{2} \rho V^2 S C_D \quad Y_A = \frac{1}{2} \rho V^2 S C_Y \quad L = \frac{1}{2} \rho V^2 S C_L \quad (3.23)$$

$$\mathcal{L}_A = \frac{1}{2} \rho V^2 S b C_{\mathcal{L}} \quad \mathcal{M}_A = \frac{1}{2} \rho V^2 S b C_{\mathcal{M}} \quad \mathcal{N}_A = \frac{1}{2} \rho V^2 S b C_{\mathcal{N}} \quad (3.24)$$

Per questo esercizio assumeremo un vettore spinta baricentrico ($e_T = 0$) e allineato con l'asse x_B ($\mu_T = 0$), dunque:

$$X_T = T(V, -z_{E,G}, \delta_T) \quad Y_T = 0 \quad Z_T = 0 \quad (3.25)$$

$$\mathcal{L}_A = 0 \quad \mathcal{M}_A = 0 \quad \mathcal{N}_A = 0 \quad (3.26)$$

Noti tali valori occorre esprimere nel sistema di riferimento body. Utilizzeremo le seguenti:

$$\begin{cases} X_A = -D \cos\alpha + L \sin\alpha \\ Y_A = -C \\ Z_A = -D \sin\alpha - L \cos\alpha \end{cases} \quad (3.27)$$

$$\begin{cases} X_G = -mg \sin\theta \\ Y_G = mg \sin\phi \cos\theta \\ Z_G = mg \cos\phi \cos\theta \end{cases} \quad (3.28)$$

$$\begin{cases} X_{tot} = X_A + X_G + X_T \\ Y_{tot} = Y_A + Y_G + Y_T \\ Z_{tot} = Z_A + Z_G + Z_T \end{cases} \quad (3.29)$$

Le caratteristiche del velivolo sono:

Caratteristiche di massa e inerziali in assi body del velivolo HARV			
Massa	m=W/g	1036 slug	15119 kg
Momento di inerzia	I_{xx}	23000 slug ft^2	31184 kg/m^2
Momento di inerzia	I_{yy}	151293 slug ft^2	205130 kg/m^2
Momento di inerzia	I_{zz}	169945 slug ft^2	230410 kg/m^2
Momento di inerzia	I_{xz}	0 slug ft^2	0 kg/m^2
Caratteristiche geometriche del velivolo HARV			
Superficie alare	S	400 ft^2	37.16 m^2
Apertura alare	b	37.42 ft	11.41 m^2
Corda media aerodinamica	c	11.52 ft	3.51 m

3.2.1 Condizioni Iniziali

Per quanto concerne i valori iniziali, si è optato per un approccio che consente all'utente di scegliere i maggiori parametri del problema (si veda la Figura 3.2). Di seguito il listato che svolge il problema assegnato.

```

1 function [] = func71(IN)
2 %load("input7_1.mat");
3 diary('ex7_1.txt')
4 diary on
5 disp('Moto del velivolo a 6 gradi di libertà');
6 %% Dichiaraione delle variabili globali
7 global g...          %Accelerazione di gravità
8           myAC          %Oggetto 'Velivolo'

```

Input in gradi e metri.		HARV (7.1)	
phi0	0	Grafica	
theta0	4.46		
psi0	0	delta_a0	0
zEG_0	0	delta_e0	-10.61
q0	0	delta_r0	0
p0	0	delta_T0	0.4749
r0	0	alpha_B0	4.558
M0	0.246	beta_d...	0.4749

Figura 3.2: Dati in INPUT esercizio 7.1

```

9
10 % Definizione della classe DSVAircraft e dell'oggetto 'Velivolo'
11 aircraftDataFileName = 'HARV.txt';
12 myAC = DSVAircraftHARV(aircraftDataFileName);
13
14 % Costanti e condizioni iniziali
15 g = 9.81; %Accelerazione di gravità [m/s^2]
16 xEG_0 = 0; %[m]
17 yEG_0 = 0;%[m]
18 zEG_0 = IN(10); %Altitudine [m]
19 [air_Temp0,sound_speed0,air_pressure0,rho0] = atmosisa(-zEG_0);
20 phi0 = convang(IN(14),'deg','rad'); %Angolo di bank
21 theta0 = convang(IN(13),'deg','rad'); %Angolo di elevazione
22 psi0 = convang(IN(12),'deg','rad'); %Angolo di heading
23 p0 = convangvel(IN(9),'deg/s','rad/s'); %Velocità angolare di rollio
24 q0 = convangvel(IN(10),'deg/s','rad/s'); %Velocità angolare di beccheggio
25 r0 = convangvel(IN(8),'deg/s','rad/s'); %Velocità angolare di imbardata
26 M0 = IN(7); %Numero di Mach
27 VO = M0*soundspeed0; %Velocità lineare del baricentro
28 alpha_B0 = convang(IN(2),'deg','rad'); %Angolo di attacco valutato rispetto l'asse x Body
29 beta_der0 = convang(IN(1),'deg','rad'); %Angolo di derapata
30 u0 = VO*cos(beta_der0)*cos(alpha_B0); %Componente della velocità lineare del baricentro lungo l'asse x Body
31 v0 = VO*sin(beta_der0); %Componente della velocità lineare del baricentro lungo l'asse y Body
32 w0 = VO*cos(beta_der0)*sin(alpha_B0); %Componente della velocità lineare del baricentro lungo l'asse z Body
33
34 % Comandi di volo iniziali
35 delta_a0 = convang(IN(6),'deg','rad');
36 delta_e0 = convang(IN(5),'deg','rad');
37 delta_r0 = convang(IN(4),'deg','rad');
38 delta_T0 = IN(3);
39
40 %% Integrazione delle equazioni del moto a 6-DoF
41 t_fin = 230; %Tempo di simulazione [s] (Manovra di avvicinamento alla pista di atterraggio di un aeroporto)
42 state_0 = [u0,v0,w0,p0,q0,r0,xEG_0,yEG_0,zEG_0,phi0,theta0,psi0];
43
44 % Assegnazione delle leggi temporali dei comandi di volo
45 global delta_a delta_e delta_r delta_T
46
47 %Possibile manovra di avvicinamento alla pista di atterraggio di un aeroporto
48 delta_a_exc1 = convang(-20,'deg','rad');
49 delta_a = @(t) interp1([0, 40, 41, 45, 46, t_fin],...
50 [delta_a0, delta_a0, delta_a_exc1, delta_a_exc1, delta_a0, delta_a0],t,'linear');
51
52 delta_e_exc1 = convang(-9.5,'deg','rad');
53 delta_e_exc2 = convang(-10,'deg','rad');
54 delta_e = @(t) interp1([0, 20, 80, 180, t_fin],...
55 [delta_e0, delta_e0, delta_e_exc1, delta_e_exc2, delta_e_exc2],t,'linear');
56
57 delta_r_exc1 = convang(10,'deg','rad');
58 delta_r = @(t) interp1([0, 200, 203, 205, 208, t_fin],...
59 [delta_r0, delta_r0, delta_r_exc1, delta_r_exc1, delta_r0, delta_r0],t,'linear');

```

```

60
61 delta_T_excl = 0.43;
62 delta_T = @(t) interp1([0, t_fin],[delta_T0 delta_T_excl],t,'linear');
63
64 % Integrazione del sistema di equazioni differenziali
65 options = odeset('RelTol',1e-9,'AbsTol',1e-9*ones(1,12));
66 [vTime,mState] = ode45(@eqLongDynamicStickFixed_6DoF_uwppqr_Euler,[0 t_fin],state_0,options);
67
68 vVel_u = mState(:,1);
69 vVel_v = mState(:,2);
70 vVel_w = mState(:,3);
71 vVel = sqrt(vVel_u.^2 + vVel_v.^2 + vVel_w.^2);
72 vAlpha_B = convang(atan(vVel_w./vVel_u),'rad','deg');
73 vBeta = convang(asin(vVel_v./vVel),'rad','deg');
74 v_p = convangvel(mState(:,4),'rad/s','deg/s');
75 v_q = convangvel(mState(:,5),'rad/s','deg/s');
76 v_r = convangvel(mState(:,6),'rad/s','deg/s');
77 vXe = mState(:,7);
78 vYe = mState(:,8);
79 vZe = mState(:,9);
80 vPhi = convang(mState(:,10),'rad','deg');
81 vTheta = convang(mState(:,11),'rad','deg');
82 vPsi = convang(mState(:,12),'rad','deg');
83 vGamma = convang(asin(cos(vAlpha_B).*cos(vBeta).*sin(vTheta))-...
84 sin(vBeta).*sin(vPhi).*cos(vTheta)-...
85 sin(vAlpha_B).*cos(vBeta).*cos(vPhi).*cos(vTheta)), 'rad','deg');
86 mQuat = zeros(length(vTime),4);
87 for i = 1:length(vTime)
88 mQuat(i,:) = angle2quat(mState(i,12),mState(i,11),mState(i,10),'ZYX');
89 end
90
91 %% Grafica
92 % Diagrammi dei comandi di volo
93 Figura1=figure(1);
94 subplot 411
95 plot(vTime,convang(delta_a(vTime),'rad','deg'),'b-','LineWidth',1.5);
96 grid on
97 xlim([0 t_fin])
98 ylim([-25 5])
99 ylabel('$\delta_a$ (deg)', 'interpreter', 'latex', 'fontsize', 11);
100 subplot 412
101 plot(vTime,convang(delta_e(vTime),'rad','deg'),'b-','LineWidth',1.5);
102 grid on
103 xlim([0 t_fin])
104 ylim([-11 -9])
105 ylabel('$\delta_e$ (deg)', 'interpreter', 'latex', 'fontsize', 11);
106 subplot 413
107 plot(vTime,convang(delta_r(vTime),'rad','deg'),'b-','LineWidth',1.5);
108 grid on
109 xlim([0 t_fin])
110 ylim([-3 12])
111 ylabel('$\delta_r$ (deg)', 'interpreter', 'latex', 'fontsize', 11);
112 subplot 414
113 plot(vTime,delta_T(vTime),'b-','LineWidth',1.5);
114 grid on
115 xlim([0 t_fin])
116 ylim([0 1])
117 xlabel('t (s)', 'interpreter', 'latex', 'fontsize', 11);
118 ylabel('$\delta_T$', 'interpreter', 'latex', 'fontsize', 11);
119
120 %Storie temporali delle variabili di stato
121 Figura2=figure(2);
122 plot(vTime,vVel_u,'-','color',[0.6350, 0.0780, 0.1840],'LineWidth',1.5);
123 hold on;
124 plot(vTime,vVel_v,'--','color',[0, 0.4470, 0.7410],'LineWidth',1.5);
125 plot(vTime,vVel_w,':', 'color',[0.4660, 0.6740, 0.1880],'LineWidth',1.5);
126 grid on
127 lgd = legend('$u(t)$','$v(t)$','$w(t)$');
128 lgd.Interpreter = 'latex';
129 lgd.FontSize = 11;
130 xlim([0 t_fin])
131 xlabel('t (s)', 'interpreter', 'latex', 'fontsize', 11);
132 ylim([-30 150])
133 ylabel('$(m/s)$', 'interpreter', 'latex', 'fontsize', 11)
134
135 Figura3=figure(3);
136 subplot 611
137
138 plot(vTime,vVel,'-','LineWidth',1.5);
139 grid on
140 xlim([0 t_fin])
141 ylim([80 130])
142 ylabel('V (m/s)', 'interpreter', 'latex', 'fontsize', 11)
143 subplot 612
144 plot(vTime,vAlpha_B,'-','LineWidth',1.5);
145 grid on
146 xlim([0 t_fin])
147 ylim([-5 10])
148 ylabel('$\alpha_B$ (deg)', 'interpreter', 'latex', 'fontsize', 11)

```

```

149 subplot 613
150 plot(vTime,vBeta,'-.','LineWidth',1.5);
151 grid on
152 xlim([0 t_fin])
153 ylim([-10 20])
154 ylabel('$\beta$ (deg)', 'interpreter', 'latex', 'fontsize', 11)
155 subplot 614
156 plot(vTime,v_p,'-.','LineWidth',1.5);
157 grid on
158 xlim([0 t_fin])
159 ylim([-5 10])
160 ylabel('p (deg/s)', 'interpreter', 'latex', 'fontsize', 11)
161 subplot 615
162 plot(vTime,v_q,'-.','LineWidth',1.5);
163 grid on
164 xlim([0 t_fin])
165 ylim([-5 5])
166 ylabel('q (deg/s)', 'interpreter', 'latex', 'fontsize', 11)
167 subplot 616
168 plot(vTime,v_r,'-.','LineWidth',1.5);
169 grid on
170 xlim([0 t_fin])
171 xlabel('t (s)', 'interpreter', 'latex', 'fontsize', 11);
172 ylim([-8 10])
173 ylabel('r (deg/s)', 'interpreter', 'latex', 'fontsize', 11)
174
175 Figura4=figure(4);
176 subplot 611
177 plot(vTime,vXe,'-.','LineWidth',1.5);
178 grid on
179 xlim([0 t_fin])
180 ylim([-4000 8000])
181 ylabel('x_{EG} (m)', 'interpreter', 'latex', 'fontsize', 11)
182 subplot 612
183 plot(vTime,vYe,'-.','LineWidth',1.5);
184 grid on
185 xlim([0 t_fin])
186 ylim([-2500 8000])
187 ylabel('y_{EG} (m)', 'interpreter', 'latex', 'fontsize', 11)
188 subplot 613
189 plot(vTime,-(vZe - zEG_0),'-.','LineWidth',1.5);
190 grid on
191 xlim([0 t_fin])
192 ylim([-2000 500])
193 ylabel('Delta h (m)', 'interpreter', 'latex', 'fontsize', 11)
194 subplot 614
195 plot(vTime,vPhi,'-.','LineWidth',1.5);
196 grid on
197 xlim([0 t_fin])
198 ylim([-10 50])
199 ylabel('phi (deg)', 'interpreter', 'latex', 'fontsize', 11)
200 subplot 615
201 plot(vTime,vTheta,'-.','LineWidth',1.5);
202 grid on
203 xlim([0 t_fin])
204 ylim([-20 10])
205 ylabel('theta (deg)', 'interpreter', 'latex', 'fontsize', 11)
206 subplot 616
207 plot(vTime,vPsi,'-.','LineWidth',1.5);
208 grid on
209 xlim([0 t_fin])
210 xlabel('t (s)', 'interpreter', 'latex', 'fontsize', 11);
211 ylim([-200 400])
212 ylabel('psi (deg)', 'interpreter', 'latex', 'fontsize', 11)
213
214 Figura5=figure(5);
215 plot(vTime,mQuat(:,1),'-','color',[0.4940, 0.1840, 0.5560],'LineWidth',1.5);
216 hold on;
217 plot(vTime,mQuat(:,2),'-','color',[0.6350, 0.0780, 0.1840],'LineWidth',1.5);
218 plot(vTime,mQuat(:,3),'-','color',[0, 0.4470, 0.7410],'LineWidth',1.5);
219 plot(vTime,mQuat(:,4),':','color',[0.4660, 0.6740, 0.1880],'LineWidth',1.5);
220 grid on
221 lgd = legend('$q_x(t)$','$q_y(t)$','$q_z(t)$','$q_w(t)$');
222 lgd.Interpreter = 'latex';
223 lgd.FontSize = 11;
224 xlim([0 t_fin])
225 ylim([-1 1.2])
226 xlabel('t (s)', 'interpreter', 'latex', 'fontsize', 11);
227 ylabel('')
228
229 exportgraphics(Figura1,'Figura1.png')
230 exportgraphics(Figura2,'Figura2.png')
231 exportgraphics(Figura3,'Figura3.png')
232 exportgraphics(Figura4,'Figura4.png')
233 exportgraphics(Figura5,'Figura5.png')
234 close all;
235 diary off
236 end

```

3.2.2 Risultati

Vengono infine mostrati i risultati relativi ai dati assegnati in Figura 3.2. Tali risultati sono riportati in Figura 3.4.

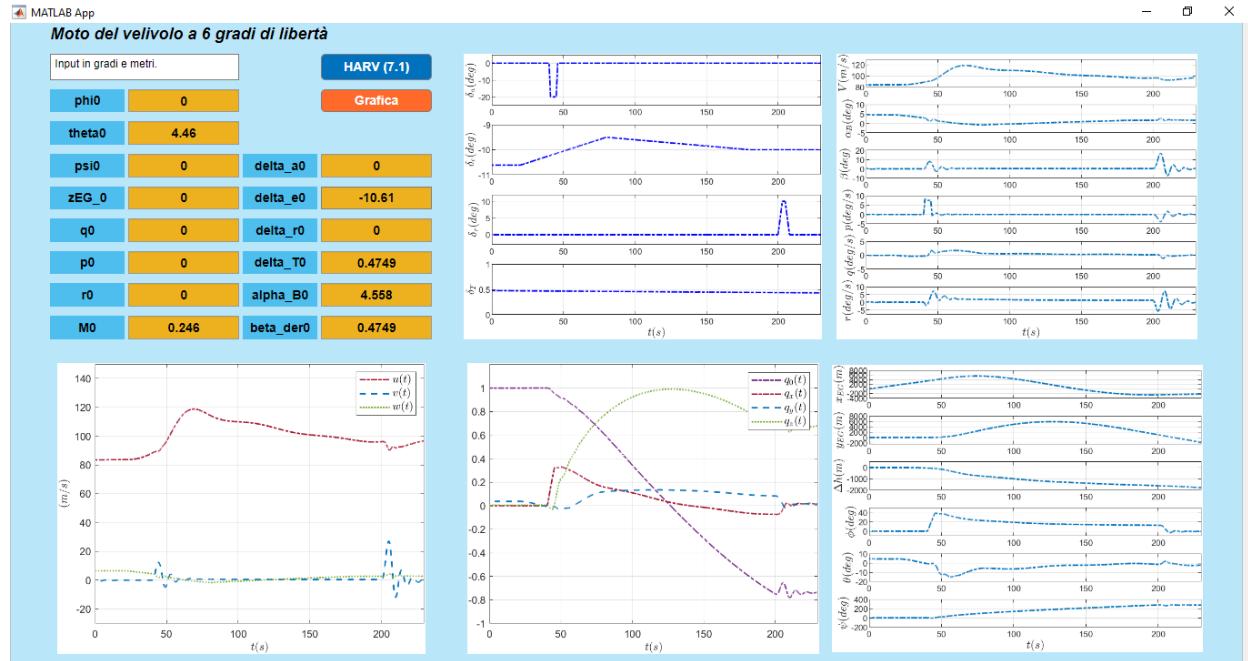
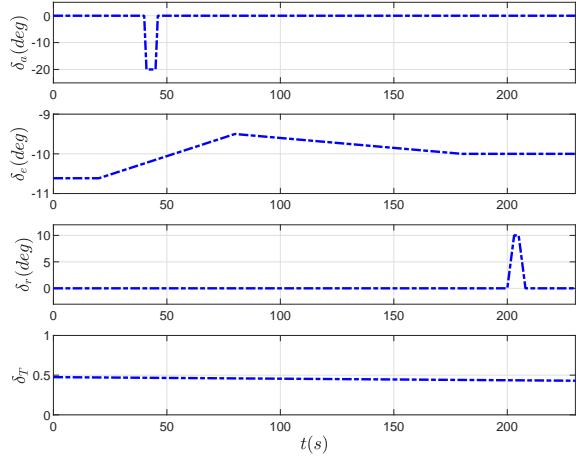
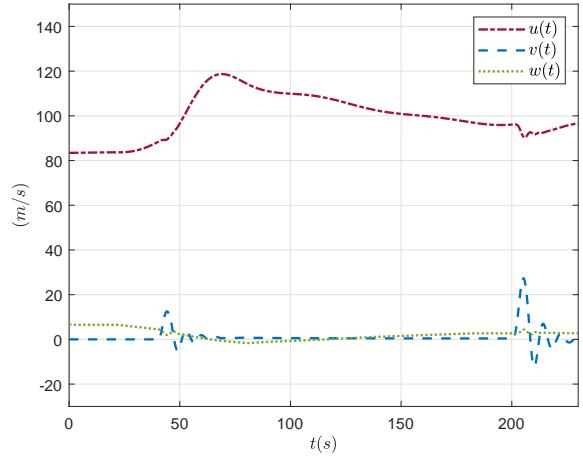


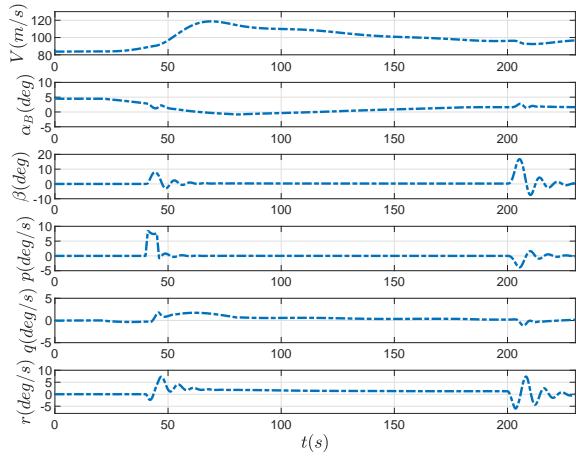
Figura 3.3: Vista dell'esercizio 7.1 in APP



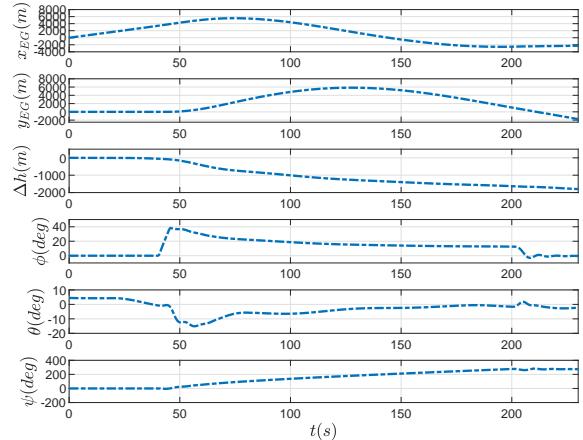
(a) Leggi di comando HARV 7.1



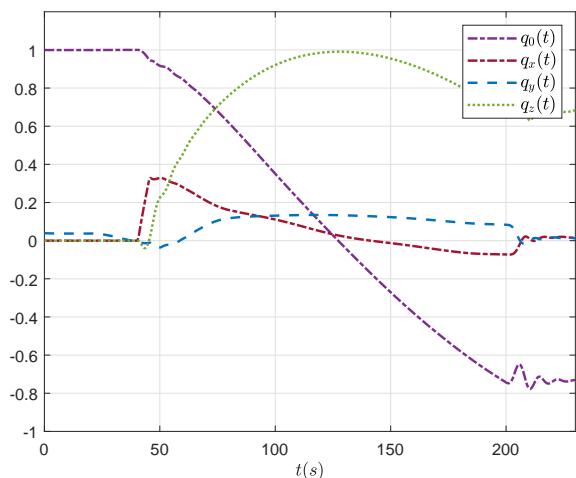
(b) Componenti del vettore Velocità del CG



(c) Velocità del CG, α , β e componenti della velocità angolare



(d) Coordinate del CG e angoli di Eulero



(e) Componenti del quaternione



(f) Velivolo HARV

Figura 3.4: Storie temporali dell'HARV - Esercizio 7.1

3.3 Esercizio 2 (7.6-7.7-7.11) - Ricerca delle condizioni di Trim per il moto a 3 DoF

In questo esercizio si procede ad integrare numericamente le equazioni del moto 3-DoF assegnando una legge del movimento non nulla del comando longitudinale ($\delta_t(t)$). Dall'interfaccia dell'APP è possibile scegliere la velocità iniziale V_0 e confrontarla con al più altre due velocità iniziali.

I dati in Input sono dunque:

- l'escursione della manetta δ_t in radianti
- il numero di velocità considerate
- i valori delle velocità considerate (in m/s)
- quota in m
- velocità angolare di beccheggio iniziale
- angolo di rampa iniziale

Input Dati	
delta_T_excursion	0.3
NVel	3
Velocità 1	220
Velocità 2	285
Velocità 3	350
Quota	-4000
Vel ang di beccheggio	0
Angolo di rampa	0

Figura 3.5: Dati in Input - Esercizio 2

Il presente listato presenta la risoluzione dell'esercizio.

```

1 function [] = func7ult(InputUlterioreCap7)
2 diary('ex7_ult.txt')
3 diary on
4 disp('Moto del velivolo a 3 gradi di libertà');
5 disp('Risoluzione del problema di trim ad una data altitudine e velocità di volo');
6
7 %% INPUT
8 %load("InputUlterioreCap7.mat");
9 delta_T_excursion = InputUlterioreCap7(1);
10 NVel = InputUlterioreCap7(2);
11 vVel0 = [InputUlterioreCap7(3),InputUlterioreCap7(4),InputUlterioreCap7(5)];
12
13 %% Dichiarazione delle variabili globali
14 global g... %Accelerazione di gravità
15 zEG_0 q0 gamma0... %Condizioni iniziali
16 rho0 ... %Densità dell'aria all'altitudine h = (-zEG_0)
17 myAC %Oggetto 'Velivolo'
18
19 %% Definizione della classe DSVAircraft e dell'oggetto 'Velivolo'
20 aircraftDataFileName = 'DSV_Aircraft_data.txt';
21
22 % Definizione dell'oggetto 'Velivolo'
23 myAC = DSVAircraft(aircraftDataFileName);
24
25 if (myAC.err == -1)
26     disp('Terminazione.')
27 else

```

```

28 disp(['File ',aircraftDataFileName,' letto correttamente.']);
29
30 % Costanti e condizioni iniziali
31 g = 9.81; %Accelerazione di gravità [m/s^2]
32 xEG_0 = 0; %[m]
33 zEG_0 = InputUlterioreCap7(6); %Altitudine [m]
34 %VO = vVeloIN; %Velocità di volo [m/s];
35 q0 = convangvel( InputUlterioreCap7(7), 'deg/s','rad/s'); %Velocità angolare di beccheggio
36 gamma0 = convang( InputUlterioreCap7(8), 'deg','rad'); %Angolo di rampa
37 [air_Temp0,sound_speed0,air_pressure0,rho0] = atmosisa(-zEG_0);
38
39 %% Processo di minimizzazione della funzione di costo
40 % Valore di tentativo iniziale per il design vector
41 x0 = [0; %Valore di tentativo iniziale per alpha0 [rad]
42 0; %Valore di tentativo iniziale per delta_e0 [rad]
43 0; %Valore di tentativo iniziale per delta_s0 [rad]
44 0.5]; %Valore di tentativo iniziale per delta_T0
45
46 % Definizione del vincolo imposto al parametro delta_s_0
47 Aeq = zeros(4);
48 Aeq(3,3) = 1;
49 delta_s_0 = convang(0.000,'deg','rad');
50 beq = zeros(4,1);
51 beq(3,1) = delta_s_0;
52 %La funzione 'fmincon' consente di imporre una condizione di vincolo ai
53 %parametri rispetto ai quali si vuol ricercare il punto di minimo della
54 %funzione. Trattandosi di una relazione di uguaglianza, il sistema
55 %matriciale da compilare è del tipo [Aeq]\{csi} = [beq], dove \{csi} è
56 %il design vector avente per componenti [alpha, delta_e, delta_s, delta_T].
57 %Nel caso esaminato l'unica tra le quattro a non fornire un'identità è la
58 %terza equazione.
59
60 % Limiti
61 lb = [convang(-15,'deg','rad'),... %Valore minimo per alpha
62 convang(-20,'deg','rad'),... %Valore minimo per delta_e
63 convang(-5,'deg','rad'),... %Valore minimo per delta_s
64 0.2]; %Valore minimo per delta_T
65 ub = [convang(15,'deg','rad'),... %Valore massimo per alpha
66 convang(13,'deg','rad'),... %Valore massimo per delta_e
67 convang(5,'deg','rad'),... %Valore massimo per delta_s
68 1.0]; %Valore massimo per delta_T
69
70 % Opzioni di ricerca del minimo
71 options = optimset('tolfun',1e-9,'Algorithm','interior-point');
72
73 % Inizializzazione dei vettori contenenti le variabili di output
74 alpha0_deg = zeros(NVel,1);
75 delta_e0_deg = zeros(NVel,1);
76 delta_s0_deg = zeros(NVel,1);
77 delta_T0 = zeros(NVel,1);
78
79 for i = 1:NVel
80
81 % Chiamata alla funzione 'fmincon'
82 % vVelo = VO(i);
83 [x,fval] = fmincon(@(x) costLongEquilibriumStaticStickFixedVel(x,vVelo(i)),...
84 x0,...
85 [],[],Aeq,beq,...
86 lb,ub,...
87 @myNonLinearConstraint,...
88 options);
89
90 alpha0_rad(i) = x(1);
91 alpha0_deg(i) = convang(x(1),'rad','deg');
92 theta0_rad(i) = alpha0_rad(i) - myAC.mu_x + gamma0;
93 theta0_deg = convang(theta0_rad,'rad','deg');
94 delta_e0_rad = x(2);
95 delta_e0_deg(i) = convang(x(2),'rad','deg');
96 delta_s0_rad = x(3);
97 delta_s0_deg(i) = convang(x(3),'rad','deg');
98 delta_T0 = x(4);
99
100 end
101
102
103
104 end
105
106 %% Integrazione delle equazioni del moto a 3-DoF
107 t_fin = 120; %Tempo di simulazione [s]
108
109 global delta_e...
110 delta_s...
111 delta_T
112
113 delta_e = @(t) interp1([0, t_fin],[delta_e0_rad, delta_e0_rad],t,'linear');
114 delta_s = @(t) interp1([0, t_fin],[delta_s0_rad, delta_s0_rad],t,'linear');
115 delta_T = @(t) interp1(...
116 [0, 2, 2.5, t_fin],...

```

```

117 [delta_T0, delta_T0, delta_T0 + delta_T_excursion, delta_T0 + delta_T_excursion],t,'linear');
118
119 for i = 1:NVel
120 state0 = [vVel0(i),alpha0_rad(i),q0,xEG_0,zEG_0,theta0_rad(i)];
121
122 % Assegnazione delle leggi temporali dei comandi di volo
123
124
125 % Integrazione del sistema di equazioni differenziali
126 options = odeset('RelTol', 1e-9,'AbsTol', 1e-9*ones(1,6));
127
128 [vTime,mState] = ode45(@eqLongDynamicStickFixed,[0 t_fin],state0,options);
129
130 mState_dot = zeros(length(vTime),6);
131 for j = 1:length(vTime)
132
133 mState_dot(j,:) = eqLongDynamicStickFixed(vTime(j),mState(j,:));
134
135 end
136
137 %% figura 2
138 vVel = mState(:,1);
139 if i==1
140 vAlpha_rad = mState(:,2);
141 elseif i==2
142 vAlpha_rad = mState(:,2);
143 elseif i==3
144 vAlpha_rad = mState(:,2);
145 end
146
147 vAlpha_deg = convang(vAlpha_rad,'rad','deg');
148 v_u = vVel.*cos(vAlpha_rad - myAC.mu_x);
149 v_w = vVel.*sin(vAlpha_rad - myAC.mu_x);
150 v_q = convangvel(mState(:,3),'rad/s','deg/s');
151 vAlphaB_rad = vAlpha_rad - myAC.mu_x;
152 vAlphaB_deg = convang(vAlphaB_rad,'rad','deg');
153 vTheta_rad = mState(:,6);
154 vTheta_deg = convang(vTheta_rad,'rad','deg');
155 vGamma_deg = vTheta_deg - vAlphaB_deg;
156 vXe = mState(:,4);
157 vZe = mState(:,5);
158
159 vDelta_V = vVel - vVel0(i);
160 vDelta_alpha = vAlpha_deg - alpha0_deg(i);
161 vDelta_q = v_q - convangvel(q0,'rad/s','deg/s');
162 vDelta_h = -(vZe - zEG_0);
163
164 figure(2)
165 h(1) = subplot(3,2,1);
166 plot(vTime,vXe,'-','LineWidth',1.5); hold on;
167 grid on
168 set(gca,'xtick',0:20:120);
169 xlim([0 t_fin])
170 ylim([0 40000])
171 ylabel('$x_{EG} (m)$','interpreter','latex','fontsize',11)
172 h(2) = subplot(3,2,2);
173 plot(vTime,vDelta_h,'-','LineWidth',1.5); hold on;
174 grid on
175 set(gca,'xtick',0:20:120);
176 xlim([0 t_fin])
177 ylim([-1000 10000])
178 ylabel('$\Delta h (m)$','interpreter','latex','fontsize',11)
179 h(3) = subplot(3,2,3);
180 plot(vTime,vDelta_V,'-','LineWidth',1.5); hold on;
181 grid on
182 xlim([0 t_fin])
183 ylim([-30 215])
184 ylabel('$\Delta V (m/s)$','interpreter','latex','fontsize',11)
185 h(4) = subplot(3,2,4);
186 plot(vTime,vDelta_alpha,'-','LineWidth',1.5); hold on;
187 grid on
188 xlim([0 120])
189 ylim([-2 0.3])
190 ylabel('$\Delta \alpha (deg)$','interpreter','latex','fontsize',11)
191 h(5) = subplot(3,2,5);
192
193 plot(vTime,vDelta_q,'-','LineWidth',1.5); hold on;
194 grid on
195 xlim([0 120])
196 xlabel('$t (s)$','interpreter','latex','fontsize',11);
197 ylim([-2 2])
198 ylabel('$\Delta q (deg/s)$','interpreter','latex','fontsize',11)
199 lgd = legend('Velocità 1','Velocità 2','Velocità 3');
200 lgd.Interpreter = 'latex';
201 h(6) = subplot(3,2,6);
202 plot(vTime,vTheta_deg,'--','color',[0, 0.4470, 0.7410],'LineWidth',1.5); hold on;
203 hold on;
204 plot(vTime,vGamma_deg,':', 'color',[0.4660, 0.6740, 0.1880],'LineWidth',1.5); hold on;
205 grid on

```

```

206 lgd = legend('$\theta(t)$','$\gamma(t)$');
207 lgd.Interpreter = 'latex';
208 lgd.FontSize = 11;
209 xlim([5 120])
210 xlabel('$t (s)$','interpreter','latex','fontsize',11);
211 ylim([-15 25])
212 ylabel('$(deg)$','interpreter','latex','fontsize',11)
213
214 %% figura 4
215 vGamma_rad = vTheta_rad - vAlpha_rad - myAC.mu_x;
216 vVel_dot = mState_dot(:,1);
217 vAlpha_dot_rad = mState_dot(:,2);
218 vTheta_dot_rad = mState_dot(:,6);
219 vGamma_dot_rad = vTheta_dot_rad - vAlpha_dot_rad;
220 v_q_dot = mState_dot(:,3);
221 v_fxa = -(sin(vGamma_rad) + (vVel_dot/g));
222 v_fza = cos(vGamma_rad) + (vVel.*vGamma_dot_rad/g);
223
224 figure(4)
225 h(1) = subplot(3,2,1);
226 plot(vTime,convang(v_q_dot,'rad','deg'),'-.','LineWidth',1.5); hold on;
227 grid on
228 xlim([0 10])
229 xlabel('$t (s)$','interpreter','latex','fontsize',11);
230 ylabel('$\dot{q} (deg/s^2)$','interpreter','latex','fontsize',11);
231 ylim([-0.15 0.15])
232 h(2) = subplot(3,2,2);
233 plot(vTime,convang(v_q_dot,'rad','deg'),'-.','LineWidth',1.5); hold on;
234 grid on
235 xlim([10 120])
236 xlabel('$t (s)$','interpreter','latex','fontsize',11);
237 ylabel('$\dot{\dot{q}} (deg/s^2)$','interpreter','latex','fontsize',11);
238 ylim([-0.05 0.08])
239 h(3) = subplot(3,2,[3,4]);
240 plot(vTime,v_fza,'-.','LineWidth',1.5); hold on;
241 grid on
242 xlim([0 120])
243 ylim([0 2])
244 ylabel('$f_{z,a}$','interpreter','latex','fontsize',11);
245 h(4) = subplot(3,2,[5,6]);
246 plot(vTime,v_fxa,'-.','LineWidth',1.5); hold on;
247 grid on
248 xlim([0 120])
249 xlabel('$t (s)$','interpreter','latex','fontsize',11);
250 ylim([-0.6 0.2])
251 ylabel('$f_{x,a}$','interpreter','latex','fontsize',11);
252 lgd = legend('Velocità 1','Velocità 2','Velocità 3');
253 lgd.Interpreter = 'latex';
254
255 end
256 %% Grafica
257 %Diagrammi dei comandi di volo (figura 1)
258 figure("Name","Diagrammi dei comandi di volo","NumberTitle","off");
259 subplot 311
260 plot(vTime,convang(delta_e(vTime),'rad','deg'),'b-.','LineWidth',1.5);
261 grid on
262 xlim([0 20])
263 ylim([-8 0])
264 ylabel('$\delta_e(deg)$','interpreter','latex','fontsize',11);
265 subplot 312
266 plot(vTime,convang(delta_s(vTime),'rad','deg'),'b-.','LineWidth',1.5);
267 grid on
268 xlim([0 20])
269 ylim([-1 1])
270 ylabel('$\delta_s(deg)$','interpreter','latex','fontsize',11);
271 subplot 313
272 plot(vTime,delta_T(vTime),'b-.','LineWidth',1.5);
273 grid on
274 xlim([0 20])
275 xlabel('$t (s)$','interpreter','latex','fontsize',11);
276 ylim([0 1.1])
277 ylabel('$\delta_T$','interpreter','latex','fontsize',11);
278 diary off
279 end

```

Le funzioni presenti all'interno del precedente listato sono quelle successivamente mostrate.

costLongEquilibriumStaticStickFixed

```

1 function [f] = costLongEquilibriumStaticStickFixed(x)
2
3 % Dichiarazione delle variabili globali
4 global g ...
5     zEG_0 V0 q0 gamma0 ...
6     rho0 ...
7     myAC
8
9 mu_rel = (myAC.W/g)/(rho0*myAC.S*myAC.b);
10

```

```

11 % Assegnazione delle componenti del design vector
12 alpha = x(1);
13 delta_e = x(2);
14 delta_s = x(3);
15 delta_T = x(4);
16
17 F1 = (delta_T*myAC.T/myAC.W)*cos(alpha - myAC.mu_x + myAC.mu_T) - ...
18 sin(gamma0) - ...
19 ((rho0*V0^2)/(2*(myAC.W/myAC.S)))*...
20 (myAC.CD_0 + myAC.K*((myAC.CL_alpha*alpha + myAC.CL_delta_e*delta_e + ...
21 myAC.CL_delta_s*delta_s)^myAC.m));
22
23 F2 = q0*(1 - ((myAC.mac/myAC.b)/(4*mu_rel))*myAC.CL_q) - ...
24 ((delta_T*myAC.T/myAC.W)*(g/V0))*sin(alpha - myAC.mu_x + myAC.mu_T) + ...
25 (g/V0)*cos(gamma0) - ...
26 ((rho0*V0^2)/(2*(myAC.W/myAC.S)))*(g/V0)*...
27 (myAC.CL_alpha*alpha + myAC.CL_delta_e*delta_e + myAC.CL_delta_s*delta_s);
28
29 F3 = (myAC.Cm_T_0 + myAC.Cm_T_alpha*alpha)*delta_T + ...
30 myAC.Cm_0 + myAC.Cm_alpha*alpha + ...
31 myAC.Cm_delta_e*delta_e + ...
32 myAC.Cm_delta_s*delta_s + ...
33 (myAC.mac/(2*V0))*myAC.Cm_q*q0;
34
35 % Funzione obiettivo
36 f = F1*F1 + F2*F2 + F3*F3;
37
38 end

```

costLongEquilibriumStaticStickFixedVel

```

1 function [f] = costLongEquilibriumStaticStickFixedVel(x,V0)
2
3 % Dichiarazione delle variabili globali
4 global g ... % gravity acceleration
5 zEG_0 q0 gamma0 ... % initial conditions
6 rho0 ... % air density at z0
7 myAC % the aircraft object, allocated here
8
9 mu_rel = (myAC.W/g)/(rho0*myAC.S*myAC.b);
10
11 % Assegnazione delle componenti del design vector
12 alpha = x(1);
13 delta_e = x(2);
14 delta_s = x(3);
15 delta_T = x(4);
16
17 F1 = (delta_T*myAC.T/myAC.W)*cos(alpha - myAC.mu_x + myAC.mu_T) - ...
18 sin(gamma0) - ...
19 ((rho0*V0^2)/(2*(myAC.W/myAC.S)))*...
20 (myAC.CD_0 + myAC.K*((myAC.CL_alpha*alpha + myAC.CL_delta_e*delta_e + ...
21 myAC.CL_delta_s*delta_s)^myAC.m));
22
23 F2 = q0*(1 - ((myAC.mac/myAC.b)/(4*mu_rel))*myAC.CL_q) - ...
24 ((delta_T*myAC.T/myAC.W)*(g/V0))*sin(alpha - myAC.mu_x + myAC.mu_T) + ...
25 (g/V0)*cos(gamma0) - ...
26 ((rho0*V0^2)/(2*(myAC.W/myAC.S)))*(g/V0)*...
27 (myAC.CL_alpha*alpha + myAC.CL_delta_e*delta_e + myAC.CL_delta_s*delta_s);
28
29 F3 = (myAC.Cm_T_0 + myAC.Cm_T_alpha*alpha)*delta_T + ...
30 myAC.Cm_0 + myAC.Cm_alpha*alpha + ...
31 myAC.Cm_delta_e*delta_e + ...
32 myAC.Cm_delta_s*delta_s + ...
33 (myAC.mac/(2*V0))*myAC.Cm_q*q0;
34
35 % Funzione obiettivo
36 f = F1*F1 + F2*F2 + F3*F3;
37
38 end

```

eqLongDynamicStickFixed

```

1 function [dStatedt] = eqLongDynamicStickFixed(t,state)
2
3 global g...
4     delta_e delta_s delta_T...
5     myAC
6
7 % Assegnazione delle componenti del vettore di stato
8 V = state(1);
9 alpha = state(2);
10 q = state(3);
11 xEG = state(4);
12 zEG = state(5);
13 theta = state(6);
14
15 T_over_W = myAC.T/myAC.W;
16 alphaB_plus_muT = alpha - myAC.mu_x + myAC.mu_T;
17 gamma_t = theta - alpha + myAC.mu_x;

```

```

18 rhoVsquare_over_TwoWeightSurface = density(-zEG)*V^2/(2*(myAC.W/myAC.S));
19 CL_coeff = myAC.CL_alpha*alpha + myAC.CL_delta_e*delta_e(t)+...
20     myAC.CL_delta_s*delta_s(t);
21 V_dot = g*((delta_T(t)*T_over_W)*cos(alphaB_plus_muT) - ...
22     sin(gamma_t) - (rhoVsquare_over_TwoWeightSurface)*...
23     (myAC.CD_0 + myAC.K*(CL_coeff)^myAC.m));
24
25 mu_rel = (myAC.W/g)/(density(-zEG)*myAC.S*myAC.b);
26 macOverb_over_FourMuRel_dot_CLAlphadot = (myAC.mac/myAC.b)*myAC.CL_alpha_dot/(4*mu_rel);
27 macOverb_over_FourMuRel_dot_CLq = (myAC.mac/myAC.b)*myAC.CL_q/(4*mu_rel);
28 g_over_V = g/V;
29 alpha_dot = 1/(1 + macOverb_over_FourMuRel_dot_CLAlphadot)*...
30     (q*(1 - macOverb_over_FourMuRel_dot_CLq) - ...
31     delta_T(t)*(T_over_W)*(g_over_V)*sin(alphaB_plus_muT) + ...
32     (g_over_V)*cos(gamma_t) - ...
33     (rhoVsquare_over_TwoWeightSurface*g_over_V)*CL_coeff);
34
35 V_over_radiusgirationSquared = (V/myAC.k_y)^2;
36 macOverb_over_TwoMuRel = (myAC.mac/myAC.b)/(2*mu_rel);
37 CM_T = (myAC.Cm_T_0 + myAC.Cm_T_alpha*alpha)*delta_T(t);
38 CM_coeff = myAC.Cm_0 + myAC.Cm_alpha*alpha + myAC.Cm_delta_e*delta_e(t) + ...
39     myAC.Cm_delta_s*delta_s(t) + ...
40     (myAC.mac/(2*V))*(myAC.Cm_alpha_dot*alpha_dot + myAC.Cm_q*q);
41 q_dot = (V_over_radiusgirationSquared)*(macOverb_over_TwoMuRel)*...
42     (CM_T + CM_coeff);
43
44 xEG_dot = V*cos(gamma_t);
45
46 zEG_dot = -V*sin(gamma_t);
47
48 theta_dot = q;
49
50 dStatedt = [V_dot;alpha_dot;q_dot;xEG_dot;zEG_dot;theta_dot];
51
52 end

```

myNonLinearConstraint

```

1 function [c,ceq] = myNonLinearConstraint(x)
2
3 c = []; %Assegnazione di relazioni non lineari di disuguaglianza
4 ceq = []; %Assegnazione di relazioni non lineari di uguaglianza
5
6 end

```

3.3.1 Risultati

Per i valori presentati nella Figura 3.5, i risultati sono i seguenti:

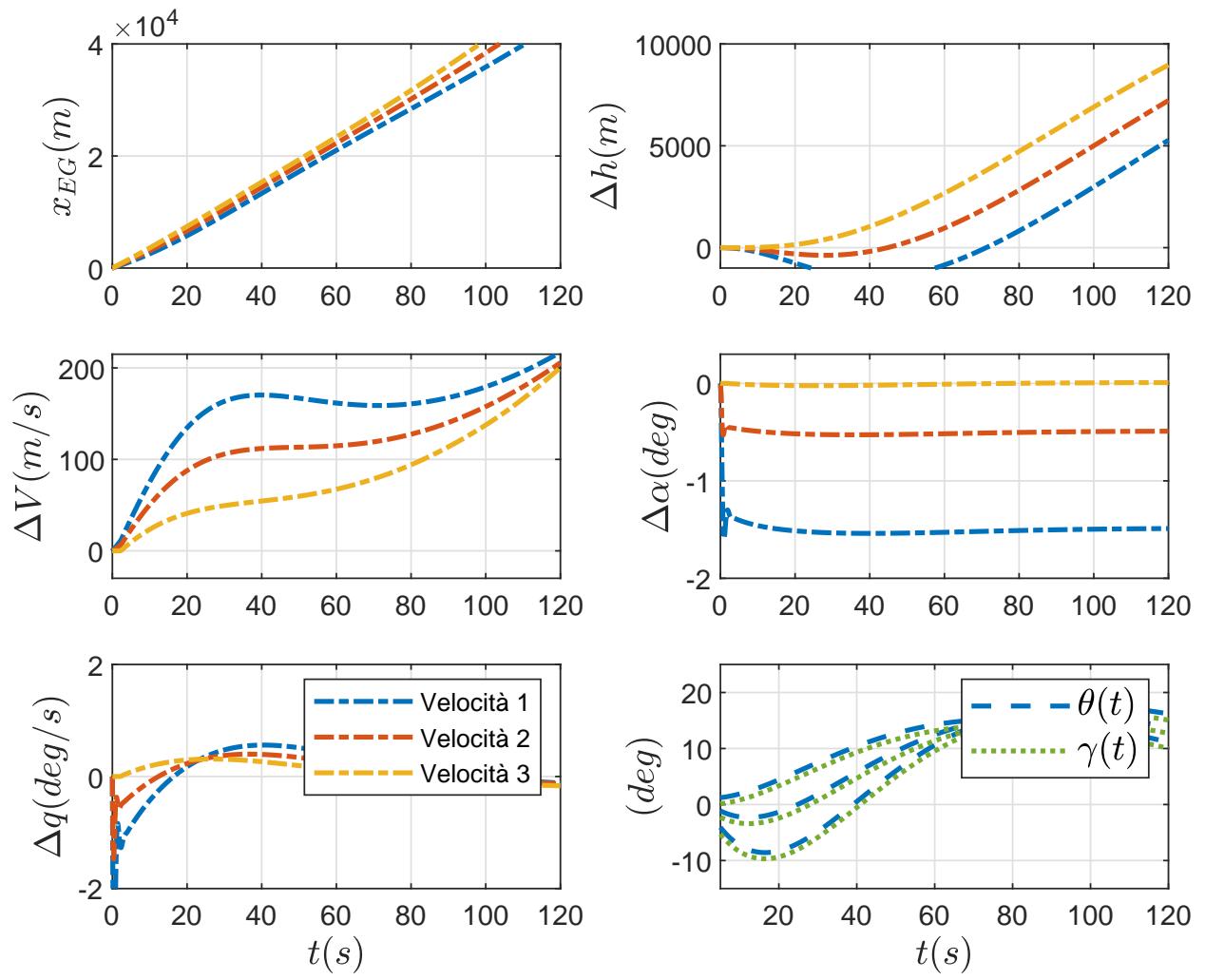


Figura 3.6: Storie temporali

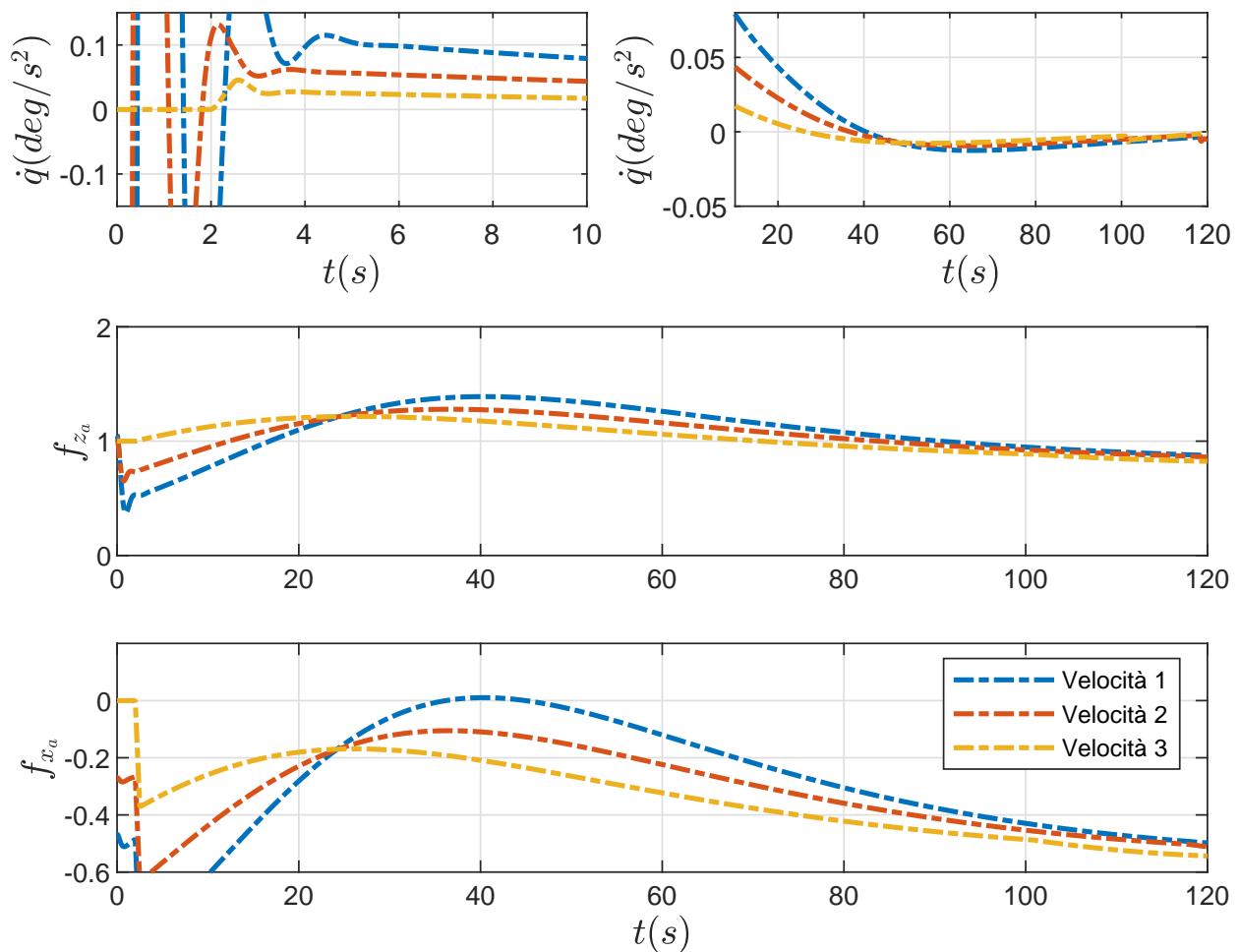


Figura 3.7: Fattori di carico

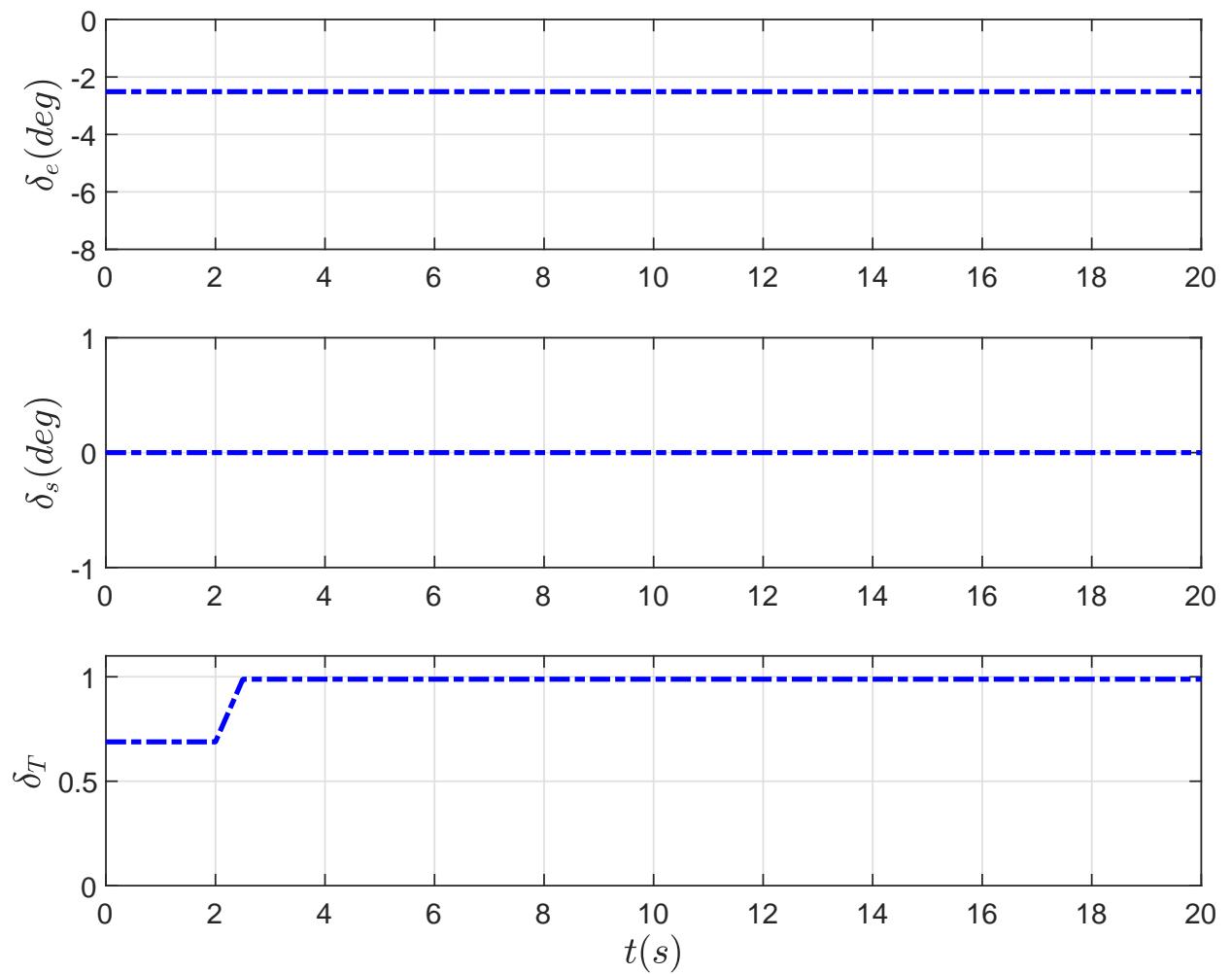


Figura 3.8: Comandi

3.4 Esercizio 3 (7.7) - Velocità di trim al variare dell'angolo iniziale dello stabilizzatore

Lo scopo di questa esercitazione è vedere gli effetti di diversi angoli iniziali dello stabilizzatore sulle condizioni di trim.

Nell'APP è possibile inserire fino a tre $\delta_{s,0}$. Tali valori sono *color coded*, come si vede nelle successive figure. I risultati vengono raggiunti grazie al codice seguente.

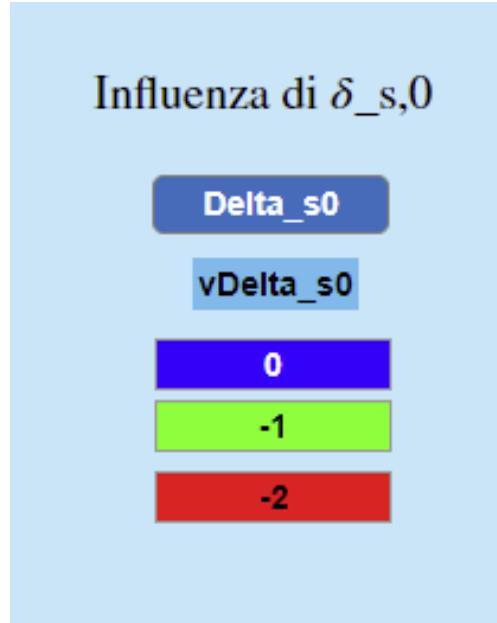


Figura 3.9: Input dell'esercizio 3

```

1 function [] = func7delta(vDelta_s0)
2
3 disp('Moto del velivolo a 3 gradi di libertà');
4 disp('Risoluzione del problema di trim ad una data altitudine e velocità di volo');
5 %load("vDelta_s0.mat")
6 vDelta_s0 = convang(vDelta_s0,'deg','rad');
7
8 %% Dichiarazione delle variabili globali
9 global g... %Accelerazione di gravità
10    zEG_0 V0 q0 gamma0 ... %Condizioni iniziali
11    rho0 ... %Densità dell'aria all'altitudine h = (-zEG_0)
12    myAC %Oggetto 'Velivolo'
13
14 %% Ricerca delle condizioni di trim
15 aircraftDataFileName = 'DSV_Aircraft_data.txt';
16
17 %Definizione dell'oggetto 'Velivolo'
18 myAC = DSVAircraft(aircraftDataFileName);
19
20 if (myAC.err == -1)
21     disp('Terminazione.')
22 else
23     disp(['File ',aircraftDataFileName,' letto correttamente.']);
24
25 % Costanti e condizioni iniziali
26 g = 9.81; %Accelerazione di gravità [m/s^2]
27 xEG_0 = 0; %[m]
28 zEG_0 = -4000; %Altitudine [m]
29 q0 = convangvel(0.000,'deg/s','rad/s'); %Velocità angolare di beccheggio
30 gamma0 = convang(0.000,'deg','rad'); %Angolo di rampa [rad]
31 [air_Temp0,sound_speed0,air_pressure0,rho0] = atmosisa(-zEG_0);
32 NVel = 20;
33 vVelo = linspace(220.0,350.0,NVel); %Velocità di volo [m/s]
34
35 %% Processo di minimizzazione della funzione di costo
36 % Valore di tentativo iniziale per il design vector
37 x0 = [0; %Valore di tentativo iniziale per alpha_0 [rad]
38        0; %Valore di tentativo iniziale per delta_e0 [rad]
39        0; %Valore di tentativo iniziale per delta_s0 [rad]
40        0.5]; %Valore di tentativo iniziale per delta_T0
41
42 % Limiti
43 lb = [convang(-15,'deg','rad'),... %Valore minimo per alpha

```

```

44         convang(-20,'deg','rad'),... %Valore minimo per delta_e
45         convang(-5,'deg','rad'),... %Valore minimo per delta_s
46         0.2]; %Valore minimo per delta_T
47     ub = [convang(15,'deg','rad'),... %Valore massimo per alpha
48         convang(13,'deg','rad'),... %Valore massimo per delta_e
49         convang(5,'deg','rad'),... %Valore massimo per delta_s
50         1.0]; %Valore massimo per delta_T
51
52 % Opzioni di ricerca del minimo
53 options = optimset('tolfun',1e-9,'Algorithm','interior-point');
54
55 %Vettore dei valori di \delta_{s0} per i quali si intende
56 %valutare il punto di minimo della funzione di costo
57 %al variare della velocità
58
59 %Inizializzazione dei vettori contenenti le variabili di output
60 alpha0_deg = zeros(NVel,length(vDelta_s0));
61 delta_e0_deg = zeros(NVel,length(vDelta_s0));
62 delta_s0_deg = zeros(NVel,length(vDelta_s0));
63 delta_T0 = zeros(NVel,length(vDelta_s0));
64
65 for j = 1:length(vDelta_s0)
66
67     Aeq = zeros(4);
68     Aeq(3,3) = 1;
69     beq = zeros(4,1);
70     beq(3,1) = vDelta_s0(j);
71
72     for i = 1:NVel
73
74         %Chiamata alla funzione 'fmincon'
75         [x,fval] = fmincon(@(x) costLongEquilibriumStaticStickFixedVel7(x,vVel0(i)),...
76             x0,...
77             [],[],Aeq,beq,...
78             lb, ub,...
79             @myNonLinearConstraint,...
80             options);
81
82         alpha0_deg(i,j) = convang(x(1),'rad','deg');
83         delta_e0_deg(i,j) = convang(x(2),'rad','deg');
84         delta_s0_deg(i,j) = convang(x(3),'rad','deg');
85         delta_T0(i,j) = x(4);
86
87     end
88
89 end
90
91 %% Grafica
92 figure(1)
93 subplot 311
94 plot(vVel0,alpha0_deg(:,3),'r--^','LineWidth',1.5,'markersize',2.5);
95 hold on;
96 plot(vVel0,alpha0_deg(:,2),'g-square','LineWidth',1.5,'markersize',2.5);
97 hold on;
98 plot(vVel0,alpha0_deg(:,1),'b:o','LineWidth',1.5,'markersize',2.5);
99 grid on
100 lgd = legend('$\delta_{s,0_3}$','$\delta_{s,0_2}$','$\delta_{s,0_1}$');
101 lgd.Interpreter = 'latex';
102 lgd.FontSize = 11;
103 xlim([vVel0(1)-10 vVel0(end)+10])
104 % ylim([0 3.5])
105 ylabel('$\alpha_0(deg)$','interpreter','latex','fontsize',11)
106 subplot 312
107 plot(vVel0,delta_e0_deg(:,3),'r--^','LineWidth',1.5,'markersize',2.5);
108 hold on;
109 plot(vVel0,delta_e0_deg(:,2),'g-square','LineWidth',1.5,'markersize',2.5);
110 hold on;
111 plot(vVel0,delta_e0_deg(:,1),'b:o','LineWidth',1.5,'markersize',2.5);
112 grid on
113 lgd = legend('$\delta_{e,0_3}$','$\delta_{e,0_2}$','$\delta_{e,0_1}$');
114 lgd.Interpreter = 'latex';
115 lgd.FontSize = 11;
116 xlim([vVel0(1)-10 vVel0(end)+10])
117 % ylim([-7 2])
118 ylabel('$\delta_{e0}(deg)$','interpreter','latex','fontsize',11)
119 subplot 313
120 plot(vVel0,delta_T0(:,3),'r--^','LineWidth',1.5,'markersize',2.5);
121 hold on;
122 plot(vVel0,delta_T0(:,2),'g-square','LineWidth',1.5,'markersize',2.5);
123 hold on;
124 plot(vVel0,delta_T0(:,1),'b:o','LineWidth',1.5,'markersize',2.5);
125 grid on
126 lgd = legend('$\delta_{T,0_3}$','$\delta_{T,0_2}$','$\delta_{T,0_1}$');
127 lgd.Interpreter = 'latex';
128 lgd.FontSize = 11;
129 xlim([vVel0(1)-10 vVel0(end)+10])
130 xlabel(['$V_0$ (m/s)'],'interpreter','latex','fontsize',11);
131 % ylim([0 1])
132 ylabel('$\delta_{T0}$','interpreter','latex','fontsize',11)

```

3.4.1 Risultati

I risultati, nel caso degli Input della Figura 3.9, sono i seguenti:

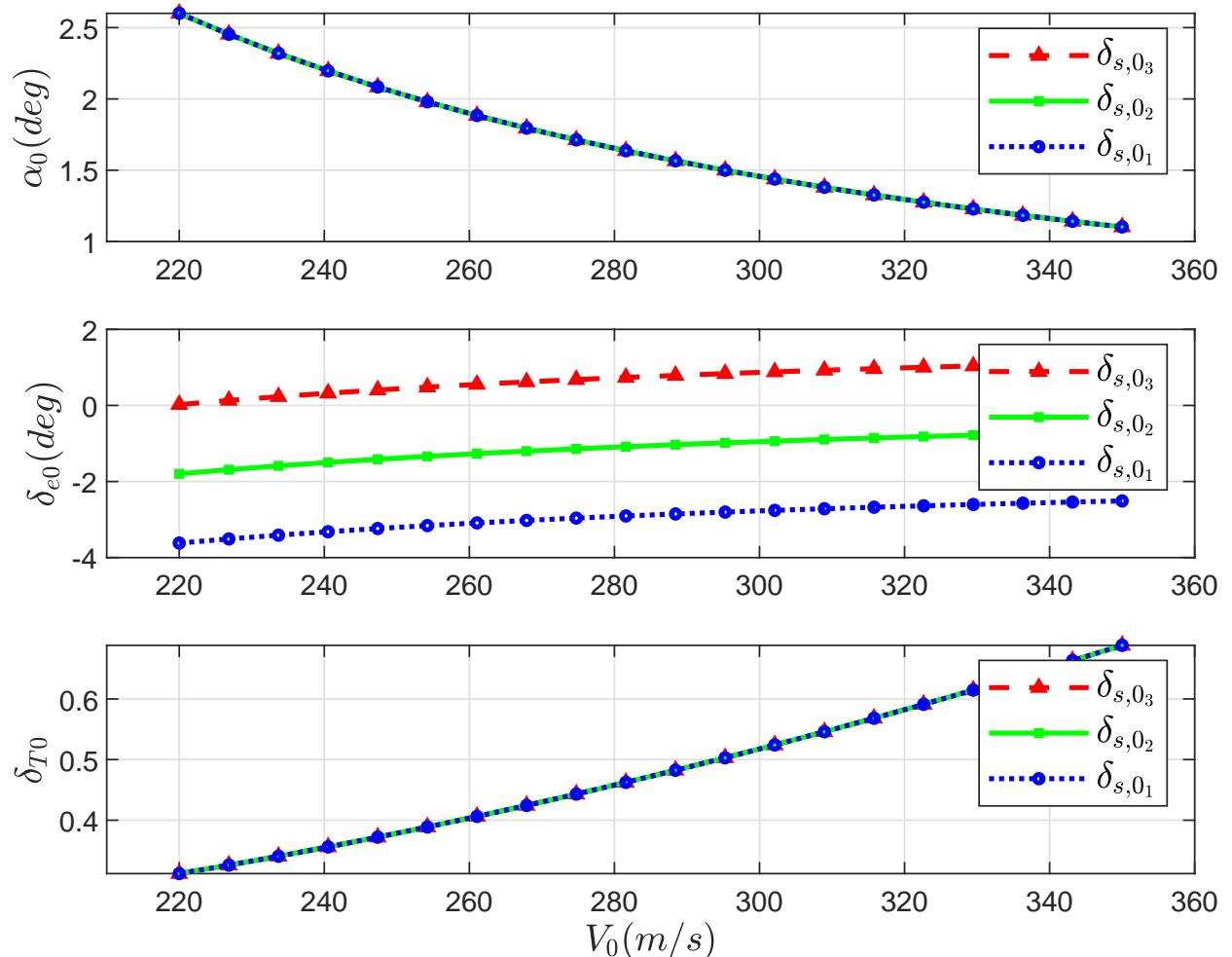


Figura 3.10: Risultati dell'esercizio 3

3.5 Esercizio 4 - Dinamica del Volo in Atmosfera Turbolenta

3.5.1 Gust

In questo esercizio aggiungiamo una raffica di vento ad un istante della simulazione.

```
1 clear all; clc; close all;
2 diary('ex7_gust2.txt')
3 diary on
4 % caso=1    delta e cost
5 % caso=2    delta e variabile
6 caso=1;
7
8 %% 3 dof stick fixed flight simulation, initial values problem resolution
9 disp(['Aircraft motion, 3DoF.']);
10 disp(['Solving the problem of trim at given altitude and flight speed.']);
11
12
13 %% declarations
14 global ...
15 t_m t_fm t_f ...
16 t0 t1 t2 t3 ...
17 g ...          % gravity acceleration
18 z_0 V_0 q_0 gamma_0 ... % initial conditions
19 rho_0 ...       % air density at z_0
20 myAC ...        % aircraft object
21 X ...
22 Y ...
23 Z ...
24 u_wind ...
25 v_wind ...
26 w_wind ...
27
28
29
30 %% Graphics Initial Settings
31 set(groot, 'defaultAxesTickLabelInterpreter','latex');
32 set(groot, 'defaultLegendInterpreter','latex');
33
34 %% populate aircraft data
35 aircraftDataFileName = 'DSV_Aircraft_Data.txt';
36
37
38 %% aircraft object
39 myAC = DSVAircraft(aircraftDataFileName);
40
41 if (myAC.err == -1)
42 disp(['... terminating.']);
43 else
44 disp(['File ', aircraftDataFileName, ' correctly read.']);
45
46
47 %% initial conditions
48 z_0      = -4000;           % above sea level altitude (m)
49 V_0      = 200;             % flight speed (m/s)
50 q_0      = 0;               % pitching angular speed (rad/s)
51 gamma_0 = 0;               % path climb angle (rad)
52
53
54 %% ISA model
55 [air_Temp_0, sound_speed_0, air_pressure_0, rho_0] = atmosisa(-z_0);
56
57
58 %% gravity acceleration
59 g = 9.81;    % (m/s^2)
60
61
62 %% initial guess for the design vector (first attempt)
63 x0 = [
64     0; ... % alpha_0
65     0; ... % delta_e_0
66     0; ... % delta_s_0
67     0.5 ... % delta_T_0
68 ];
69
70
71 %% cost function minimization
72
73 % Aeq (linear constraint)
74 Aeq      = zeros(4,4);
75 ceq      = zeros(4,1);
76 Aeq(3,3) = 1;              % pick delta_s
77 delta_s_0 = convang(0,'deg','rad');
78 ceq(3,1) = delta_s_0;      % keep delta_s fix
79
80 % lower (lb) and upper (ub) boundaries
81 lb = [convang(-15,'deg','rad'), ... % minimum alpha
82       convang(-20,'deg','rad'), ... % minimum elevator deflection
```

```

83     convang(-5,'deg','rad'), ... % minimum stabilizer incidence
84     0.2 ... % minimum thrust fraction
85 ];
86
87 ub = [convang( 15,'deg','rad'), ... % maximum alpha
88     convang( 13,'deg','rad'), ... % maximum elevator deflection
89     convang(  2,'deg','rad'), ... % maximum stabilizer incidence
90     1.0 ... % maximum thrust fraction
91 ];
92
93 options = optimset( ...
94     'tolfun',1e-4, ... % tolerance value
95     'Algorithm','interior-point' ... % algorithm type
96 );
97
98 [x,fval] = fmincon(@costLongEquilibriumStaticStickFixed2, ...
99     x0, ...
100    [], ... % A, A*x<=b
101    [], ... % b
102    Aeq, ... % Aeq, Aeq*x=beq
103    ceq, ... % beq
104    lb,ub, ...
105    @myNonLinearConstraint, ...
106    options ...
107 );
108
109 alpha_0_deg = convang(x(1),'rad','deg');
110 delta_e_0_deg = convang(x(2),'rad','deg');
111 delta_s_0_deg = convang(x(3),'rad','deg');
112 delta_T_0 = x(4);
113
114 %% data storage
115 trim_data_h_4000(1) = V_0;
116 trim_data_h_4000(2) = alpha_0_deg;
117 trim_data_h_4000(3) = delta_e_0_deg;
118 trim_data_h_4000(4) = delta_s_0_deg;
119 trim_data_h_4000(5) = delta_T_0;
120 trim_data_h_4000(6) = fval;
121
122 trim_data_h_4000 = trim_data_h_4000(:);
123
124 %% trim solution matrix
125 Trim_solution = [V_0; ...
126     convang(trim_data_h_4000(2:4),'deg','rad'); ... % alpha, delta_e, delta_s
127     trim_data_h_4000(5) ... % delta_T
128 ];
129
130
131 %% storage data in a file
132 save('trim_results_h_4000_copia.mat','trim_data_h_4000');
133
134 end
135
136
137 %% numerical solution for a longitudinal-symmetrical motion
138 %% hypothesis:
139 %% - stick fixed
140 %% - control surfaces deflections (input values) fixed to trim ones
141 %% - state vector = [u, w, q, Xeg, Zeg, theta];
142 %% - initial conditions
143 global v_t_delta_e v_t_delta_s v_t_delta_T
144
145 %% observation time
146 t_fin = 150; % final time
147 TIME = linspace(0,t_fin,100); % time vector
148 TIME = TIME(:); % time column vector
149
150
151 %% from previous problem
152 %% state vector = [u, w, q, Xeg, Zeg, theta];
153 %% state vector = [u, w, q, Xeg, Zeg, theta];
154 %% state vector = [u, w, q, Xeg, Zeg, theta];
155 alpha_0 = Trim_solution(2); % to be recalculated
156
157 %% meshgrid wind
158 x_wind = linspace(0,30000,50);
159 y_wind = linspace(-2000,2000,30);
160 z_wind = linspace(0,-10000,5);
161 [X,Y,Z] = meshgrid(x_wind,y_wind,z_wind);
162 u_wind = 0*X;
163 v_wind = 0*Y;
164 mag_w_wind = 50;
165 w_wind = 0*Z + mag_w_wind;
166 %% size u,v,w_winds = dim(y_wind), dim(x_wind), dim(z_wind)
167 w_wind(:,1:5,:,:)= 0;
168 w_wind(:,8:50,:,:)= 0;
169 w_wind((1:9),:,:)= 0;
170 w_wind((21:30),:,:)= 0;
171
```

```

172 figure(3);
173 tit = title('Visualizzazione 3D Campo di moto');
174 tit.Interpreter = 'Latex';
175 vet = quiver3(X,Y,Z,u_wind,v_wind,-w_wind);
176 vet.Color = [0, 0.4470, 0.7410];
177 vet.LineWidth = 2;
178
179
180 %% initial conditions
181 u_0 = V_0*cos(alpha_0-myAC.mu_x);
182 w_0 = V_0*sin(alpha_0-myAC.mu_x);
183
184 % state vector = [V, alpha, q, Xeg, Zeg, theta] intial conditions
185 I_C = [u_0; ...
186         w_0; ...
187         q_0; ...
188         0; ... % Xeg_0 = 0
189         z_0; ...
190         gamma_0 + alpha_0-myAC.mu_x ... % theta = gamma + alpha - mu_x
191 ];
192
193
194 %% input time stories
195 % trim conditions as initial conditions
196 delta_e_0 = Trim_solution(3);
197 delta_s_0 = Trim_solution(4);
198 delta_T_0 = Trim_solution(5);
199
200 % matrix forms : (time, values)
201 v_t_delta_e = [TIME,delta_e_0*ones(length(TIME),1)];
202 v_t_delta_s = [TIME,delta_s_0*ones(length(TIME),1)];
203 v_t_delta_T = [TIME,delta_T_0*ones(length(TIME),1)];
204
205 %length(TIME)
206 prova = zeros(100,2);
207 prova(:,1) = TIME;
208 prova(:,2) = delta_e_0*ones;
209
210 prova(80:82,2) = 1.25*delta_e_0*ones;
211 %prova(71:end,2) = delta_e_0*ones;
212
213
214
215 %delta_e_prova = [TIME,,1];
216
217
218 switch caso
219     case 1
220         v_t_delta_e = v_t_delta_e;
221     case 2
222         v_t_delta_e = prova;
223 end
224
225
226 %% numerical integration by use of Runge-Kutta method
227 [v_time, vSTAT0] = ode45(@RHS_wind, ...
228                           [0 t_fin], ...
229                           I_C ...
230                           );
231
232 V = sqrt(vSTAT0(:,1).^2 + vSTAT0(:,2).^2);
233
234 % number of iterations
235 n_iter = max(size(v_time));
236 for i=size(v_time)-8:size(v_time)
237     for j=1:6
238         vSTAT0(i,j)=vSTAT0(i-1,j);
239     end
240 end
241
242 %% aircraft visualization
243 % (x, z, theta)
244 mXYZe=[vSTAT0(:,4),vSTAT0(:,4)*0,vSTAT0(:,5)];
245 mEulerAngles=[0*vSTAT0(:,6),vSTAT0(:,6),0*vSTAT0(:,6)];
246
247 %% Setup the figure/scene for 3D visualization
248 h_fig3 = figure(3);
249 grid on;
250 hold on;
251 light('Position',[1 0 -4],'Style','local');
252 % Trick to have Ze pointing downward and correct visualization
253 set(gca,'XDir','reverse');
254 set(gca,'ZDir','reverse');
255 daspect([1 1 1]);
256
257 %% Load aircraft shape
258 shapeScaleFactor = 900.0;
259 %shape = loadAircraftMAT('shuttle.mat', shapeScaleFactor);
260 shape = loadAircraftMAT('aircraft_mig29.mat', shapeScaleFactor);

```

```

261
262 %% Settings
263 % General settings
264 options.samples = [1,40,80,120,200,250,300,400,500,600,700,800,numel(v_time)];
265 % Try [1,20,40,50,170,200]
266 options.theView = [143 15];
267 % Initial Value = [105 15];
268 % body axes settings
269 options.bodyAxes.show = true;
270 options.bodyAxes.magX = 1.5*shapeScaleFactor;
271 options.bodyAxes.magY = 2.0*shapeScaleFactor;
272 options.bodyAxes.magZ = 2.0*shapeScaleFactor;
273 options.bodyAxes.lineWidth = 2.5;
274
275 % helper lines
276 options.helperLines.show = true;
277 options.helperLines.lineStyle = ':';
278 options.helperLines.lineColor = 'k';
279 options.helperLines.lineWidth = 1.5;
280
281 % trajectory
282 options.trajectory.show = true;
283 options.trajectory.lineStyle = '-';
284 options.trajectory.lineColor = [0.6350, 0.0780, 0.1840]; % Or. Value set to : 'k'
285 options.trajectory.lineWidth = 1.5;
286
287 %% Plot body and trajectory
288 plotTrajectoryAndBodyE(h_fig3, shape, mXYZe, mEulerAngles, options);
289
290
291 %% state vector components' time histories
292
293 tht = vSTATO(:,6);
294
295 fig1=figure(1);
296 % u velocity
297 subplot(3,2,1);
298 plot(v_time,vSTATO(:,1), 'linewidth', 1.3);
299 ylabel('$u \text{ (m/s)}$', 'Interpreter', 'Latex');
300 xlabel('$t$', 'Interpreter', 'Latex');
301 %xlim([0 105]);
302 grid on; grid minor;
303
304 % w velocity
305 subplot(3,2,2);
306 plot(v_time,vSTATO(:,2), 'linewidth', 1.3);
307 ylabel('$w \text{ (m/s)}$', 'Interpreter', 'Latex');
308 xlabel('$t$', 'Interpreter', 'Latex');
309 %xlim([0 105]);
310 grid on; grid minor;
311
312 % pitching angular velocity
313 subplot(3,2,3);
314 plot(v_time,vSTATO(:,3),'linewidth', 1.3);
315 ylabel('$q \text{ (rad/s)}$', 'Interpreter', 'Latex');
316 xlabel('$t$', 'Interpreter', 'Latex');
317 %xlim([0 105]);
318 grid on; grid minor;
319
320 % Xeg
321 subplot(3,2,4);
322 plot(v_time,vSTATO(:,4), 'linewidth', 1.3);
323 ylabel('$X_{\{E,G\}(m)}$', 'Interpreter', 'Latex');
324 xlabel('$t$', 'Interpreter', 'Latex');
325 %xlim([0 105]);
326 grid on; grid minor;
327
328 % Zeg
329 subplot(3,2,5);
330 plot(v_time,-vSTATO(:,5), 'linewidth', 1.3);
331 ylabel('$h \text{ (m)}$', 'Interpreter', 'Latex');
332 xlabel('$t$', 'Interpreter', 'Latex');
333 %axis([0 105 3900 4800]);
334 grid on; grid minor;
335
336 % theta
337 subplot(3,2,6);
338 plot(v_time,tht, 'linewidth', 1.3);
339 ylabel('$\theta \text{ (rad)}$', 'Interpreter', 'Latex');
340 xlabel('$t$', 'Interpreter', 'Latex');
341 %xlim([0 105]);
342 grid on; grid minor;
343
344 %% profilo 2D della raffica ascensionale (z vs. x)
345 fig3=figure(3);
346 tiledlayout(2,1)
347 nexttile
348 x_wind_plot = [0, 2.4490e+03, 2.4490e+03, 4.2857e+03, 4.2857e+03, 30000];
349 w_wind_plot = [0, 0, mag_w_wind, mag_w_wind, 0, 0];

```

```

350 %
351 plot(x_wind_plot, w_wind_plot,'LineWidth',1.5); hold on
352 xlabel('$x~(m)$','Interpreter','Latex');
353 ylabel('$w_{wind}~(m/s)$','Interpreter','Latex'); grid on; grid minor;
354 nexttile
355 v_t_delta_e_deg=convang(v_t_delta_e(:,2),'rad','deg');
356 prova_deg=convang(prova(:,2),'rad','deg');
357
358 switch caso
359 case 1
360 plot(TIME,v_t_delta_e_deg,'-k',LineWidth=1.3);
361 case 2
362 plot(TIME,prova_deg,'-k',LineWidth=1.3);
363 end
364 hold off; grid on;
365 ylabel('$\delta_{\text{angle}}~(\text{deg})$','Interpreter','Latex');
366 xlabel('$t$','Interpreter','Latex');
367 %% AoA = angle of attack
368 % atan(w/u)
369 fig4=figure(4);
370 tiledlayout(2,2)
371 nexttile
372 plot(v_time,convang(atan2(vSTATO(:,2),vSTATO(:,1)),'rad','deg'), 'linewidth', 1.3);
373 legend('$\alpha~(t)$');
374 xlabel('$t$','Interpreter','Latex');
375 ylabel('$\alpha~(\text{deg})$','Interpreter','Latex');
376 %xlim([0 105]);
377 grid on; grid minor;
378
379
380
381 %% vertical load factor f_zA
382 V_dot = gradient(V, v_time);
383 Omega_v_V = -vSTATO(:,3).*-vSTATO(:,2)+vSTATO(:,3).*-vSTATO(:,1);
384 acc = V_dot + Omega_v_V;
385 nexttile
386 fza = -acc/9.81 + 1;
387 plot(v_time(1:18:end), fza(1:18:end), 'linewidth', 1.3);
388 xlabel('$t$','Interpreter','Latex');
389 legend('$f_z~(t)$','Interpreter','Latex');
390 xlim([0 105]);
391 grid on; grid minor;
392
393
394
395 %% lift and pitch moment coefficient (see 9.6 @agodemar)
396
397 v_q = -vSTATO(:,3);
398 de = x(2);
399 a = atan2(vSTATO(:,2),vSTATO(:,1));
400 a_dot = gradient(a, v_time);
401 mac = myAC.mac; % mean aerodynamic chord
402
403 CLO = 0.2;
404 % CLa = 4.1800; % (1/rad)
405 % CLde = 0.2870; % (1/rad)
406 % CLq = 4.720;
407 % CLa_dot = 2.270;
408 %
409 % CM0 = -0.0150;
410 % CMa = -0.836; % (1/rad)
411 % CMde = -0.5070; % (1/rad)
412 % CMq = -8.34;
413 % CMa_dot = -4.000;
414
415 CLa = myAC.CL_alpha; % (1/rad)
416 CLde = myAC.CL_delta_e; % (1/rad)
417 CLq = myAC.CL_q;
418 CLa_dot = myAC.CL_alpha_dot;
419
420 CM0 = myAC.Cm_0;
421 CMa = myAC.Cm_alpha; % (1/rad)
422 CMde = myAC.Cm_delta_e; % (1/rad)
423 CMq = myAC.Cm_q;
424 CMa_dot = myAC.Cm_alpha_dot;
425
426
427 for i = 1:length(v_time)
428
429 alpha = a(i);
430 alpha_dot = a_dot(i);
431 delta_e = de;
432 q = v_q(i);
433
434 CL_st(i) = CLO + CLa*alpha + CLde*delta_e;
435 % CL_st(i) = CL_st(i)/1.51;
436 CL_dn(i) = (CLq*q + CLa_dot*alpha_dot)*mac*0.5/V(i);
437 % CL_dn(i) = CL_dn(i)*9.5;
438 CL(i) = CL_st(i) + (CLq*q + CLa_dot*alpha_dot)*mac*0.5/V(i);

```

```

439 CM_st(i) = CMO + CMalpha + CMde*delta_e;
440 CM_dn(i) = (CMq*q + CMalpha_dot*alpha_dot)*mac*0.5/V(i);
441 CM(i) = CM_st(i) + CM_dn(i);
442
443
444
445 end
446
447
448 %% Steady and Unsteady Lift vs Reduced Frequency
449
450 L_steady = 0.5*myAC.S*rho_0*V.^2.*CL_st';
451 L_steady(958:end)=L_steady(957);
452 L_unsteady = 0.5*myAC.S*rho_0*V.^2.*(CL_st+CL_dn)';
453 L_unsteady(958:end) = L_unsteady(957);
454 rap = L_unsteady./L_steady;
455 Str = a_dot.*myAC.mac./2.*V;
456
457 nexttile
458 plot(v_time,Str,'LineWidth',1.4); grid on; grid minor;
459 xlabel('t(s)', 'Interpreter', 'Latex');
460 ylabel('Reduced Frequency of Motion', 'Interpreter', 'Latex');
461 %
462 nexttile
463 cost = 1*ones(length(v_time'),1);
464 plot(v_time,rap); grid on; grid minor;
465 hold on;
466 plot(v_time,cost,'--');
467 strn = 'Ratio of Unsteady Lift to Steady Lift vs Time';
468 tit = title(strn);
469 tit.Interpreter = 'Latex';
470 xlabel('$t(s)$', 'Interpreter', 'Latex');
471 ylabel('Ratio of Unsteady Lift to Steady Lift', 'Interpreter', 'Latex');
472 %
473
474
475 CL_tot = CL_st+CL_dn;
476
477 fig5=figure(5)
478 plot(v_time, CL_st, 'linewidth', 1.3);
479 hold on;
480 plot(v_time, CL_dn, 'linewidth', 1.3);
481 hold on;
482 plot(v_time, CL_tot,'--', 'linewidth', 1.3);
483 legend('$C_L{st}$','$C_L{din}$','$C_L$');
484 xlabel('t (s)', 'Interpreter', 'Latex');
485 ylabel('Lift Coefficient', 'Interpreter', 'Latex');
486 grid on; grid minor;
487 % create smaller axes in top right, and plot on it
488 axes('Position',[.5 .6 .1 .2])
489 box on
490 plot(v_time(114:387),CL_dn(114:387),'-', 'linewidth', 1.3);
491 xlabel('$t (s)$', 'Interpreter', 'Latex');
492 ylabel('$C_L{din}$', 'Interpreter', 'Latex');
493 % legend('$C_L{din}$');
494 grid on; grid minor;
495
496 fig2=figure(2)% plot(v_time, CM);
497 % hold on;
498 plot(v_time, CM_st, 'linewidth', 1.3);
499 hold on;
500 plot(v_time, CM_dn, 'linewidth', 1.3);
501 hold on;
502 plot(v_time, CM_st + CM_dn,'--', 'linewidth', 1.3);
503 leg = legend('$C_M{st}$','$C_M{din}$','$C_M{st}+C_M{din}$');
504 leg.Location = 'SouthEast';
505 leg.Box = 'On';
506 leg.FontSize = 10;
507 xlabel('t (s)', 'Interpreter', 'Latex');
508 ylabel('Pitch Coefficient', 'Interpreter', 'Latex');
509 xlim([0 105]);
510 grid on; grid minor;
511
512
513 diary off
514
515 saveas(fig1,"Gust1.eps",'epsc')
516 saveas(fig2,"Gust2.eps",'epsc')
517 saveas(h_fig3,"Gust3.eps",'epsc')
518 saveas(fig4,"Gust4.eps",'epsc')
519 saveas(fig5,"Gust5.eps",'epsc')

```

Viene utilizzata la seguente funzione **RHS_wind.m**

```

1 function dx = RHS_wind(t,x)
2
3 %% Declaring global variables
4 global ...
5 g ... % gravity acceleration

```

```

6      myAC ...           % the aircraft object
7      v_t_delta_T ...    % command time histories
8      v_t_delta_e ...    %
9      v_t_delta_s ...    %
10     X ...
11     Y ...
12     Z ...
13     w_wind ...
14     u_wind ...
15
16
17 %% Give the design vector components proper names
18 u = x(1);
19 w = x(2);
20 q = x(3);
21 x_G_E = x(4);
22 z_G_E = x(5);
23 theta = x(6);
24
25 % give dimensions to the return vector
26 dx = zeros(6,1);
27
28 %% Using Matlab built-in ISA model for density
29 [air_Temp, sound_speed, air_pressure, rho] = atmosisa(-z_G_E);
30
31 %% Aircraft relative density
32 mu_rel = (myAC.W/g)/(rho*myAC.S*myAC.b);
33
34 % TIME = v_t_delta_T(:, 1) = v_t_delta_e(:, 1) = v_t_delta_s(:, 1) =
35 delta_T = interp1(v_t_delta_T(:,1),v_t_delta_T(:,2),t);
36 delta_e = interp1(v_t_delta_e(:,1),v_t_delta_e(:,2),t);
37 delta_s = interp1(v_t_delta_s(:,1),v_t_delta_s(:,2),t);
38
39
40 %% wind gusts
41 % w_wind is a fcn of 3 variables (X,Y,Z) > interpolation on query pts
42 % (xGE,0,zGE) AC position
43 W_wind_E = interp3(X,Y,Z,w_wind,x_G_E,0,z_G_E,'linear');
44 U_wind_E = interp3(X,Y,Z,u_wind,x_G_E,0,z_G_E,'linear');
45
46 vett_vel_assi_body = angle2dcm(0,theta,0,'ZYX')*[U_wind_E;0;W_wind_E];
47
48 U_wind = vett_vel_assi_body(1);
49 V_wind = vett_vel_assi_body(2);
50 W_wind = vett_vel_assi_body(3);
51
52 U_relativo = U_wind - u;
53 W_relativo = W_wind - w;
54
55 V_rel = sqrt(U_relativo^2 + W_relativo^2);
56 alpha_relativo = atan2(-W_relativo,-U_relativo) + myAC.mu_x;
57 alpha_body = atan2(w,u);
58
59 % flight-path angle
60 gamma = theta - alpha_body;
61
62
63 %% thrust
64 T = delta_T*myAC.T;
65
66
67 %% lift
68 L = 0.5*rho*(V_rel^2)*myAC.S* ...
69   (myAC.CL_alpha*alpha_relativo + myAC.CL_delta_e*delta_e + myAC.CL_delta_s*delta_s);
70
71
72 %% drag
73 D = 0.5*rho*(V_rel^2)*myAC.S* ...
74   (myAC.CD_0 + myAC.K*((myAC.CL_alpha*alpha_relativo ...
75   + myAC.CL_delta_e*delta_e ...
76   + myAC.CL_delta_s*delta_s)^myAC.m));
77
78
79 %% forces component in body reference frame
80 X_force = T*cos(myAC.mu_T) - D*cos(alpha_relativo) + ...
81   L*sin(alpha_relativo) - myAC.W*sin(theta);
82
83 Z_force = T*sin(myAC.mu_T) - D*sin(alpha_relativo) - ...
84   L*cos(alpha_relativo) + myAC.W*cos(theta);
85
86
87 %% RHS eqns (see 7.10 and 7.52 @agodemar)
88 % eqns to be dynamically written in (u,w,q)
89
90 dx(1,1) = -q*w+(g/myAC.W)*X_force;
91 dx(2,1) = q*u+(g/myAC.W)*Z_force;
92 dx(4,1) = sqrt(u^2+w^2)*cos(gamma);
93 dx(5,1) = -sqrt(u^2+w^2)*(-sin(gamma));
94 dx(6,1) = q;

```

```

95 dx(3,1) = (((V_rel)^2/myAC.k_y^2)*(myAC.mac/myAC.b)/(2*mu_rel)) ...
96   *( myAC.Cm_0 + myAC.Cm_alpha*alpha_relativo ...           % Aerodynamics
97     + (myAC.mac/(2*(u^2+w^2)))*myAC.Cm_q*q ...
98     + myAC.Cm_delta_s*delta_s ...
99     + myAC.Cm_delta_e*delta_e ...
100    + (myAC.Cm_T_0 + myAC.Cm_T_alpha*alpha_relativo)*delta_T);
101
102
103
104 end

```

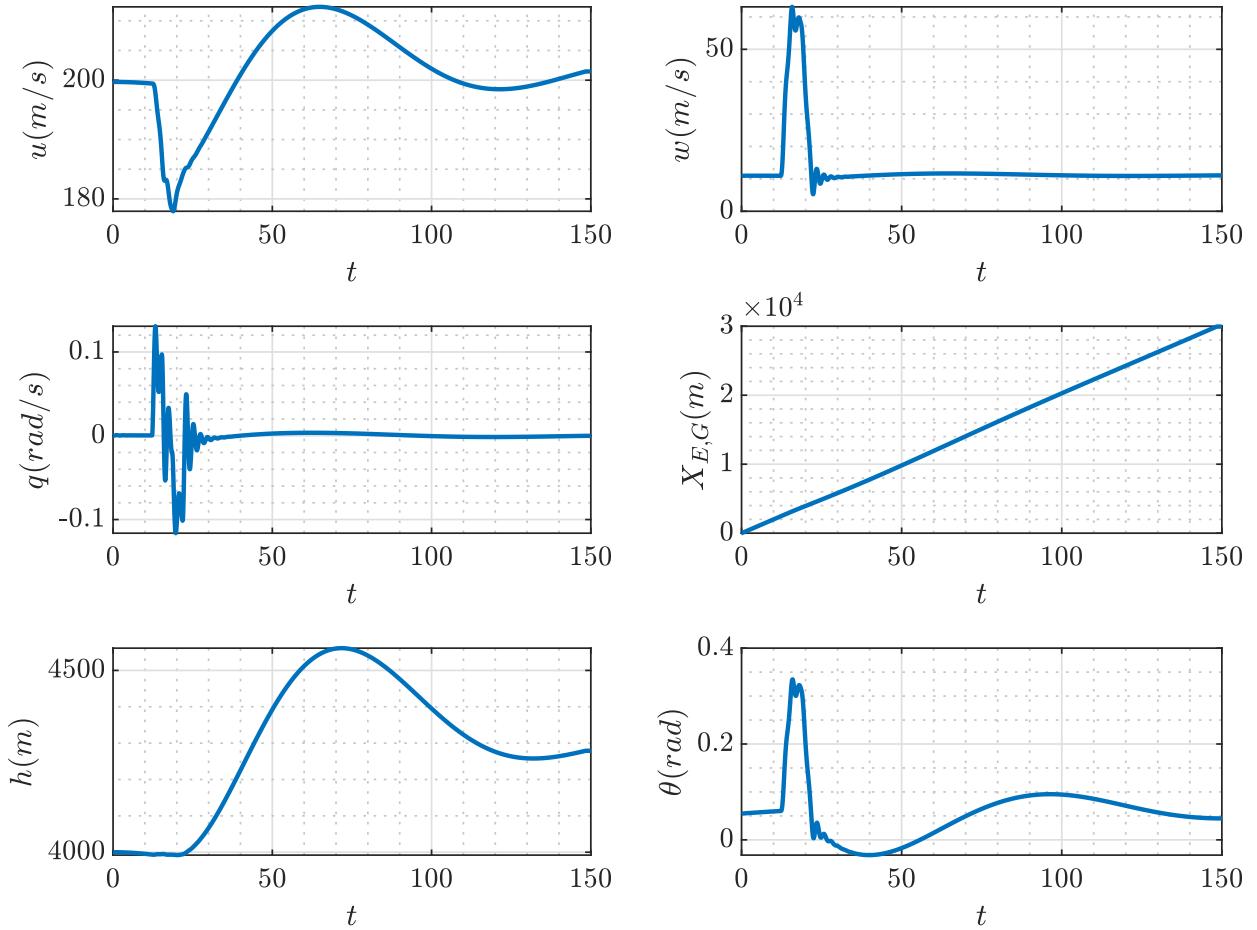


Figura 3.11: Gust - Velocità, q , coordinate del CG, θ

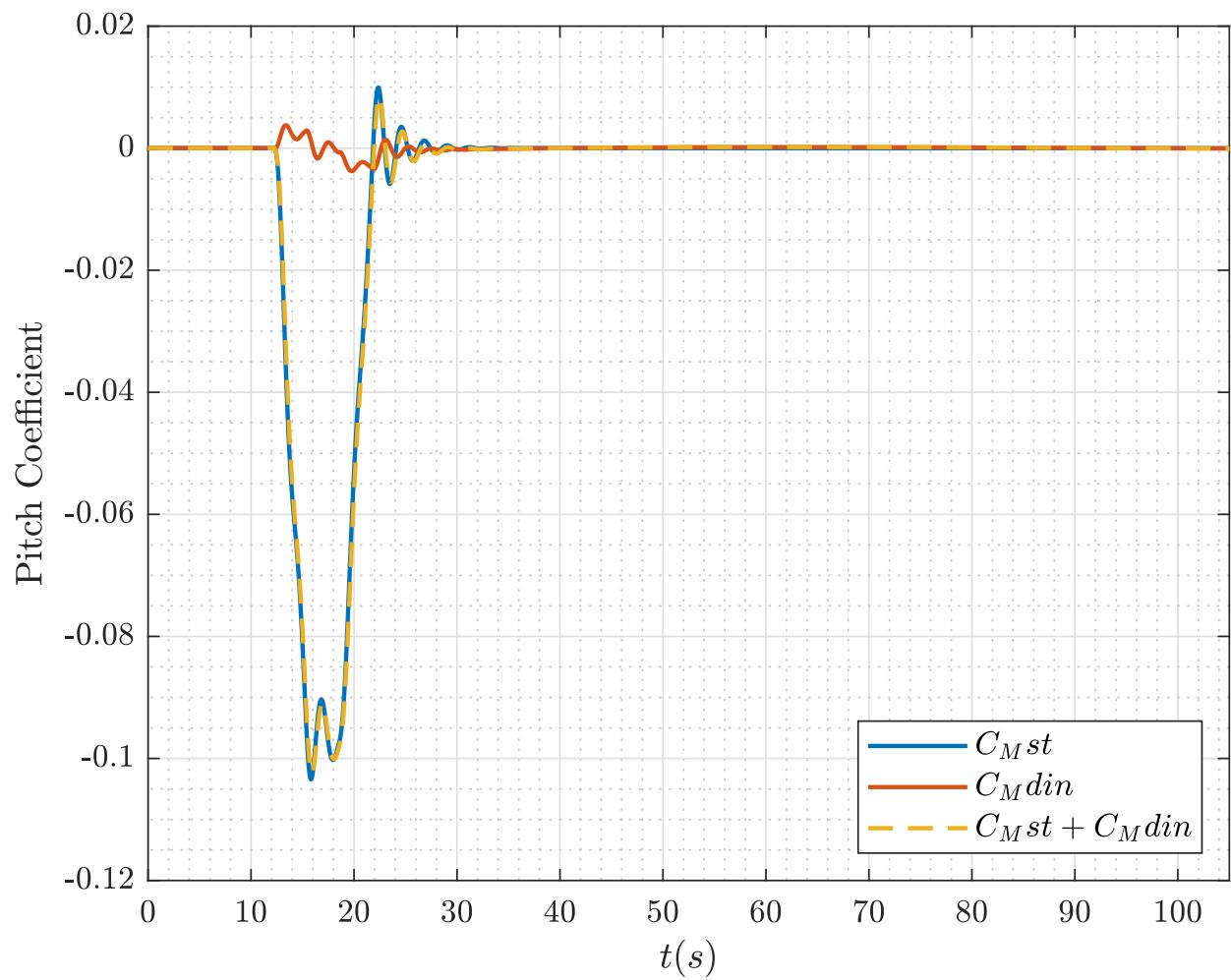


Figura 3.12: Gust - C_M stazionario e dinamico

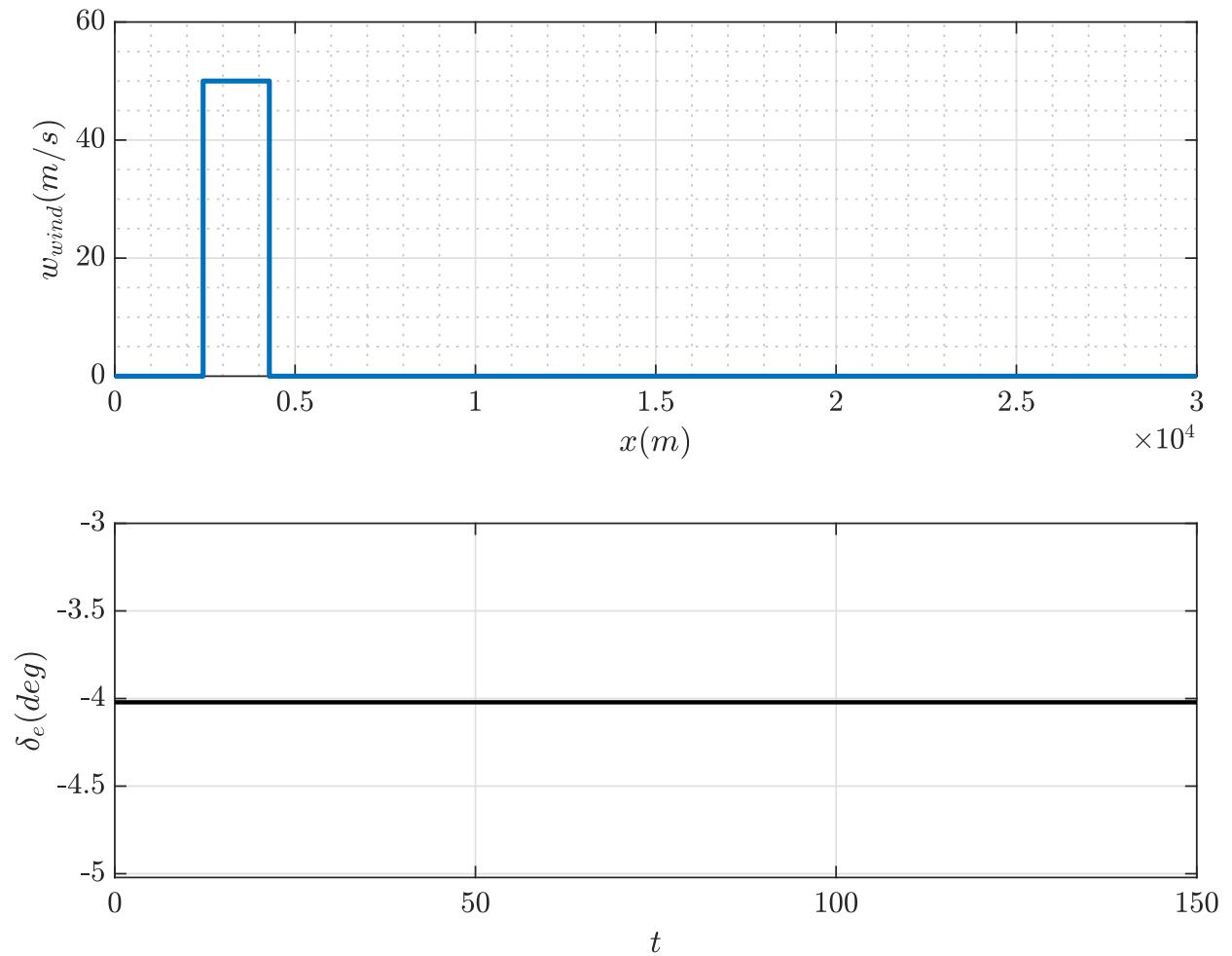


Figura 3.13: Gust - Vento e comandi del δ_e

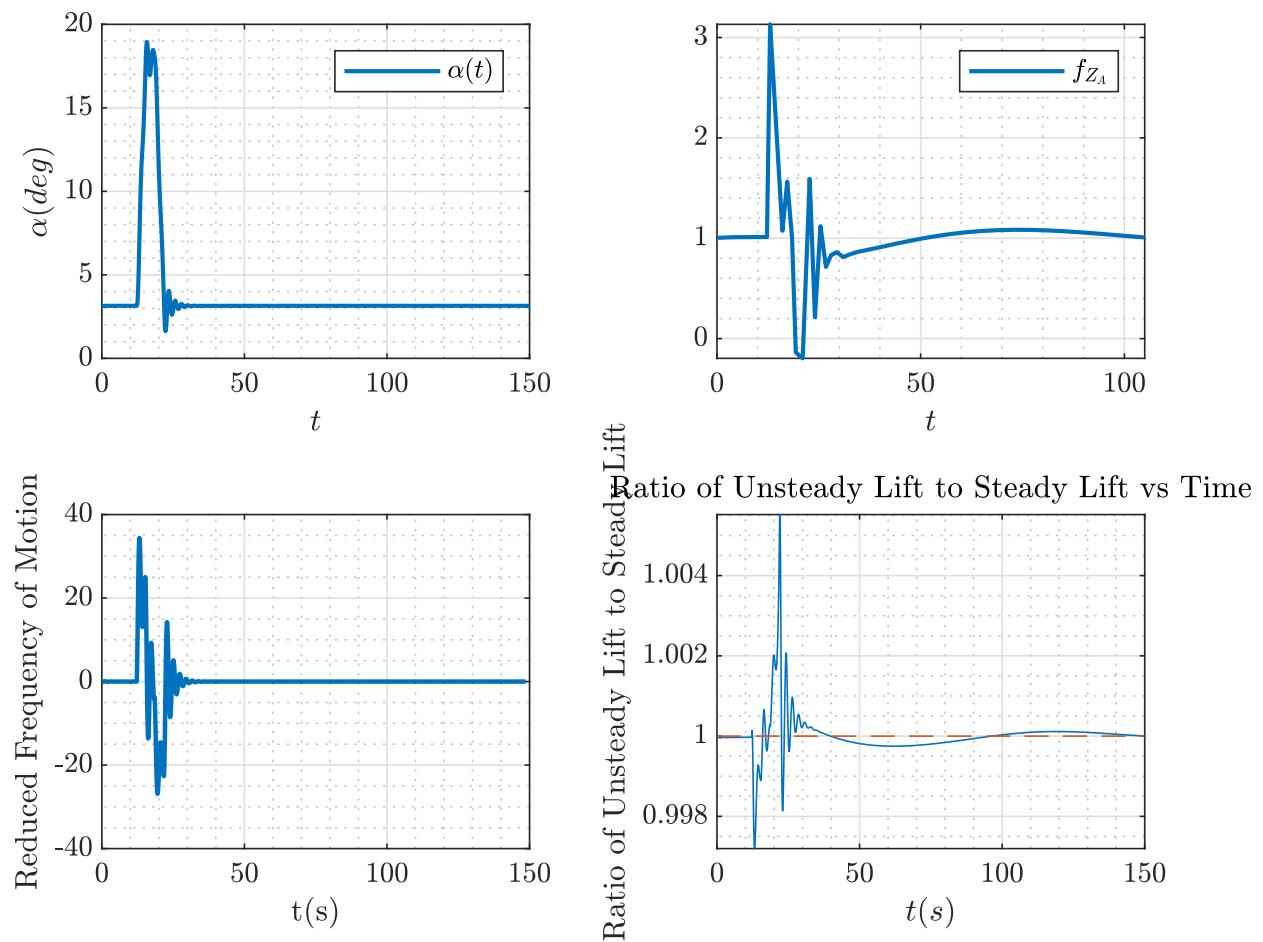


Figura 3.14: Gust - Angolo di attacco, fattore di carico, frequenza ridotta (numero di Strouhal = $\dot{\alpha}c/2V$), rapporto fra portanza instazionaria e stazionaria

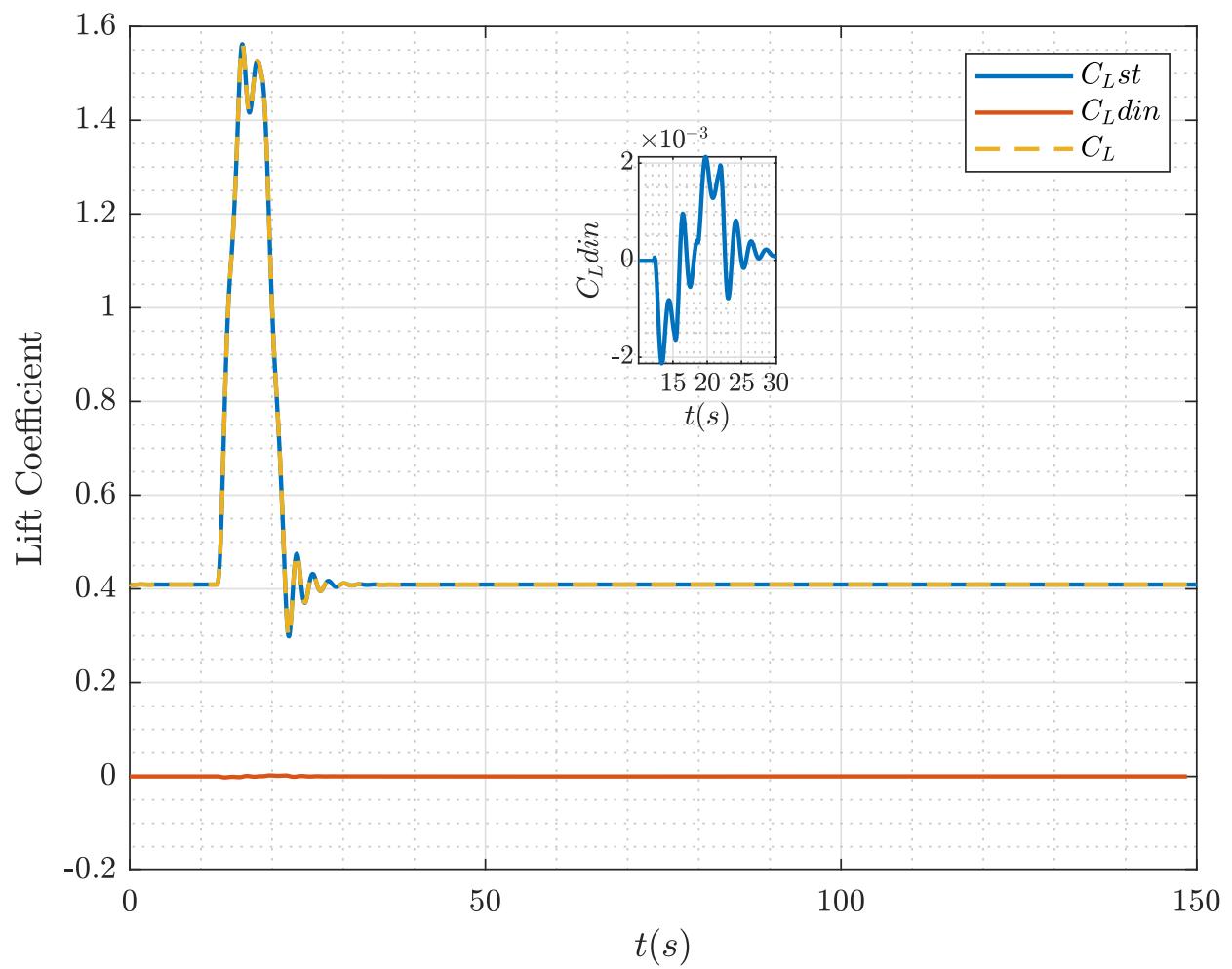


Figura 3.15: Gust - C_L stazionario e instazionario

3.5.2 Gust e δ_e

Si utilizza lo stesso codice precedente, dando in input il caso 2. In quest'ultimo viene dato un comando all'equilibratore per riportare il velivolo alla quota iniziale dopo la raffica.

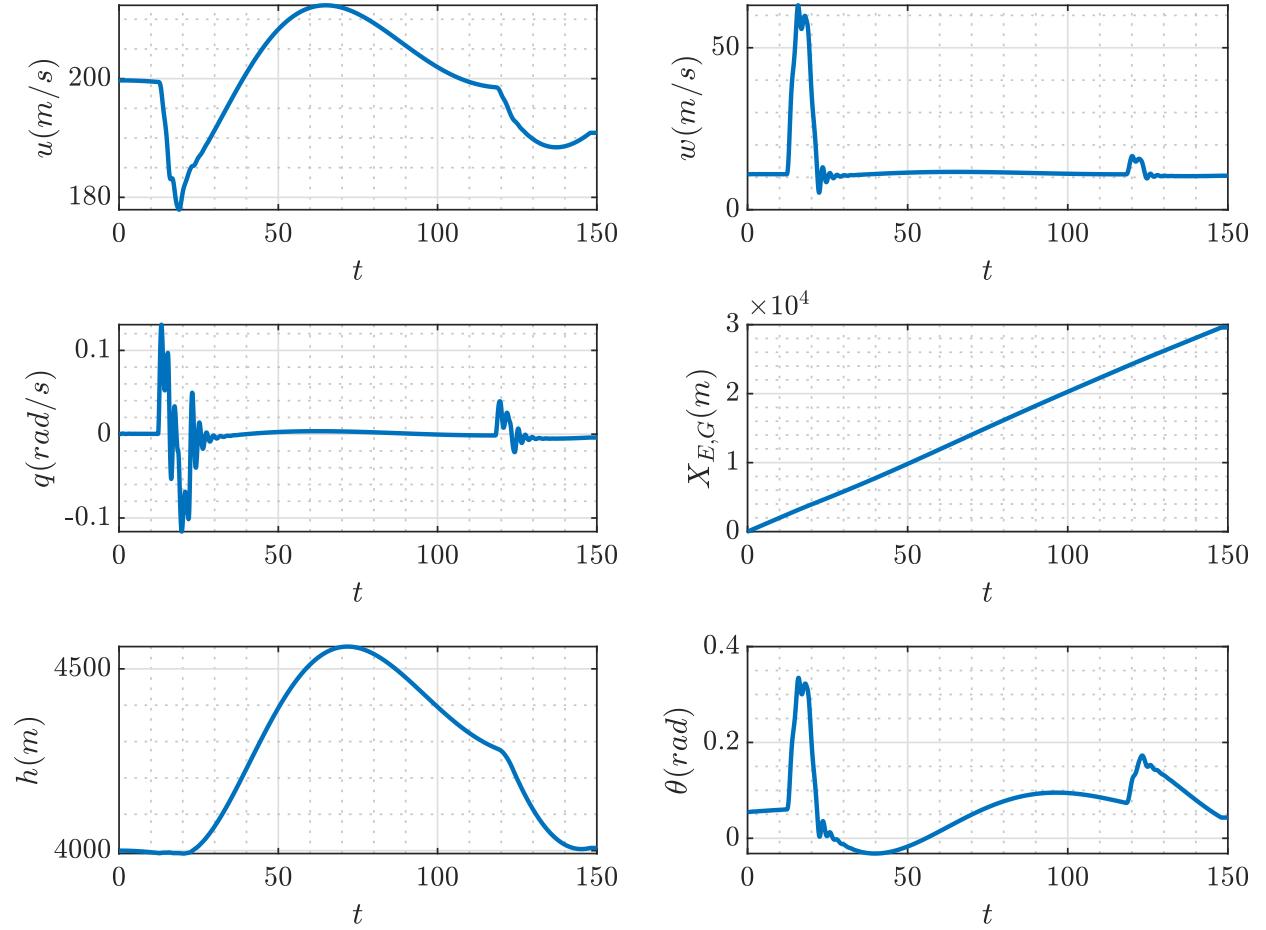


Figura 3.16: Gust e δ_e - Velocità, q, coordinate del CG, θ

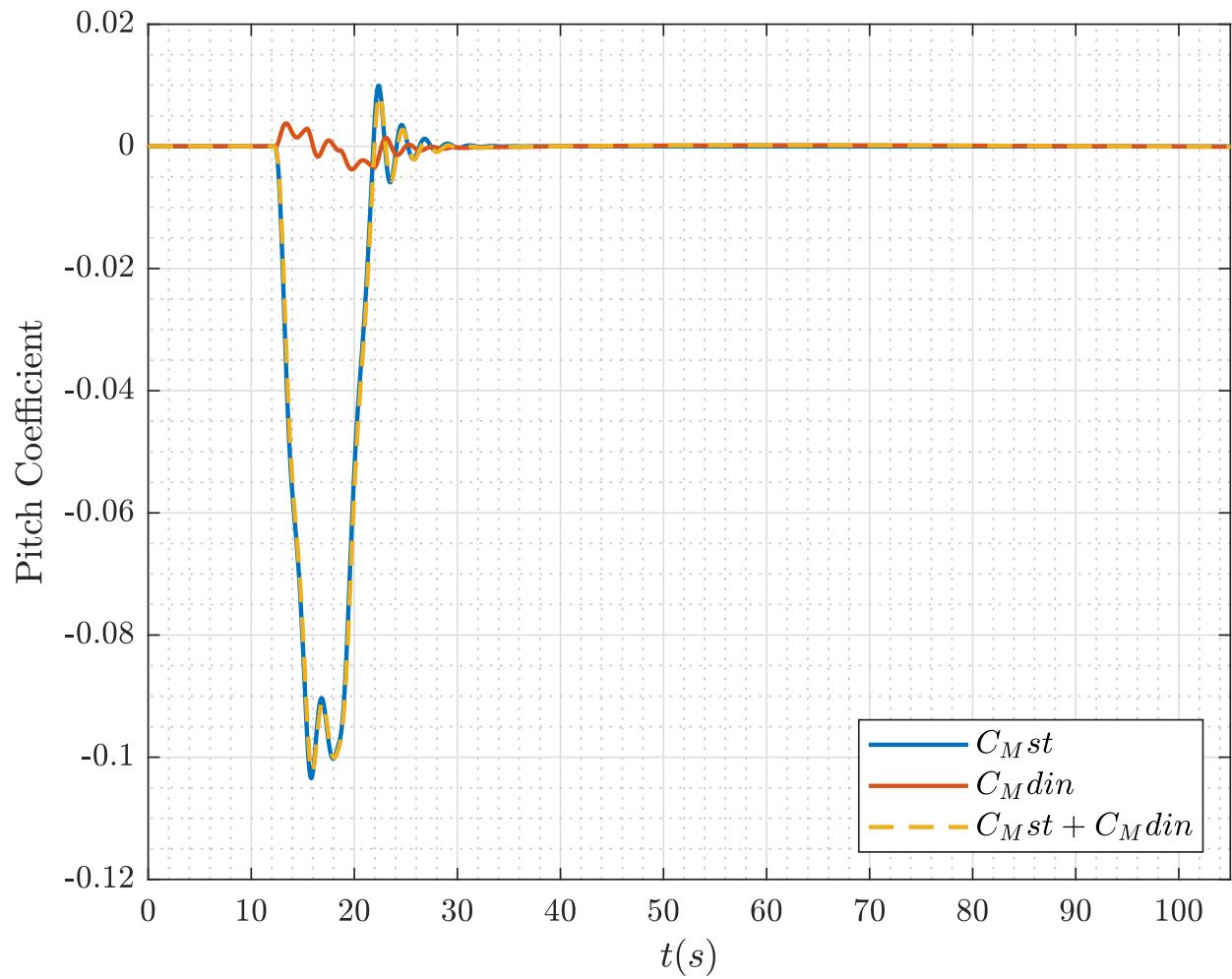


Figura 3.17: Gust e δ_e - C_M stazionario e dinamico

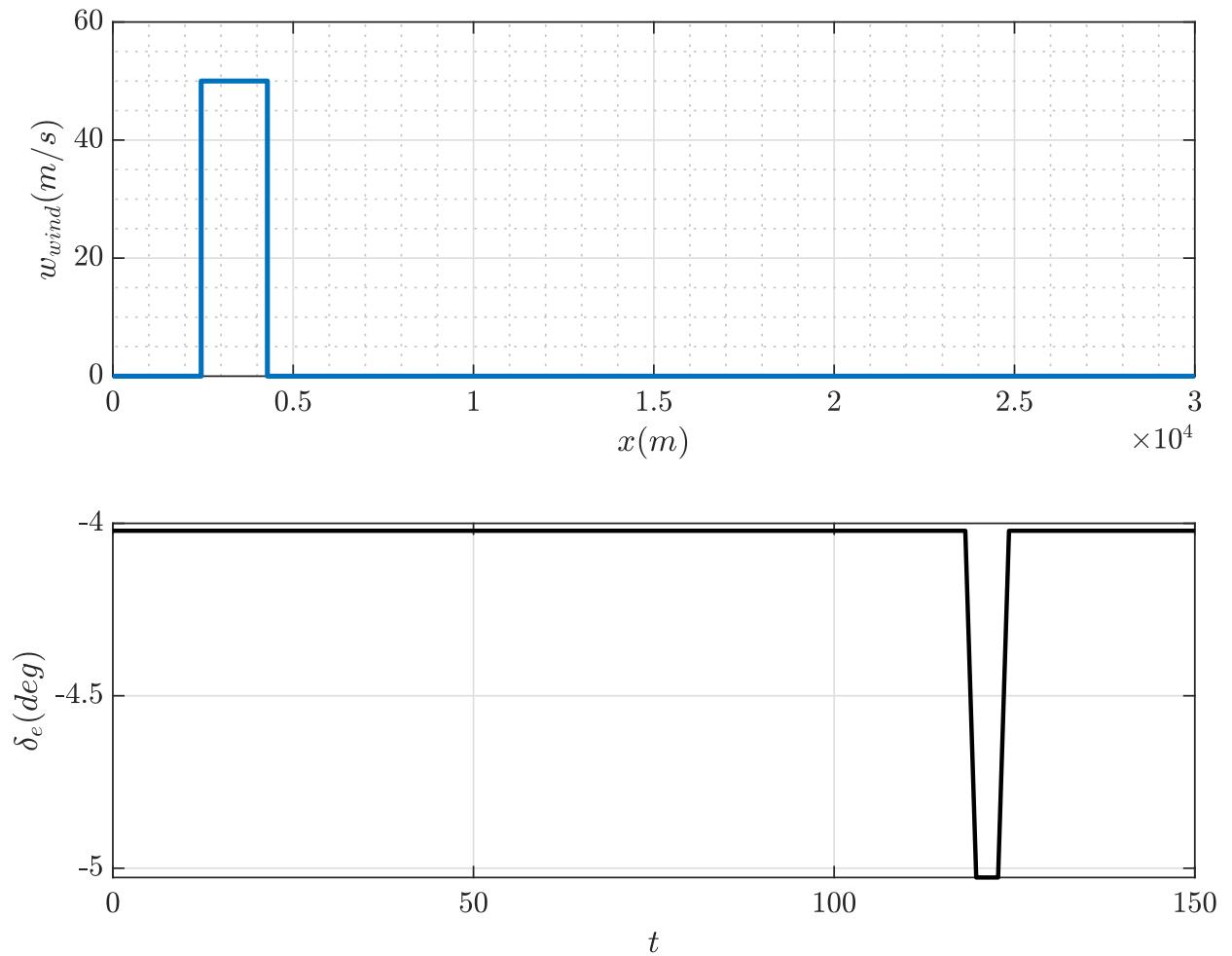


Figura 3.18: Gust e δ_e - Vento e comandi del δ_e

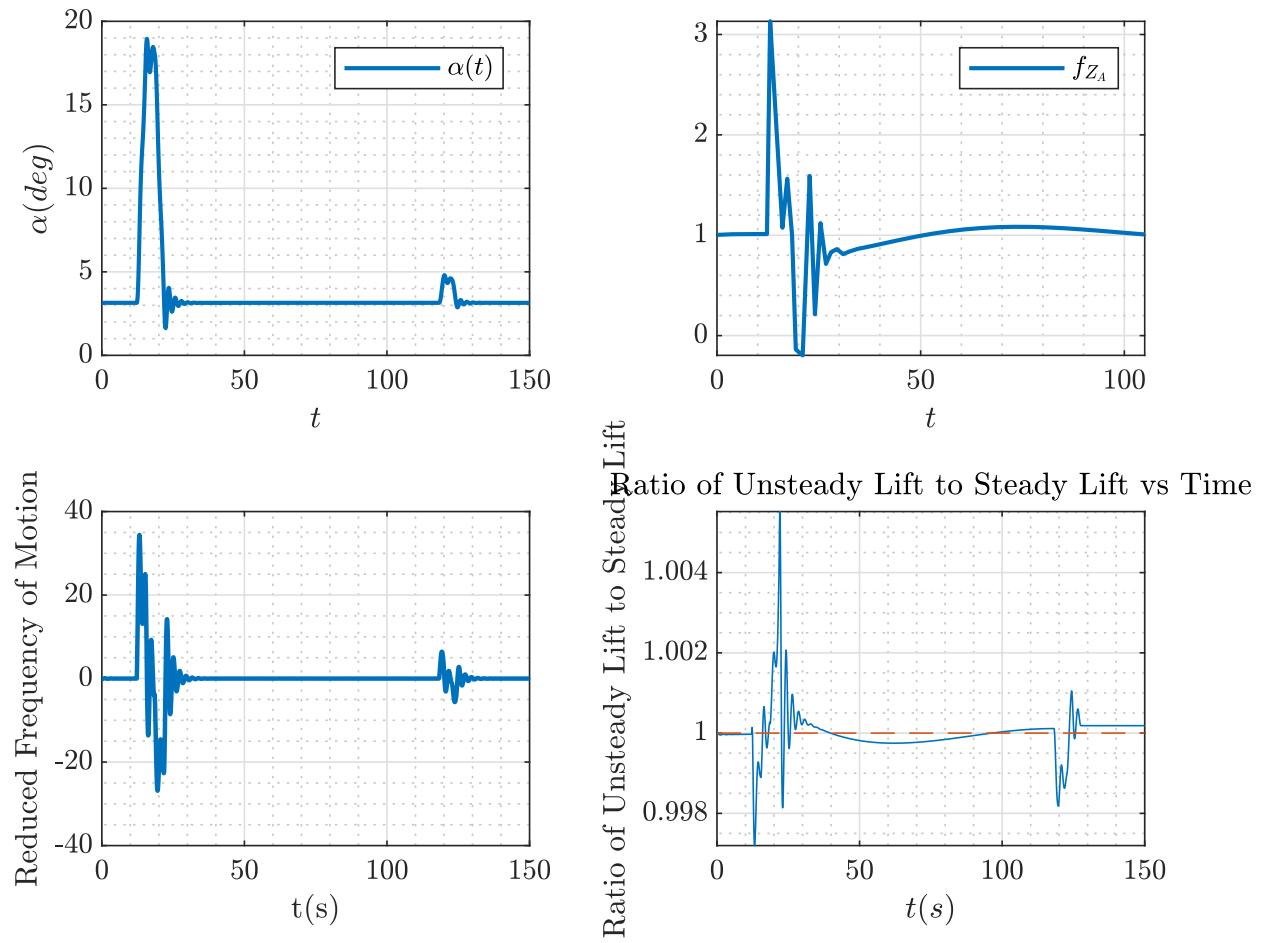


Figura 3.19: Gust e δ_e - Angolo di attacco, fattore di carico, frequenza ridotta (numero di Strouhal = $\dot{\alpha} \bar{c} / 2V$), rapporto fra portanza instazionaria e stazionaria

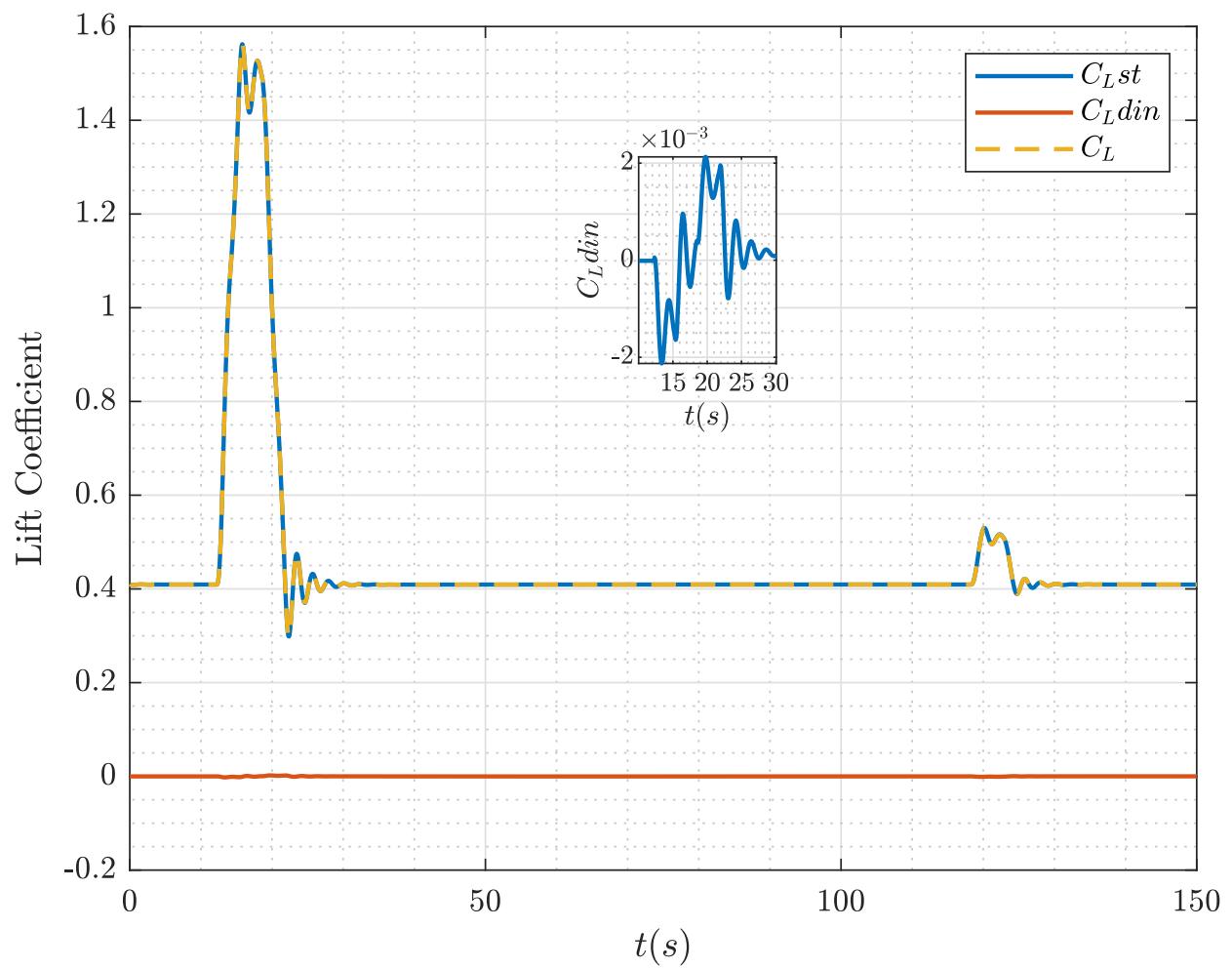


Figura 3.20: Gust e δ_e - C_L stazionario e instazionario

Capitolo 4

Equazioni del moto delle superfici di governo

4.1 Intro

Lo scopo di questa esercitazione è l'analisi del moto longitudinal-simmetrico a comandi liberi di un velivolo rigido, evoluzione del problema esaminato a comandi bloccati.

Si parte dalla seguente:

$$(\dot{K}_{xm}) + \Omega_{ym}K_z - \Omega_{zm}K_y + m(y_g a_z - z_g a_y)_n e_{CS} = \mathcal{M} \quad (4.1)$$

Che sostituendo con $(x, y, z) = (c, t, n)$, X_G=c_GCS=0, Y_G=t_GCS=e_CS, Z_G=n_GCS=0

$$(\dot{K}_{r,CS}) + (\Omega_{cs})_t(K_{r,CS})_n - (\Omega_{cs})_n(K_{r,CS})_t + m(a_c)_n e_{CS} = \mathcal{H}_{CS} \quad (4.2)$$

Da cui attraverso alcuni passaggi discende la:

$$I_c[(\dot{\omega}_{CS})_c + (\dot{\omega}_{CS})_t(\dot{\omega}_{CS})_n] + m_{CS}(a_C)_n e_{CS} = \mathcal{H}_{CS} \quad (4.3)$$

Precisiamo poi \mathcal{H}_{CS} .

$$\mathcal{H}_{CS} = \mathcal{H}_{A,CS} + m_{CS}g_n e_{CS} + \mathcal{H}_{C,CS} \quad (4.4)$$

Riscrivendo la velocità angolare rispetto al velivolo come:

$$\Omega_{CS} = \omega_{CS} + \Omega_B = \dot{\delta}_{CS}\mathbf{i}_c + p \mathbf{i}_B + q \mathbf{j}_B + r \mathbf{k}_B \quad (4.5)$$

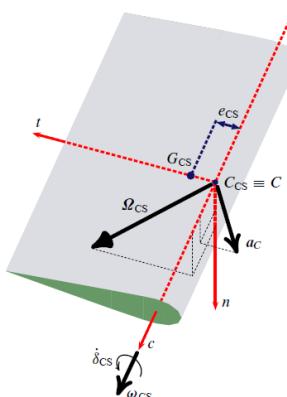
e attraverso alcuni passaggi matematici (riguardando l'angolo di freccia Λ), si ottiene la formula:

$$I_c \ddot{\delta}_{CS} + (\dot{p} + qr) I_{CXB} - (\dot{q} - pr) I_{CYB} + m_{CS}(a_{Gz_B} g_{z_B}) e_{CS} = \mathcal{H}_{A,CS} + \mathcal{H}_{C,CS} \quad (4.6)$$

Che nel caso dell'equilibratore diventa:

$$I_{he} \ddot{\delta}_e - \dot{q}(m_e e_e x_{B,C_e} - I_{he} \cos \Gamma_{he}) + m_e(a_{Gz_B} g_{z_B}) e_e = \mathcal{H}_{A,e} + \mathcal{H}_{C,e} \quad (4.7)$$

Figura 4.1: Assi superficie di governo, centro di massa G_{CS} , accelerazione a_C del polo C e velocità angolare istantanea Ω_{CS} , rispetto al riferimento inerziale. Il vettore $\omega_{CS} = \dot{\delta}_{CS}\mathbf{i}_c$ è la velocità angolare istantanea dell'organo di comando rispetto al velivolo



A comandi liberi e staticamente compensati:

$$I_{he}\ddot{\delta}_e - (\dot{q} - pr)I_{heY_B} + m_e(a_{G_{z_B}}g_{z_B})e_e = \mathcal{H}_{A,e} + \mathcal{H}_{C,e} \quad (4.8)$$

Nell'ipotesi di comandi liberi il pilota non esercita alcuna azione sui comandi e le superfici di governo sono libere di ruotare con la corrente. Le variazioni di forma conseguenti a queste rotazioni generalmente non fanno variare significativamente la distribuzione delle masse del velivolo, motivo per cui è lecito utilizzare le equazioni del moto valide per il corpo rigido. Tuttavia ai 6 gradi di libertà del corpo rigido, se ne aggiungono di ulteriori rappresentati dalle deflessioni delle superfici di controllo. Queste influenzano invece l'entità delle forze e dei momenti aerodinamici e hanno effetto sull'evoluzione del velivolo. Dunque costituiscono nuove variabili di stato, cioè nuove incognite del problema.

E' necessario allora aggiungere al sistema di equazioni differenziali viste nel caso a comandi bloccati tante equazioni quante sono i comandi longitudinali del velivolo, che descrivono l'equilibrio alla rotazione delle superfici di governo attorno al proprio asse di cerniera.

Si procederà con la risoluzione del problema ai valori iniziali per tale sistema di equazioni. Si assume lo stesso modello aerodinamico e propulsivo utilizzato in precedenza. Combinando la (4.7) con la seguente:

$$\begin{aligned} C_{\mathcal{H}_e} &= C_{\mathcal{H}_0} + C_{\mathcal{H}_\alpha} \alpha_H + C_{\mathcal{H}_{\delta_s}} \delta_s + C_{\mathcal{H}_{\delta_t}} \delta_t \\ &\quad \frac{\bar{c}_e}{2V} (C_{\mathcal{H}_{\dot{\alpha}}} \dot{\alpha}_H + C_{\mathcal{H}_q} q + C_{\mathcal{H}_{\dot{\delta}_e}} \dot{\delta}_e) \end{aligned} \quad (4.9)$$

Si ottiene questo nuovo sistema:

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & M_{32} & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{Bmatrix} \dot{V} \\ \dot{\alpha} \\ \dot{q} \\ \dot{x}_{E,G} \\ \dot{z}_{E,G} \\ \dot{\theta} \end{Bmatrix} = \begin{Bmatrix} f_1(V, \alpha, , z_{E,G}, \theta, \delta_T, \delta_e, \delta_s, \delta_t,) \\ f_2(V, \alpha, q, z_{E,G}, \theta, \delta_T, \delta_e, \delta_s, \delta_t,) \\ f_3(V, \alpha, , z_{E,G}, , \delta_T, \delta_e, \delta_s, \delta_t, \dot{\delta}_e) \\ f_4(V, \alpha, , , \theta, , , , ,) \\ f_5(V, \alpha, , , \theta, , , , ,) \\ f_6(, , q, , , , , ,) \end{Bmatrix} \quad (4.10)$$

Si è tenuto conto anche degli effetti dovuti alla velocità di deflessione dell'elevatore $\dot{\delta}_e$. L'elemento

$$M_{32} = -\frac{1}{4\mu} \frac{\bar{c}^2}{k_y^2} \frac{V}{b} C_{\mathcal{M}_{\dot{\alpha}}} \quad (4.11)$$

è un termine di massa apparente. Le variabili di stato sono:

$$\begin{Bmatrix} V \\ \alpha \\ q \\ x_{G,E} \\ z_{G,E} \\ \theta \end{Bmatrix} = \begin{Bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \end{Bmatrix} \quad (4.12)$$

Mentre le variabili in input sono le grandezze:

$$\begin{Bmatrix} \delta_T \\ \delta_e \\ \delta_s \\ \delta_t \end{Bmatrix} = \begin{Bmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \end{Bmatrix} \quad (4.13)$$

Le grandezze:

$$\begin{aligned} f_1 &= g \left\{ \frac{u_1 T_{max}(x_1, x_5)}{W} \cos(x_2 - \mu_x + \mu_T) - \sin(x_6 + \mu_x - x_2) \right. \\ &\quad \left. - \frac{\rho x_1^2}{2(W/S)} [C_{D_0} + k(C_{L_\alpha} x_2 + C_{L_{\delta_e}} u_2 + C_{L_{\delta_s}} u_3)^m] \right\} \end{aligned} \quad (4.14)$$

$$\begin{aligned} f_2 &= \frac{1}{1 + \frac{\bar{c}/b}{4\mu(x_5)} C_{L_{\dot{\alpha}}}} \left[x_3 \left(1 - \frac{\bar{c}/b}{4\mu(x_5)} C_{L_q} \right) \right. \\ &\quad \left. - \frac{u_1 T_{max}(x_1, x_5)}{W} \frac{g}{x_1} \sin(x_2 - \mu_x + \mu_T) + \frac{g}{x_1} \cos(x_6 + \mu_x - x_2) \right. \\ &\quad \left. - \frac{\rho x_1^2}{2(W/S)} \frac{g}{x_1} (C_{L_\alpha} x_2 + C_{L_{\delta_e}} u_2 + C_{L_{\delta_s}} u_2) \right] \end{aligned} \quad (4.15)$$

$$f_3 = \frac{x_1^2}{k_y^2} \frac{\bar{c}/b}{4\mu(x_5)} \left[C_{\mathcal{M},T}(u_1) + C_{\mathcal{M}_0} + C_{\mathcal{M}_\alpha} x_2 + C_{\mathcal{M}_{\delta_e}} u_2 + C_{\mathcal{M}_{\delta_s}} u_3 + C_{\mathcal{M}_{\delta_t}} u_4 + \frac{\bar{c}}{2x_1} (C_{\mathcal{M}_q} x_3 + C_{\mathcal{M}_{\delta_e}} \dot{u}_2) \right] \quad (4.16)$$

$$f_4 = x_1 \cos(x_6 + \mu_x - x_2) \quad (4.17)$$

$$f_5 = -x_1 \sin(x_6 + \mu_x - x_2) \quad (4.18)$$

$$f_6 = x_3 \quad (4.19)$$

sono funzioni delle variabili di stato, del grado di parzializzazione della spinta $\delta_T = T/T_{max}(h, M)$, delle escursione δ_e, δ_s e δ_t e di $\dot{\delta}_e$.

L'equazione della dinamica dell'equilibratore diventa:

$$I_{h_e} \ddot{\delta}_e - \dot{q}(m_e e_e x_{B,C_e} - I_{h_e} \cos \Lambda_{h_e}) + m_e (a_{G_{z_B}} - g_{z_B}) e_e = \mathcal{H}_{A,e} \quad (4.20)$$

in cui $\mathcal{H}_{A,e}$ è il momento di cerniera aerodinamico agente sull'equilibratore, la cui legge è:

$$\begin{aligned} \mathcal{H}_{A,e} &= q_\infty S_e c_e C_{\mathcal{H}_{A,e}} \\ &= q_\infty S_e c_e [C_{\mathcal{H}_0} + C_{\mathcal{H}_\alpha} \alpha_H + C_{\mathcal{H}_{\delta_e}} \delta_e + C_{\mathcal{H}_{\delta_s}} \delta_s + C_{\mathcal{H}_{\delta_t}} \delta_t \\ &\quad + \frac{\bar{c}_e}{2V} (C_{\mathcal{H}_\alpha} \alpha_H + C_{\mathcal{H}_q} q + C_{\mathcal{H}_{\delta_e}} \delta_e)] \end{aligned} \quad (4.21)$$

L'equazione (4.20) è una ODE del 2° ordine, che va ad affiancarsi a un sistema di equazioni di ODE del 1° ordine, tenendo conto che:

$$\delta_e = \frac{d(\delta_e)}{dt} \quad (4.22)$$

In più, considerando che:

$$a_{G_{z_B}} - g_{z_B} = [\dot{V} \sin(\alpha - \mu_x) - V \dot{\gamma} \cos(\alpha - \mu_x)] - g \cos \theta \quad (4.23)$$

Ricordando che:

$$\gamma = \theta - \alpha + \mu_x \quad \dot{\gamma} = \dot{\theta} - \dot{\alpha} \quad (4.24)$$

e quindi:

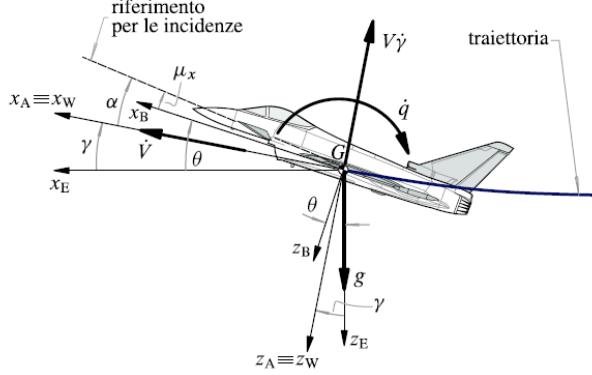
$$\begin{aligned} a_{G_{z_B}} - g_{z_B} &= [\dot{V} \sin(\alpha - \mu_x) - V(\dot{\theta} - \dot{\alpha}) \cos(\alpha - \mu_x)] - g \cos \theta \\ &= [\dot{x}_1 \sin(x_2 - \mu_x) - x_1 \dot{x}_6 \cos(x_2 - \mu_x)] + x_1 \dot{x}_2 \cos(x_2 - \mu_x) - g \cos x_6 \end{aligned} \quad (4.25)$$

Quindi l'equazione (4.20) infine diventa:

$$\begin{aligned} \dot{x}_7 &= \left(\frac{e_e x_{B,C_e}}{k_e^2} - \cos \Lambda_{h_e} \right) \dot{x}_3 \\ &\quad - \frac{e_e}{k_e^2} [\dot{x}_1 \sin(x_2 - \mu_x) - x_1 \dot{x}_6 \cos(x_2 - \mu_x) + x_1 \dot{x}_2 \cos(x_2 - \mu_x) - g \cos x_6] \\ &\quad + \frac{\rho(x_5) x_1^2 S_e \bar{c}_e}{2m_e k_e^2} \left\{ C_{\mathcal{H}_0} + C_{\mathcal{H}_\alpha} \left[\left(1 - \frac{d\epsilon}{d\alpha} \right) (x_2 - \mu_x) - \epsilon_0 + u_3 + \mu_x \right] \right. \\ &\quad \left. + C_{\mathcal{H}_{\delta_e}} x_8 + C_{\mathcal{H}_{\delta_s}} u_3 + C_{\mathcal{H}_{\delta_t}} u_4 + \frac{\bar{c}_e}{2x_1} \left[C_{\mathcal{H}_{\dot{\alpha}}} \left[\left(1 - \frac{d\epsilon}{d\alpha} \right) \dot{x}_2 + C_{\mathcal{H}_q} x_3 + C_{\mathcal{H}_{\delta_e}} \dot{x}_8 \right] \right] \right\} \\ \dot{x}_8 &= x_7 \end{aligned} \quad (4.26)$$

In cui $x_8 = \delta_e$ e $I_{h_e} = m_e k_e^2$ e con k_e il raggio di inerzia dell'equilibratore attorno all'asse di cerniera.

Figura 4.2: Accelerazioni lineari ed angolari nel generico istante di un moto longitudinal-simmetrico. Schema per il calcolo di $a_{Gz_B} - g_{z_B}$.



Dalle osservazioni precedenti, si perviene alla seguente notazione:

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & M_{32} & 1 & 0 & 0 & 0 & 0 & M_{38} \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ M_{71} & M_{72} & M_{73} & 0 & 0 & M_{76} & 1 & M_{78} \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{Bmatrix} \dot{x}_1 \\ \dot{x}_2 \\ \dot{x}_3 \\ \dot{x}_4 \\ \dot{x}_5 \\ \dot{x}_6 \\ \dot{x}_7 \\ \dot{x}_8 \end{Bmatrix} = \begin{Bmatrix} f_1 \\ f_2 \\ f_3 \\ f_4 \\ f_5 \\ f_6 \\ f_7 \\ f_8 \end{Bmatrix} \quad (4.27)$$

In cui i termini della nuova matrice di massa sono:

$$M_{38} = -\frac{1}{4\mu(x_5)} \frac{\bar{c}^2}{k_y^2} \frac{x_1}{b} C_{\mathcal{M}_{\dot{\delta}_e}} \quad (4.28)$$

$$M_{71} = \frac{e_e}{k_e^2} x_1 \sin(x_2 - \mu_x) \quad (4.29)$$

$$M_{72} = \frac{e_e}{k_e^2} x_1 \cos(x_2 - \mu_x) - \frac{\rho(x_5) x_1 S_e \bar{c}_e^2}{4m_e k_e^2} \left(1 - \frac{d\epsilon}{d\alpha} \right) C_{\mathcal{H}_{\dot{\alpha}}} \quad (4.30)$$

$$M_{73} = -\left(\frac{e_e x_{B,C_e}}{k_e^2} - \cos \Lambda_{h_e} \right) \quad (4.31)$$

$$M_{76} = -\frac{e_e}{k_e^2} x_1 \cos(x_2 - \mu_x) \quad (4.32)$$

$$M_{78} = \frac{\rho(x_5) x_1 S_e \bar{c}_e^2}{4m_e k_e^2} \left(1 - \frac{d\epsilon}{d\alpha} \right) C_{\mathcal{H}_{\dot{\delta}_e}} \quad (4.33)$$

Le funzioni f_1, f_2, f_4, f_5 e f_6 sono date dalle precedenti definizioni con x_8 al posto di u_2 . La f_3 è ridefinita come segue:

$$f_3 = \frac{x_1^2}{k_y^2} \frac{\bar{c}/b}{2\mu(x_5)} \left[C_{\mathcal{M},T} + C_{\mathcal{M}_0} + C_{\mathcal{M}_\alpha} x_2 + C_{\mathcal{M}_{\dot{\delta}_e}} x_8 + C_{\mathcal{M}_{\dot{\delta}_s}} u_3 + \frac{\bar{c}}{2x_1} C_{\mathcal{M}_q} x_3 \right] \quad (4.34)$$

sottraendo il termine $M_{38}\dot{x}_8$. Le f_7 e f_8 sono:

$$\begin{aligned} f_7 &= \frac{e_e}{k_e^2} g \cos x_6 + \frac{\rho(x_5) x_1^2 S_e \bar{c}_e}{2m_e k_e^2} \left\{ C_{\mathcal{H}_0} + C_{\mathcal{H}_\alpha} \left[\left(1 - \frac{d\epsilon}{d\alpha} \right) (x_2 - \mu_x) - \epsilon_0 + u_3 + \mu_x \right] \right. \\ &\quad \left. + C_{\mathcal{H}_{\dot{\delta}_e}} x_8 + C_{\mathcal{H}_{\dot{\delta}_s}} u_3 + C_{\mathcal{H}_{\dot{\delta}_t}} u_4 + \frac{\bar{c}_e}{2x_1} C_{\mathcal{H}_q} x_3 \right\} \quad (4.35) \end{aligned}$$

$$\dot{f}_8$$

$$= x_7$$

Il sistema (4.27) regge il moto longitudinal-simmetrico di un velivolo rigido a comandi liberi. Ha quindi 4 gradi di libertà, dicui 3 di corpo rigido e 1 rappresentato dall'escursione angolare dell'equilibratore.

Con una notazione più sintetica la (4.27) diventa la:

$$[M(\mathbf{x}, t)]\{\mathbf{x}\} = \{f(\mathbf{x}, t)\} \quad (4.36)$$

da cui si ricava la:

$$\{\mathbf{x}\} = [M(\mathbf{x}, t)]^{-1}\{f(\mathbf{x}, t)\} \quad (4.37)$$

in cui la $[M(\mathbf{x}, t)]$ è la *mass matrix*.

4.2 Input Esercizi

Nell'APP creata, è possibile dare in input i seguenti valori:

- x_0 valore iniziale della coordinata x del CG;
- z_0 valore iniziale della coordinata z del CG;
- V_0 valore iniziale della velocità del CG;
- q_0 valore iniziale della velocità angolare di pitch;
- γ_0 valore iniziale dell'angolo di rampa;
- α_0 valore iniziale dell'angolo di attacco;
- $\delta_{e,0}$ valore iniziale dell'angolo di deflessione dell'elevator;
- $\delta_{s,0}$ valore iniziale dell'angolo di deflessione dello stabilizer;
- $\delta_{T,0}$ valore iniziale del grado di parzializzazione della spinta;
- *Breakpoint t₁* tempo di breakpoint;
- Switch serve a scegliere la lunghezza della simulazione.

Per tempo di breakpoint si indica il tempo in cui c'è un cambio nei comandi del velivolo. È un parametro che è in uso dal secondo esercizio in poi.

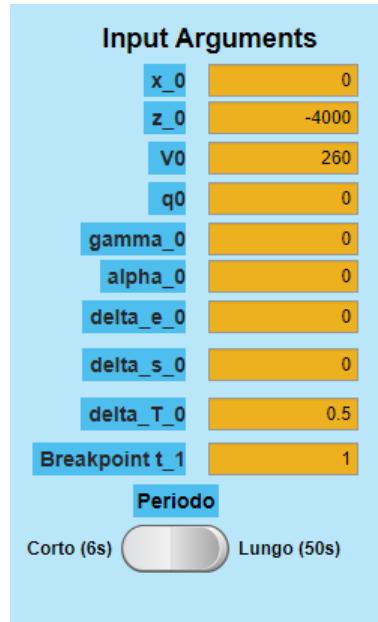


Figura 4.3: Valori in Input

4.3 Esercizio 1 (11.1)

L'obiettivo di questo esercizio è la risoluzione della dinamica a comandi liberi. Si ricercano le condizioni di trim e poi si ipotizzano costanti tutti i comandi tranne la deflessione dell'equilibratore, libero di ruotare intorno all'asse di cerniera. Esso presenta oscillazioni iniziali fino al raggiungimento della condizione stazionaria che dipende anche dalla deflessione dell'aletta tab δ_t . Si studia infine la storia temporale del momento di cerniera. Il codice necessario per il calcolo è il seguente.

```

1 function [] = func111(INex11_1)
2 % -----
3 % Esercizi Capitolo 11
4 % -----
5 % Esercizio N°1
6 % -----
7
8 %%%%%%%%CALCOLO CONDIZIONI DI TRIM%%%%%%
9 % CALCOLO CONDIZIONI DI TRIM %
10 %%%%%%%%CALCOLO CONDIZIONI DI TRIM%%%%%
11 diary('ex11_1.txt')
12 diary on
13
14 % Ricerca di un set completo di condizioni iniziali per un volo trizzato disp('Moto di un aeroplano a 3-DoF.');
15 disp('Soluzione di un problema di trim ad una data altitudine e velocità di volo');
16
17 %% INPUT
18 % Tipo di Periodo: 1=corto (6s) 2=lungo (50s)
19
20
21 periodo = INex11_1(10);
22 % x_0 = ; % posizione longitudinale [m]
23 % z_0 = -4000.0; % a.s.l. (about sea level) altezza [m]
24 % V0 = 260; % vettore velocità [m/s]
25 % q0 = 0.0; % velocità angolare di pitch [rad/s]
26 % gamma0 = 0.0; % angolo di salita [rad]
27
28 % % Vettore dei valori di primo tentativo
29 % x0 = [ ...
30 % 0; ... % 1) alpha_0
31 % 0; ... % 2) delta_e_0. Deflessione equilibratore
32 % 0; ... % 3) delta_s_0. Deflessione stabilizzatore
33 % 0.5]; % 4) delta_T_0. Manette
34
35 %%
36 % Dichiarazione variabili globali
37
38 global ...
39 g ... % accelerazione di gravità
40 z_0 V0 q0 gamma0 ...% condizioni iniziali
41 rho0 ... % densità dell'aria alla quota Z_0
42 delta_tab ... % deflessione aletta tab
43 delta_s ... % legge di comando stabilizzatore
44 delta_T ... % legge di comando manetta
45 myAC % oggetto aereo. (è una sorta di record con
46 % diversi campi)
47
48 % Aircraft data file name
49 aircraftDataFileName = 'DSV11.txt';
50
51 % Aircraft object
52 myAC = DSVAircraft111(aircraftDataFileName); % viene passato il nome del file
53
54 if (myAC.err == -1) %parametro di controllo per vedere se vi è un
55 disp('... terminazione.');//errore.
56 else
57 disp(['File ', aircraftDataFileName, ' letto correttamente.']);
58
59 % Condizioni iniziali. Condizioni di primo tentativo per la ricerca delle
60 % condizioni di trim
61 x_0 = INex11_1(1); % posizione longitudinale [m]
62 z_0 = INex11_1(2); % a.s.l. (about sea level) altezza [m]
63 V0 = INex11_1(3); % vettore velocità [m/s]
64 q0 = INex11_1(4); % velocità angolare di pitch [rad/s]
65 gamma0 = INex11_1(5); % angolo di salita [rad]
66
67 % Funzione matlab del modello ISA. Lo usiamo per il calcolo della densità
68 [air_Temp_0,sound_speed_0, air_pressure_0, rho0] = atmosisa(-z_0);
69
70 % Accelerazione di gravità
71 g = 9.81; % [m/s^2]
72
73
74 % Vettore dei valori di primo tentativo
75 x0 = [ ...
76 INex11_1(6); ... % 1) alpha_0
77 INex11_1(7); ... % 2) delta_e_0. Deflessione equilibratore
78 INex11_1(8); ... % 3) delta_s_0. Deflessione stabilizzatore

```

```

79 INex11_1(9)]; % 4) delta_T_0. Manette
80 %%%%%%% MINIMIZZAZIONE DELLA FUNZIONE DI COSTO %%%%%%
82
83 % Limiti inferiori (low bound)
84 lb =[convang(-15,'deg','rad'),...%minimum alpha
85 convang(-20,'deg','rad'),...%minimum elevator deflection (delta_e)
86 convang(-5,'deg','rad'),...%minimum stabilizer incidence(delta_s)
87 0.2 ... % minimum thrust fraction (delta_T)
88 ];
89
90 % Limiti superiori (upper bound)
91 ub =[convang( 15,'deg','rad'), ... % maximum alpha
92 convang( 13,'deg','rad'), ... % maximum elevator deflection
93 convang( 5,'deg','rad'), ... % maximum stabilizer incidence
94 1.0 ... % maximum thrust fraction
95 ];
96
97 %Vincolo lineare per delta_s_0 (ipostizziamo sia nullo)
98 Aeq = zeros(4,4);
99 beq = zeros(4,1); Aeq(3,3) = 1;
100 delta_s_0 = convang(0,'deg','rad'); % Ipozziamo deflessione nulla dello stabilatore
101 beq(3,1) = delta_s_0; % manteniamo fisso delta_s
102
103 options = optimset( ...
104 'tolfun',1e-9, ... % tolleranza sull'errore (tolleration function)
105 'Algorithm','interior-point'); %tipologia di algoritmo da utilizzare
106
107 % Funzione per ricavare la condizione di trim
108 [x,fval] = ...
109 fmincon(@costLongEquilibriumStaticStickFixed, ... % funzione per ò ricerca del minimo
110 x0, ... % valore di primo tentativo
111 [], ... % A, A*x=b
112 [], ... % b
113 Aeq, ... % Aeq, Aeq*x=beq
114 beq, ... % beq
115 lb,ub, ... % lower, upper bounds
116 @myNonLinearConstraint, ... % nonlinear const.
117 options); % opzioni di ricerca del minimo
118
119 alpha_0_rad = x(1);
120 alpha_0_deg = convang(x(1),'rad','deg'); delta_e_0_rad = x(2);
121 delta_e_0_deg = convang(x(2),'rad','deg'); delta_s_0_rad = x(3);
122 delta_s_0_deg = convang(x(3),'rad','deg'); delta_T_0 = x(4);
123
124 theta_trim = gamma0+alpha_0_rad-myAC.mu_x;
125
126 % Angolo di Eulero iniziale.
127 % Serve per definire le condizioni iniziali. Si veda la fig. (7.8)
128 % Si osservi che alpha è l'angolo assoluto, non alpha body
129
130 % Mostro a display i risultati ottenuti
131 disp('Le condizioni di trim iniziali ottenute sono: ');
132 disp(['alpha_0 = ', num2str(alpha_0_deg), '']);
133 disp(['delta_e_0 = ', num2str(delta_e_0_deg), '']);
134 disp(['delta_s_0 = ', num2str(delta_s_0_deg), '']);
135 disp(['delta_T_0 = ', num2str(delta_T_0)]);
136 %%%%%% RISOLUZIONE EQUAZIONI DELLA DINAMICA %%%%%%
137 % Risolto il problema del trim si passa alla risoluzione delle equazioni
138 % della dinamica per le quali si fa riferimento alle eq. (7.61)
139
140 % Istanti di inizio e fine osservazione del fenomeno
141 t_0 = 0; % istante iniziale
142 switch periodo
143 case 1
144     t_finale = 6; % istante finale di osservazione del fenomeno
145 case 2
146     t_finale = 50; % istante finale di osservazione del fenomeno
147 end
148
149 % Mesh
150 N = 200; % numero di punti
151 time = linspace (t_0, t_finale, N);% punti mesh uniforme
152 % Leggi di comando
153 delta_tab = @(t) interp1 ([t_0 t_finale], [0 0], t);
154 delta_s = @(t) interp1([t_0 t_finale],[delta_s_0_rad delta_s_0_rad],t);
155 delta_T = @(t) interp1([t_0 t_finale], [delta_T_0 delta_T_0], t);
156
157 % Vettore condizioni iniziali
158 X0 = [ V0;
159 alpha_0_rad; q0;
160 x_0; z_0;
161 theta_trim
162 0; % derivata prima deflessione equilibratore nullo
163 delta_e_0_rad];% deflessione iniziale pari a quella ottenuta dalle condizioni di trim
164
165
166 % Risolvo il sistema di equazioni iniziali X0 mediante l'ode 45

```

```

168 options = odeset( 'RelTol', 1e-9, 'AbsTol', 1e-9*ones(1,8));
169
170 [vTime, X] = ode45(@eqLongDynamicStickFree, ... % eq. (7.61)
171 time, ... % Vettore dei tempi
172 X0', ... % Vettore iniziale
173 options); % Vettore delle opzioni
174
175
176 % Incrementi rispetto i valori iniziali
177 DELTA_V = X(:,1) - VO;
178 DELTA_alpha = convang(X(:,2), 'rad', 'deg') - alpha_0_deg;
179 DELTA_q = convangvel(X(:,3), 'rad/s', 'deg/s') - q0;
180 X_EG = X(:,4);
181 DELTA_h = -(X(:,5) - z_0);
182 THETA = X(:,6);
183 GAMMA = THETA + myAC.mu_x - X(:,2);
184 DELTA_E = X(:,8);
185
186 %%%%%% FATTORI DI CARICO %%%%%%
187 %
188 %%%%%%
189
190 %Per calcolare q_punto, f_xA e f_zA, occorre riprendere le equazioni 7.61
191 %e calcolare i vettori V_punto e gamma_punto. Si osservi che gamma_punto
192 %= theta_punto - alpha_punto (è sufficiente derivare l'equazione 7.64)
193 %eq. (7.61a)
194
195 V = X(:,1);
196 alpha = X(:,2);
197 q = X(:,3);
198 z_EG = X(:,5);
199 theta = X(:,6);
200 delta_e = X(:,8);
201
202
203 % Funzione matlab del modello ISA. Lo usiamo per il calcolo della densità
204 % Serve per aggiornare i parametri atmosferici dato che z_EG può variare
205 [air_Temp_0,sound_speed_0, air_pressure_0, rho] = atmosisa(-z_EG);
206 mu_rel = (myAC.W/g)./(rho*myAC.S*myAC.b); % m/(rho*S*b)
207
208
209 % Calcolo delle derivate mediante la funzione gradient
210 V_punto = gradient(X(:,1), time);
211 alpha_punto = gradient(X(:,2), time);
212 q_punto = gradient(X(:,3), time);
213
214
215 % eq. (7.61f)
216 theta_punto = q;
217 % Si ricavano, di seguito, le ulteriori funzioni di interesse, quali
218 % gamma_punto e le funzioni fattore di carico lungo gli assi aerodinamici
219 % xA e zA
220 gamma_punto = theta_punto - alpha_punto;
221 f_xa = -(sin(GAMMA)+V_punto/g); % fattore di carico lungo xa
222 f_za = cos(GAMMA) + (V.*gamma_punto)/g; % fattore di carico lungo za
223
224
225
226 %%%%%% MOMENTO DI CERNIERA %%%%%%
227 %
228 %%%%%%
229
230 CH_A_e = myAC.Ch_e_0 + ...
231 myAC.Ch_e_alpha*((1-myAC.DepsDalp)*alpha - myAC.mu_x) - ...
232 myAC.eps_0+delta_s([vTime])+myAC.mu_x)+...
233 myAC.Ch_e_delta_e*delta_e+...
234 myAC.Ch_e_delta_s*delta_s([vTime])+...
235 myAC.Ch_e_delta_tab*delta_tab([vTime])+...
236 myAC.mac_e*myAC.Ch_e_q*q'/(2*V');
237
238 H_A_e = 0.5*rho'*V.^2*myAC.S_e*myAC.mac_e*CH_A_e;
239
240 %% Grafica 1
241
242 fig1=figure("NumberTitle","off","Name","Leggi di comando, angolo di deflessione, fattore di carico e
accelerazione angolare")
243 tiledlayout(3,2,"TileSpacing","Tight");
244 nexttile
245 plot(time,delta_tab(time),'-k','LineWidth',1.5)
246 xlabel("t (s)"); ylabel("\delta_{tab,0} (deg)"); grid on;
247 nexttile
248 plot(time,convang(DELTA_E,'rad','deg'),'-k','LineWidth',1.5)
249 xlabel("t (s)"); ylabel("\delta_e (deg)"); grid on;
250 nexttile
251 plot(time,delta_s(time),'-k','LineWidth',1.5)
252 xlabel("t (s)"); ylabel("\delta_s,0 (deg)"); grid on;
253 nexttile
254 plot(time,convang(q_punto,'rad','deg'),'-k','LineWidth',1.5)
255 xlabel("t (s)"); ylabel("$\dot{q} (deg/s^2)$", 'Interpreter','latex'); grid on;

```

```

256 nexttile
257 plot(time,delta_T(time),'-k','LineWidth',1.5)
258 xlabel("t (s)"); ylabel("\delta_{T,0} (deg)"); grid on;
259 nexttile
260 plot(time,f_xa,'-b',time,f_za,'-g','LineWidth',1.5)
261 xlabel("t (s)"); grid on;
262 legend("f_{xA}", "f_{zA}");
263
264 %% Grafica 2
265
266
267 fig2=figure("NumberTitle","off","Name","Storie temporali delle variabili di stato")
268 h(1) = subplot(3,2,1);
269 plot(time,X_EG,'-k','LineWidth',1.5);
270 grid on
271
272 ylabel('$x_{EG}$ (m)', 'interpreter', 'latex', 'fontsize', 11)
273 h(2) = subplot(3,2,2);
274 plot(vTime,DELTA_h,'-k','LineWidth',1.5);
275 grid on
276
277 ylabel('$\Delta h$ (m)', 'interpreter', 'latex', 'fontsize', 11)
278 h(3) = subplot(3,2,3);
279 plot(vTime,DELTA_V,'-k','LineWidth',1.5);
280 grid on
281
282 ylabel('$\Delta V$ (m/s)', 'interpreter', 'latex', 'fontsize', 11)
283 h(4) = subplot(3,2,4);
284 plot(vTime,DELTA_alpha,'-k','LineWidth',1.5);
285 grid on
286
287 ylabel('$\Delta \alpha$ (deg)', 'interpreter', 'latex', 'fontsize', 11)
288 h(5) = subplot(3,2,5);
289 plot(vTime,DELTA_q,'-k','LineWidth',1.5);
290 grid on
291
292 xlabel('t (s)', 'interpreter', 'latex', 'fontsize', 11);
293 ylabel('$\Delta q$ (deg/s)', 'interpreter', 'latex', 'fontsize', 11)
294 h(6) = subplot(3,2,6);
295 plot(vTime,convang(THETA,'rad','deg'), '--b','LineWidth',1.5);
296 hold on;
297 plot(vTime,convang(GAMMA,'rad','deg'), '-.g','LineWidth',1.5);
298 grid on
299 lgd = legend('$\theta(t)$', '$\gamma(t)$');
300 lgd.Interpreter = 'latex';
301 lgd.FontSize = 11;
302
303 xlabel('t (s)', 'interpreter', 'latex', 'fontsize', 11);
304
305 ylabel('$(deg)$', 'interpreter', 'latex', 'fontsize', 11)
306
307 %%
308 fig3=figure("NumberTitle","off","Name","Storie temporali del momento di cerniera")
309 plot(vTime,H_A_e,'-k','LineWidth',.5);
310 grid on
311
312 xlabel('t (s)', 'interpreter', 'latex', 'fontsize', 11);
313 ylabel('$H_{A_e}(N*m)$', 'interpreter', 'latex', 'fontsize', 11)
314 diary off
315
316 saveas(fig1,"Fig11_1_1.eps",'epsc')
317 saveas(fig2,"Fig11_1_2.eps",'epsc')
318 saveas(fig3,"Fig11_1_3.eps",'epsc')
319 end
320 end

```

MassStickFree

```

1 function M = MassStickFree(t,x)
2
3 % Dichiarazione variabili globali
4 global ...
5 g ... % accelerazione di gravità
6 myAC % dati aereo
7 % Si rinominano le componenti del vettore di stato
8 V = x(1);
9 alpha = x(2);
10 q = x(3);
11 xEG = x(4);
12 zEG = x(5);
13 theta = x(6);
14 dot_delta_e = x(7); % derivata angolo di deflessione equilibratore
15 delta_e = x(8); % angolo di deflessione equilibratore
16
17 % Calcolo delle proprietà dell'aria mediante modello ISA
18 [air_Temp, sound_speed, air_pressure, rho] = atmosisa(-zEG);
19
20 % densità relativa del velivolo
21 mu_rel = (myAC.W/g)/(rho*myAC.S*myAC.b);

```

```

22 % Matrice di massa. Si vedano le eq. (11.75)
23 M = eye(8); % costruzione matrice identica 8x8
24
25 M(3,2) = ...
26 -(1/(4*mu_rel))*(myAC.mac^2/myAC.k_y^2) ...
27 *(V/myAC.b)*myAC.Cm_alpha_dot;
28
29 M(3,8) = ...
30 -(1/(4*mu_rel))*(myAC.mac^2/myAC.k_y^2) ...
31 *(V/myAC.b)*myAC.Cm_delta_e_dot;
32
33 M(7,1) = ...
34 (myAC.mac_e*myAC.ec_adim)/myAC.k_e^2 ...
35 *sin(alpha-myAC.mu_x);
36
37 M(7,2) = ...
38 ((myAC.mac_e*myAC.ec_adim)/myAC.k_e^2) ...
39 *V*cos(alpha-myAC.mu_x) ...
40 -rho*V*myAC.S_e*myAC.mac_e^2/(4*myAC.I_e) ...
41 *(1 - myAC.DepsDalpha)*myAC.Ch_e_alpha_dot;
42
43 M(7,3) = ...
44 cos(myAC.Lambda_e) ...
45 - myAC.mac_e*myAC.ec_adim*myAC.x_C_e/myAC.k_e^2;
46
47 M(7,6) = ...
48 -(myAC.mac_e*myAC.ec_adim)/myAC.k_e^2 ...
49 *V*cos(alpha-myAC.mu_x);
50
51 M(7,8) = ...
52 -rho*V*myAC.S_e*myAC.mac_e^2/(4*myAC.I_e) ...
53 *myAC.Ch_e_delta_e_dot;
54
55 end

```

costLongEquilibriumStaticStickFixed

```

1 function [f] = costLongEquilibriumStaticStickFixed(x)
2
3 % Dichiarazione delle variabili globali
4 global g ...
5 zEG_0 V0 q0 gamma0 ...
6 rho0 ...
7 myAC
8
9 mu_rel = (myAC.W/g)/(rho0*myAC.S*myAC.b);
10
11 % Assegnazione delle componenti del design vector
12 alpha = x(1);
13 delta_e = x(2);
14 delta_s = x(3);
15 delta_T = x(4);
16
17 F1 = (delta_T*myAC.T/myAC.W)*cos(alpha - myAC.mu_x + myAC.mu_T) - ...
18 sin(gamma0) - ...
19 ((rho0*V0^2)/(2*(myAC.W/myAC.S)))*...
20 (myAC.CD_0 + myAC.K*((myAC.CL_alpha*alpha + myAC.CL_delta_e*delta_e + ...
21 myAC.CL_delta_s*delta_s)^myAC.m));
22
23 F2 = q0*(1 - ((myAC.mac/myAC.b)/(4*mu_rel))*myAC.CL_q) - ...
24 ((delta_T*myAC.T/myAC.W)*(g/V0))*sin(alpha - myAC.mu_x + myAC.mu_T) + ...
25 (g/V0)*cos(gamma0) - ...
26 ((rho0*V0^2)/(2*(myAC.W/myAC.S)))*(g/V0)*...
27 (myAC.CL_alpha*alpha + myAC.CL_delta_e*delta_e + myAC.CL_delta_s*delta_s);
28
29 F3 = (myAC.Cm_T_0 + myAC.Cm_T_alpha*alpha)*delta_T + ...
30 myAC.Cm_0 + myAC.Cm_alpha*alpha + ...
31 myAC.Cm_delta_e*delta_e + ...
32 myAC.Cm_delta_s*delta_s + ...
33 (myAC.mac/(2*V0))*myAC.Cm_q*q0;
34
35 % Funzione obiettivo
36 f = F1*F1 + F2*F2 + F3*F3;
37
38 end

```

eqLongDynamicStickFree

```

1 function df = eqLongDynamicStickFree(t,x)
2 % Funzione per l'implementazione delle equazioni 7.61
3
4 % Dichiarazione variabili globali
5 global ...
6 g ... % accelerazione di gravità
7 delta_tab ... % deflessione aletta tab
8 delta_s ... % legge di comando stabilizzatore
9 delta_T ... % legge di comando manetta
10 myAC % oggetto aereo
11

```

```

12 % Si assegna un nome delle componenti del vettore di design
13 V = x(1);
14 alpha = x(2);
15 q = x(3);
16 x_EG = x(4);
17 z_EG = x(5);
18 theta = x(6);
19 delta_e_punto = x(7);
20 delta_e = x(8);
21
22 % Funzione matlab del modello ISA. Lo usiamo per il calcolo della densità
23 % Serve per aggiornare i parametri atmosferici dato che z_EG può variare
24 [air_Temp_0,sound_speed_0, air_pressure_0, rho] = atmosisa(-z_EG);
25
26 % densità relativa aereo
27 mu_rel = (myAC.W/g)/(rho*myAC.S*myAC.b); % m/(rho*S*b)
28
29 % Secondi membri delle eq. (11.74)
30 % eq. (7.61a) e (11.68a)
31 F1 = g.*((delta_T(t)*myAC.T/myAC.W)*cos(alpha - myAC.mu_x + myAC.mu_T)...
32 -sin(myAC.mu_x + theta - alpha) ...
33 -((rho*V^2)/(2*(myAC.W/myAC.S))) ...
34 *(myAC.CD_0 + myAC.K*((myAC.CL_alpha*alpha ...
35 + myAC.CL_delta_e*delta_e ...
36 + myAC.CL_delta_s*delta_s(t))^myAC.m)));
37
38 % eq. (7.61b) e (11.68b)
39 F2 = ((1. - myAC.CL_q*(myAC.mac/myAC.b)/(4*mu_rel))*q ...
40 -(g/V)*(delta_T(t)*myAC.T/myAC.W) ...
41 *sin(alpha - myAC.mu_x + myAC.mu_T) ...
42 +(g/V)*cos(myAC.mu_x + theta - alpha) ...
43 -((g/V)*((rho*V^2)/(2*(myAC.W/myAC.S))) ...
44 *(myAC.CL_alpha*alpha + myAC.CL_delta_e*delta_e ...
45 + myAC.CL_delta_s*delta_s(t))) ...
46 *1/(1. + myAC.CL_alpha_dot*(myAC.mac/myAC.b)/(4*mu_rel));
47
48 % eq. (7.61c) e e (11.68c)
49 F3 = (myAC.Cm_0 + myAC.Cm_alpha*alpha ...
50 + myAC.Cm_T_0 + myAC.Cm_T_alpha*alpha...
51 + myAC.Cm_delta_s*delta_s(t) ...
52 + myAC.Cm_delta_e*delta_e ...
53 +(myAC.mac/(2*V))*myAC.Cm_q*q ...
54 +(myAC.mac/(2*V))*myAC.Cm_alpha_dot*F2)...
55 *((V/myAC.k_y)^2*(myAC.mac/myAC.b)/(2*mu_rel));
56 % eq. (7.61d) e (11.68d)
57 F4 = V*cos(theta + myAC.mu_x - alpha);
58
59 % eq. (7.61e) e e (11.68e)
60 F5 = -V*sin(theta + myAC.mu_x - alpha);
61
62 % eq. (7.61f) e (11.68f)
63 F6 = q;
64 % eq. (11.77a)
65 F7 = (((myAC.ec_adim*myAC.mac_e)/myAC.k_e^2)*g*cos(theta)...
66 +(rho*V^2*myAC.S_e*myAC.mac_e)/(2*myAC.I_e)...
67 *(myAC.Ch_e_0 + myAC.Ch_e_alpha * ((1-myAC.DepsDalpha)...
68 *(alpha - myAC.mu_x) - myAC.eps_0 + delta_s(t) + myAC.mu_x)...
69 + myAC.Ch_e_delta_e*delta_e ...
70 + myAC.Ch_e_delta_s*delta_s(t) ...
71 + myAC.Ch_e_delta_tab*delta_tab(t) ...
72 +(myAC.mac_e/(2*V))*myAC.Ch_e_q*q); % myAC.ec_adim è l'eccentricità dell'equilibratore adimensionalizzata
    per mac_e
73
74 % eq. (11.78b)
75 F8 = delta_e_punto;
76
77 % Si richiama la funzione per creare la matrice delle masse
78 M_mass = MassStickFree(t,x);
79 % Costruzione della funzione di costo
80 f = [F1;
81 F2;
82 F3;
83 F4;
84 F5;
85 F6;
86 F7; F8];
87
88 df = inv(M_mass)*f;
89 end

```

myNonLinearConstraint

```

1 function [c,ceq] = myNonLinearConstraint(x)
2
3 c = []; %Assegnazione di relazioni non lineari di disuguaglianza
4 ceq = []; %Assegnazione di relazioni non lineari di uguaglianza
5
6 end

```

4.3.1 Risultati

Per i dati precedentemente assegnati le condizioni di trim sono:

Condizioni di Trim

Soluzione di un problema di trim ad una data altitudine e velocità di volo
File DSV11.txt aperto.
File DSV11.txt letto correttamente.

Le condizioni di trim iniziali ottenute sono:
 $\alpha_0 = 2.019^\circ$
 $\delta_{e,0} = -4.3981^\circ$
 $\delta_{s,0} = 4.3398e-22^\circ$
 $\delta_{T,0} = 0.41716$

Le successive figure mostrano le storie temporali delle leggi di comando bloccate, la storia temporale dei fattori di carico, le storie temporali delle variabili di stato per comandi liberi, la storia temporale della deflessione dell'eqilibratore e del momento di cerniera.

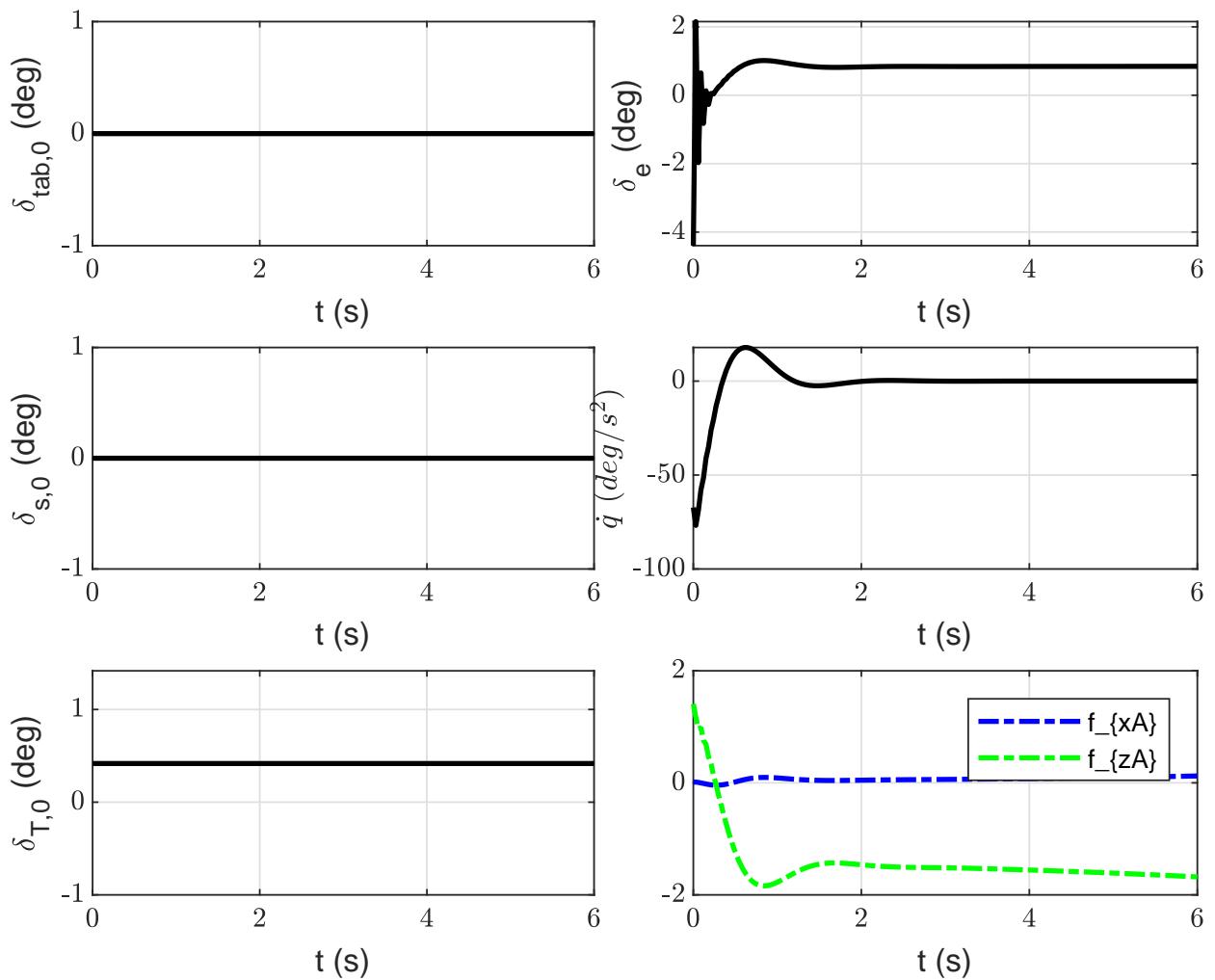


Figura 4.4: Esercizio 1 - Leggi di comando, angolo di deflessione, fattore di carico e accelerazione angolare

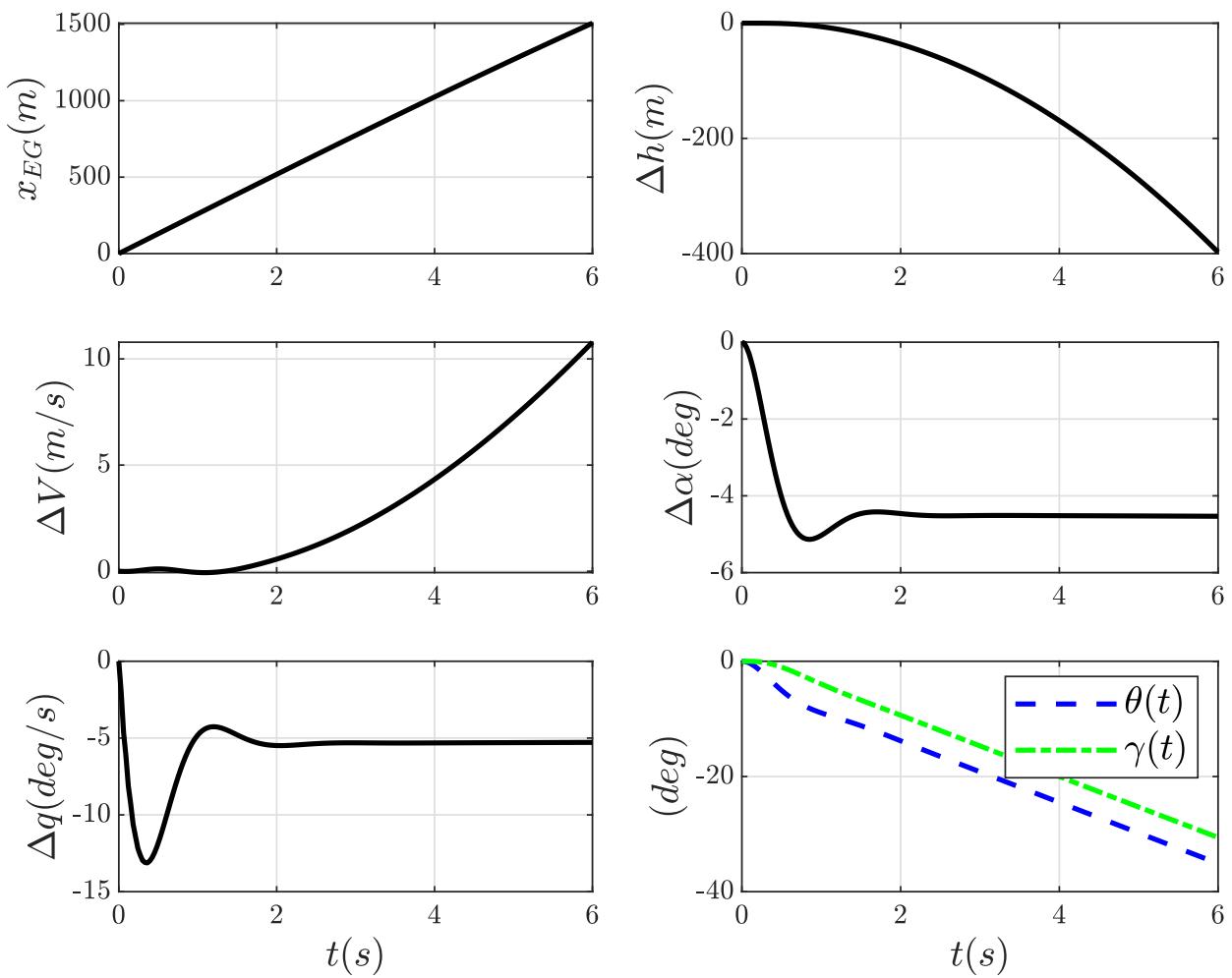


Figura 4.5: Esercizio 1 - Storie temporali delle variabili di stato

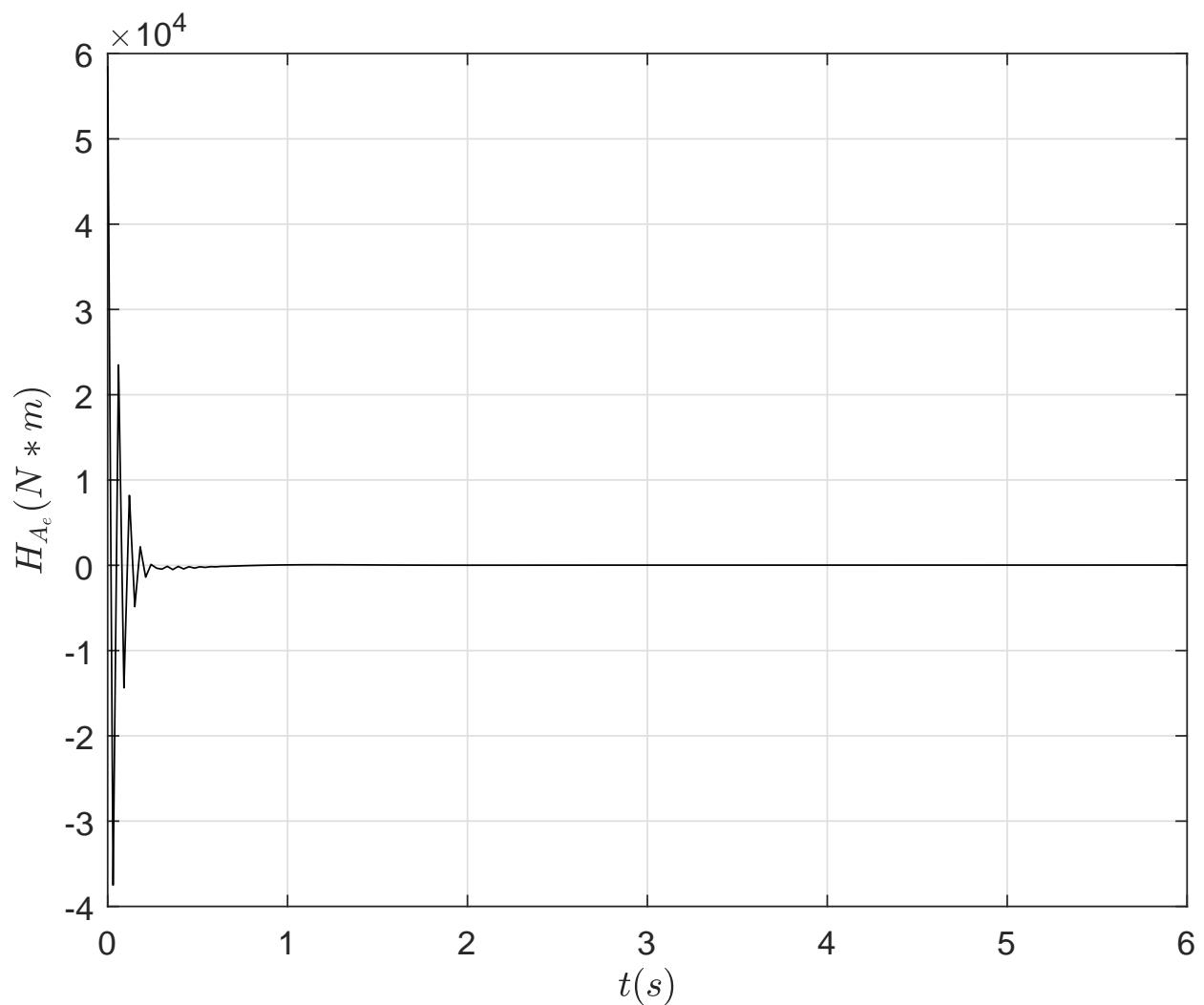


Figura 4.6: Esercizio 1 - Storie temporali del momento di cerniera

4.4 Esercizo 2 (11.2)

Il seguente esercizio prevede l'integrazione numerica delle equazioni del moto longitudinal-simmetrico. La prima parte del moto è a comandi bloccati (in condizioni di trim), successivamente si avrà un moto a comandi liberi (ipotizzando la deflessione dell'aletta tab nulla). Si risolve dunque prima il problema a comandi bloccati e poi quello a comandi liberi, utilizzando i metodi precedentemente visti. Il tempo in cui c'è il cambio fra comandi liberi è bloccati viene detto t_1 (Breakpoint), ed è uno dei valori in input nell'applicativo. Il codice utilizzato è il seguente listato.

```

1 function [] = func112(INex11_2)
2 %
3 %   Esercizi Capitolo 11
4 %
5 %   Esercizio N°2
6 %
7
8
9 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
10 %           CALCOLO CONDIZIONI DI TRIM
11 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
12
13 diary('ex11_2.txt')
14 diary on
15 % Ricerca di un set completo di condizioni iniziali per un volo trizzato disp('Moto di un aeroplano a 3-DoF.');
16 disp('Soluzione di un problema di trim ad una data altitudine e velocità di volo');
17
18 % INPUT
19 % Tipo di Periodo: 1=corto (6s)      2=lungo (50s)
20
21
22 %load("INex11_2.mat")
23 periodo = INex11_2(10);
24 t_switch = INex11_2(11);
25 % x_0 = ;                         % posizione longitudinale [m]
26 % z_0 = -4000.0;                   % a.s.l. (about sea level) altezza [m]
27 % V0 = 260;                        % vettore velocità [m/s]
28 % q0 = 0.0;                        % velocità angolare di pitch [rad/s]
29 % gamma0 = 0.0;                    % angolo di salita [rad]
30
31 % % Vettore dei valori di primo tentativo
32 % x0 = [ ...
33 % 0; ...                           % 1) alpha_0
34 % 0; ...                           % 2) delta_e_0. Deflessione equilibratore
35 % 0; ...                           % 3) delta_s_0. Deflessione stabilizzatore
36 % 0.5];                            % 4) delta_T_0. Manette
37
38 %%
39 % Dichiarazione variabili globali
40
41 global ...
42 g ...                                % accelerazione di gravità
43 z_0 V0 q0 gamma0 ...% condizioni iniziali
44 rho0 ...                                % densità dell'aria alla quota Z_0
45 delta_tab ...                          % deflessione aletta tab
46 delta_s ...                            % legge di comando stabilizzatore
47 delta_T ...                            % legge di comando manetta
48 delta_e_fix ...
49 myAC ...                                % oggetto aereo. (è una sorta di record con
50 % diversi campi)
51
52 % Aircraft data file name
53 aircraftDataFileName = 'DSV1.txt';
54
55 % Aircraft object
56 myAC = DSVaircraft112(aircraftDataFileName); % viene passato il nome del file
57
58 if (myAC.err == -1)          %parametro di controllo per vedere se vi è un
59 disp('... terminazione.');//errore.
60 else
61 disp(['File ', aircraftDataFileName, ' letto correttamente.']);
62
63 % Condizioni iniziali. Condizioni di primo tentativo per la ricerca delle
64 % condizioni di trim
65 x_0 = INex11_2(1);                  % posizione longitudinale [m]
66 z_0 = INex11_2(2);                  % a.s.l. (about sea level) altezza [m]
67 V0 = INex11_2(3);                   % vettore velocità [m/s]
68 q0 = INex11_2(4);                   % velocità angolare di pitch [rad/s]
69 gamma0 = INex11_2(5);                % angolo di salita [rad]
70
71 % Funzione matlab del modello ISA. Lo usiamo per il calcolo della densità
72 [air_Temp_0,sound_speed_0, air_pressure_0, rho0] = atmosisa(-z_0);
73
74 % Accelerazione di gravità
75 g = 9.81;                           % [m/s^2]
76
77 %% Input da DSV.AIRCRAFT

```

```

78
79 % Caratteristiche di massa
80 m = myAC.mass; %massa (kg)
81 k_y = myAC.k_y; %raggio di inerzia longitudinale (m)
82
83 % Caratteristiche geometriche
84 c = myAC.mac; % corda media aerodinamica (m)
85 b = myAC.b; % apertura alare (m)
86 S = myAC.S; % superficie alare (m^2)
87 Xcg_adim = myAC.Xcg_adim; % distanza adimensionale del baricentro dal l.e.
88 Xn_adim = myAC.Xn_adim; %distanza aidimensionale del punto neutro dal l.e.
89
90 % Caratteristiche propulsive
91 T0 = myAC.T/g; % spinta netta (kgf)
92 mu_T = myAC.mu_T; % calettamento della spinta
93 Cm_T = myAC.Cm_T_0; %contributo al coefficiente mom becch
94
95 % Caratteristiche aerodinamiche
96 CD_0 = myAC.CD_0;
97 K = myAC.K;
98 m = myAC.m;
99
100 CL_alpha = myAC.CL_alpha;
101 CL_delta_e = myAC.CL_delta_e;
102 CL_delta_s = myAC.CL_delta_s;
103 CL_alpha_dot = myAC.CL_alpha_dot;
104 CL_q = myAC.CL_q;
105
106 Cm_0 = myAC.Cm_0;
107 Cm_alpha = myAC.Cm_alpha;
108 Cm_delta_e = myAC.Cm_delta_e;
109 Cm_delta_s = myAC.Cm_delta_s;
110 Cm_alpha_dot = myAC.Cm_alpha_dot;
111 Cm_q = myAC.Cm_q;
112 eps_0 = myAC.eps_0;
113 DepsDalpha = myAC.DepsDalpha;
114
115 % Caratteristiche aerodinamiche dell'equilibratore
116 Ch_e_0 = myAC.Ch_e_0;
117 Ch_e_alpha = myAC.Ch_e_alpha;
118 Ch_e_delta_e = myAC.Ch_e_delta_e;
119 Ch_e_delta_s = myAC.Ch_e_delta_s;
120 Ch_e_delta_tab = myAC.Ch_e_delta_tab;
121 Ch_e_alpha_dot = myAC.Ch_e_alpha_dot;
122 Ch_e_q = myAC.Ch_e_q;
123 Ch_e_delta_e_dot = myAC.Ch_e_delta_e_dot;
124
125
126 % Vettore dei valori di primo tentativo
127 x0 = [ ...
128 INex11_2(6); ... % 1) alpha_0
129 INex11_2(7); ... % 2) delta_e_0. Deflessione equilibratore
130 INex11_2(8); ... % 3) delta_s_0. Deflessione stabilatore
131 INex11_2(9)]; % 4) delta_T_0. Manette
132
133 %% TRIM
134 %%%%%%% MINIMIZZAZIONE DELLA FUNZIONE DI COSTO %%%%%%%
135
136 % Limiti inferiori (low bound)
137 lb =[convang(-15,'deg','rad'),...%minimum alpha
138 convang(-20,'deg','rad'),...%minimum elevator deflection (delta_e)
139 convang(-5,'deg','rad'),...%minimum stabilizer incidence(delta_s)
140 0.2 ... % minimum thrust fraction (delta_T)
141 ];
142
143 % Limiti superiori (upper bound)
144 ub =[convang( 15,'deg','rad'), ... % maximum alpha
145 convang( 13,'deg','rad'), ... % maximum elevator deflection
146 convang( 5,'deg','rad'), ... % maximum stabilizer incidence
147 1.0 ... % maximum thrust fraction
148 ];
149
150 %Vincolo lineare per delta_s_0 (ipostizziamo sia nullo)
151 Aeq = zeros(4,4);
152 beq = zeros(4,1); Aeq(3,3) = 1;
153 delta_s_0 = convang(0,'deg','rad'); % Ipozziamo deflessione nulla dello stabilatore
154 beq(3,1) = delta_s_0; % manteniamo fisso delta_s
155
156 options = optimset( ...
157 'tolfun',1e-9, ... % tolleranza sull'errore (tolleration function)
158 'Algorithm','interior-point'); %tipologia di algoritmo da utilizzare
159
160 % Funzione per ricavare la condizione di trim
161 [x,fval] = ...
162 fmincon(@costLongEquilibriumStaticStickFixed112, ... % funzione per òa ricerca del minimo
163 x0, ... % valore di primo tentativo
164 [], ... % A, A*x=b
165 [], ... % b
166 Aeq, ... % Aeq, Aeq*x=beq

```

```

167 beq, ... % beq
168 lb,ub, ... % lower, upper bounds
169 @myNonLinearConstraint, ... % nonlinear const.
170 options); % opzioni di ricerca del minimo
171
172 alpha_0_rad = x(1);
173 alpha_0_deg = convang(x(1), 'rad', 'deg'); delta_e_0_rad = x(2);
174 delta_e_0_deg = convang(x(2), 'rad', 'deg'); delta_s_0_rad = x(3);
175 delta_s_0_deg = convang(x(3), 'rad', 'deg'); delta_T_0 = x(4);
176
177 theta_trim = gamma0+alpha_0_rad-myAC.mu_x;
178
179 % Angolo di Eulero iniziale.
180 % Serve per definire le condizioni iniziali. Si veda la fig. (7.8)
181 % Si osservi che alpha è l'angolo assoluto, non alpha body
182
183 % Mostro a display i risultati ottenuti
184 disp('Le condizioni di trim iniziali ottenute sono: ');
185 disp(['alpha_0 = ', num2str(alpha_0_deg), '']);
186 disp(['delta_e_0 = ', num2str(delta_e_0_deg), '']);
187 disp(['delta_s_0 = ', num2str(delta_s_0_deg), '']);
188 disp(['delta_T_0 = ', num2str(delta_T_0)]);
189 %%%%%%%%%%%%%% RISOLUZIONE EQUAZIONI DELLA DINAMICA %
190 %%%%%%%%%%%%%%
191 %%%%%%%%%%%%%% Risolto il problema del trim si passa alla risoluzione delle equazioni
192 % della dinamica per le quali si fa riferimento alle eq. (7.61)
193
194 % Istanti di inizio e fine osservazione del fenomeno
195 t_0 = 0; % istante iniziale
196 t_1 = t_switch; % istante finale volo a comandi bloccati
197
198 switch periodo
199 case 1
200     t_finale = 6; % istante finale di osservazione del fenomeno
201 case 2
202     t_finale = 50; % istante finale di osservazione del fenomeno
203 end
204
205
206 % Mesh
207 N = 200; % numero di punti
208 time1 = linspace(t_0, t_1, N); % punti mesh uniforme
209
210 % Leggi di comando
211 delta_e_fix = @(t) interp1([t_0 t_1], [delta_e_0_rad delta_e_0_rad], t);
212 delta_tab = @(t) interp1([t_0 t_finale], [0 0], t);
213 delta_s = @(t) interp1([t_0 t_finale], [delta_s_0_rad delta_s_0_rad], t);
214 delta_T = @(t) interp1([t_0 t_finale], [delta_T_0 delta_T_0], t);
215
216 %%%%%%%%%%%%%% CASO COMANDI BLOCCATI %
217 %%%%%%%%%%%%%%
218 % Vettore condizioni iniziali
219 X01 = [V0;
220 alpha_0_rad; q0;
221 x_0; z_0;
222 theta_trim];
223
224 % Risolvo il sistema di equazioni iniziali X0 mediante l'ode 45
225 options = odeset('RelTol', 1e-9, 'AbsTol', 1e-9*ones(1,6));
226
227 [vTime1, X1] = ode45(@eqLongDynamicStickFixed112, ... % eq. (7.61)
228                         time1, ... % Vettore dei tempi
229                         X01, ... % Vettore iniziale
230                         options); % Vettore delle opzioni
231
232 %%%%%%%%%%%%%% CASO COMANDI LIBERI %
233 %%%%%%%%%%%%%%
234 % Circa il vettore dei comandi liberi occorre aggiungere un'ulteriore
235 % incognita
236 % quale, appunto, la deflessione degli equilibratori. Inoltre occorre
237 % considerare come posizione iniziale quelle che riguardano le condizioni
238 % finali precedenti. Per cui X02 = X1 + altre due termini
239 % Istanti di inizio e fine osservazione del fenomeno
240
241 % Mesh
242 N = 200; % numero di punti
243 time2 = linspace(t_1, t_finale, N); % punti mesh uniforme
244
245 %Vettore condizioni iniziali che si ottiene a partire da quelle finalinel
246 %caso di comandi bloccati. Si badi che il vettore X1 ottenuto dalla
247 %funzione ode è un vettore ove ogni riga indica il valore del vettore di
248 %stato X1 ad un dato istante. Noi siamo interessati all'istante finale
249 %per cui consideriamo l'ultima riga.
250
251 X02 = [X1(end,:)];
252 O; % derivata prima deflessione equilibratore nullo
253 delta_e_fix(t_1)]; % deflessione iniziale pari a quella all'istante t_1

```

```

256
257 % Risolvo il sistema di equazioni iniziali X02 mediante l'ode 45
258 options = odeset( 'RelTol', 1e-9, 'AbsTol', 1e-9*ones(1,8));
259 [vTime2, X2] = ode45(@eqLongDynamicStickFree, ... % eq. (7.61)
260 time2, ... % Vettore dei tempi
261 X02', ... % Vettore iniziale
262 options); % Vettore delle opzioni
263
264 % Sommo i due vettori così ottenuti. Per far ciò, dato che X1 ha solo 6
265 % colonne dato che 6 sono le incognite, occorre aggiungere altre due
266 % riguardanti la derivata di delta_e e delta_e
267 X1(:,7) = zeros(numel(vTime1),1); % colonna di valori nulli. La funzione numel
268 % restituisce il numero di elementi di vTime1
269 X1(:,8) = delta_e_fix(vTime1); % aggiungo il vettore
270 %dei valori di delta_e fissi
271
272 time = [vTime1; vTime2(2:end)]; % sommo i due vettori tempo
273 X = [X1; X2(2:end, :)]; % sommo al vettore di stato X1 il vettore X2
274 % ma elimino la prima riga che coincide con
275 % quella finale del vettore X1
276
277 % Incrementi rispetto i valori iniziali
278 DELTA_V = X(:,1) - V0;
279 DELTA_alpha = convang(X(:,2), 'rad', 'deg') - alpha_0_deg;
280 DELTA_q = convangvel(X(:,3), 'rad/s', 'deg/s') - q0;
281 X_EG = X(:,4);
282 DELTA_h = -(X(:,5) - z_0);
283 THETA = X(:,6);
284 GAMMA = THETA + myAC.mu_x - X(:,2);
285 DELTA_E = X(:,8);
286
287 %%%%%%%%%%%%%% FATTORI DI CARICO %
288 %%%
289 %%%
290 %%%
291
292 %Per calcolare q_punto, f_xA e f_zA, occorre riprendere le equazioni 7.61
293 %e calcolare i vettori V_punto e gamma_punto.Si osservi che gamma_punto
294 %= theta_punto - alpha_punto (è sufficiente derivare l'equazione 7.64)
295 %eq. (7.61a)
296
297 V = X(:,1);
298 alpha = X(:,2);
299 q = X(:,3);
300 z_EG = X(:,5);
301 theta = X(:,6);
302 delta_e = X(:,8);
303
304
305 % Funzione matlab del modello ISA. Lo usiamo per il calcolo della densità
306 % Serve per aggiornare i parametri atmosferici dato che z_EG può variare
307 [air_Temp_0,sound_speed_0, air_pressure_0, rho] = atmosisa(-z_EG);
308 mu_rel = (myAC.W/g)./(rho*myAC.S*myAC.b); % m/(rho*S*b)
309
310
311 % Calcolo delle derivate mediante la funzione gradient
312 V_punto = gradient(X(:,1), time);
313 alpha_punto = gradient(X(:,2), time);
314 q_punto = gradient(X(:,3), time);
315
316
317 % eq. (7.61f)
318 theta_punto = q;
319 % Si ricavano, di seguito, le ulteriori funzioni di interesse, quali
320 % gamma_punto e le funzioni fattore di carico lungo gli assi aerodinamici
321 % xA e zA
322 gamma_punto = theta_punto - alpha_punto;
323 f_xa = -(sin(GAMMA)+V_punto/g); % fattore di carico lungo xa
324 f_za = cos(GAMMA) + (V.*gamma_punto)/g; % fattore di carico lungo za
325
326
327 %%%%%%%%%%%%%% MOMENTO DI CERNIERA %
328 %%%
329 %%%
330 %%%
331 vTime=[vTime1;vTime2(2:end)];
332 CH_A_e = myAC.Ch_e_0 + ...
333 myAC.Ch_e_alpha*((1-myAC.DepsDalpha).*(alpha-myAC.mu_x)- ...
334 myAC.eps_0+delta_s([vTime])+myAC.mu_x)+...
335 myAC.Ch_e_delta_e*delta_e+...
336 myAC.Ch_e_delta_s*delta_s([vTime])+...
337 myAC.Ch_e_delta_tab*delta_tab([vTime])+...
338 myAC.mac_e*myAC.Ch_e_q*q'/(2*V');
339
340 H_A_e = 0.5*rho'*V.^2*myAC.S_e*myAC.mac_e*CH_A_e;
341
342 %% Grafica 1
343
```

```

344 fig1=figure("NumberTitle","off","Name","Leggi di comando, angolo di deflessione, fattore di carico e
            accelerazione angolare")
345 tiledlayout(3,2,"TileSpacing","Tight");
346
347 nexttile
348 plot(time,delta_tab(time),'-k','LineWidth',1.5)
349 xlabel("t (s)"); ylabel("\delta_{tab,0} (deg)"); grid on;
350 label = {sprintf('t_1 = %g s', t_1)};
351 xline(t_1,'-b',label,'LineWidth',1);
352
353 nexttile
354 plot(time,convang(DELTA_E,'rad','deg'),'-k','LineWidth',1.5)
355 xlabel("t (s)"); ylabel("\delta_{E} (deg)"); grid on;
356 xl=xline(t_1,'-b',label,'LineWidth',1);
357 xl.LabelHorizontalAlignment = 'left';
358
359 nexttile
360 plot(time,delta_s(time),'-k','LineWidth',1.5)
361 xlabel("t (s)"); ylabel("\delta_{s,0} (deg)"); grid on;
362
363 xline(t_1,'-b',label,'LineWidth',1);
364
365 nexttile
366 plot(time,convang(q_punto,'rad','deg'),'-k','LineWidth',1.5)
367 xlabel("t (s)"); ylabel("$\dot{q} \cdot (deg/s^2)$", 'Interpreter','latex'); grid on;
368
369 xline(t_1,'-b',label,'LineWidth',1);
370
371 nexttile
372 plot(time,delta_T(time),'-k','LineWidth',1.5)
373 xlabel("t (s)"); ylabel("\delta_{T,0} (deg)"); grid on;
374
375 xline(t_1,'-b',label,'LineWidth',1);
376
377 nexttile
378 plot(time,f_xa,'-b',time,f_za,'-g','LineWidth',1.5)
379 xlabel("t (s)"); grid on;
380 xline(t_1,'-k',label,'LineWidth',1);
381 legend("f_{xA}","f_{zA}", '');
382
383
384
385 %% Grafica 2
386
387 fig2=figure("NumberTitle","off","Name","Storie temporali delle variabili di stato")
388
389 h(1) = subplot(3,2,1);
390 plot(time,X_EG,'-k','LineWidth',1.5);
391 grid on
392 xline(t_1,'-b',label,'LineWidth',1);
393
394 ylabel('$x_{EG} (m)$','interpreter','latex','fontsize',11)
395 h(2) = subplot(3,2,2);
396 plot(vTime,DELTA_h,'-k','LineWidth',1.5);
397 grid on
398 xline(t_1,'-b',label,'LineWidth',1);
399
400 ylabel('$\Delta h (m)$','interpreter','latex','fontsize',11)
401 h(3) = subplot(3,2,3);
402 plot(vTime,DELTA_V,'-k','LineWidth',1.5);
403 grid on
404 xline(t_1,'-b',label,'LineWidth',1);
405
406 ylabel('$\Delta V (m/s)$','interpreter','latex','fontsize',11)
407 h(4) = subplot(3,2,4);
408 plot(vTime,DELTA_alpha,'-k','LineWidth',1.5);
409 grid on
410 xline(t_1,'-b',label,'LineWidth',1);
411
412 ylabel('$\Delta \alpha (deg)$','interpreter','latex','fontsize',11)
413 h(5) = subplot(3,2,5);
414 plot(vTime,DELTA_q,'-k','LineWidth',1.5);
415 grid on
416 xline(t_1,'-b',label,'LineWidth',1);
417
418 xlabel('$t (s)$','interpreter','latex','fontsize',11);
419 ylabel('$\Delta q (deg/s)$','interpreter','latex','fontsize',11)
420 h(6) = subplot(3,2,6);
421 plot(vTime,convang(THETA,'rad','deg'),"--b",'LineWidth',1.5);
422 hold on;
423 plot(vTime,convang(GAMMA,'rad','deg'),'-g','LineWidth',1.5);
424 grid on
425 xline(t_1,'-k',label,'LineWidth',1);
426 lgd = legend('$\theta(t)$','$\gamma(t)$','');
427 lgd.Interpreter = 'latex';
428 lgd.FontSize = 11;
429
430 xlabel('$t (s)$','interpreter','latex','fontsize',11);

```

```

432 ylabel('$(deg)$','interpreter','latex','fontsize',11)
433 %%
434 fig3=figure("NumberTitle","off","Name","Storie temporali del momento di cerniera")
435 plot(vTime,H_A_e,'-k','LineWidth',.5);
436 grid on
437
438 xlabel('$t~(s)$','interpreter','latex','fontsize',11);
439 ylabel('$H_{\{A\_e\}}(N*m)$','interpreter','latex','fontsize',11)
440
441 diary off
442 saveas(fig1,"Fig11_2_1.eps",'epsc')
443 saveas(fig2,"Fig11_2_2.eps",'epsc')
444 saveas(fig3,"Fig11_2_3.eps",'epsc')
445
446 end
447 end

```

4.4.1 Risultati

Le condizioni di trim trovate sono:

Condizioni di Trim

Soluzione di un problema di trim ad una data altitudine e velocità di volo
File DSV1.txt aperto.
File DSV1.txt letto correttamente.

Le condizioni di trim iniziali ottenute sono:

$\alpha_0 = 2.019^\circ$
 $\delta_e = -4.3981^\circ$
 $\delta_s = 4.3398e-22^\circ$
 $\delta_T = 0.41716$

Le successive figure mostrano le storie temporali delle leggi di comando bloccate, la storia temporale dei fattori di carico, le storie temporali delle variabili di stato per comandi liberi, la storia temporale della deflessione dell'eqilibratore e del momento di cerniera.

La linea verticale indica l'istante in cui si passa da comandi bloccati a comandi liberi.

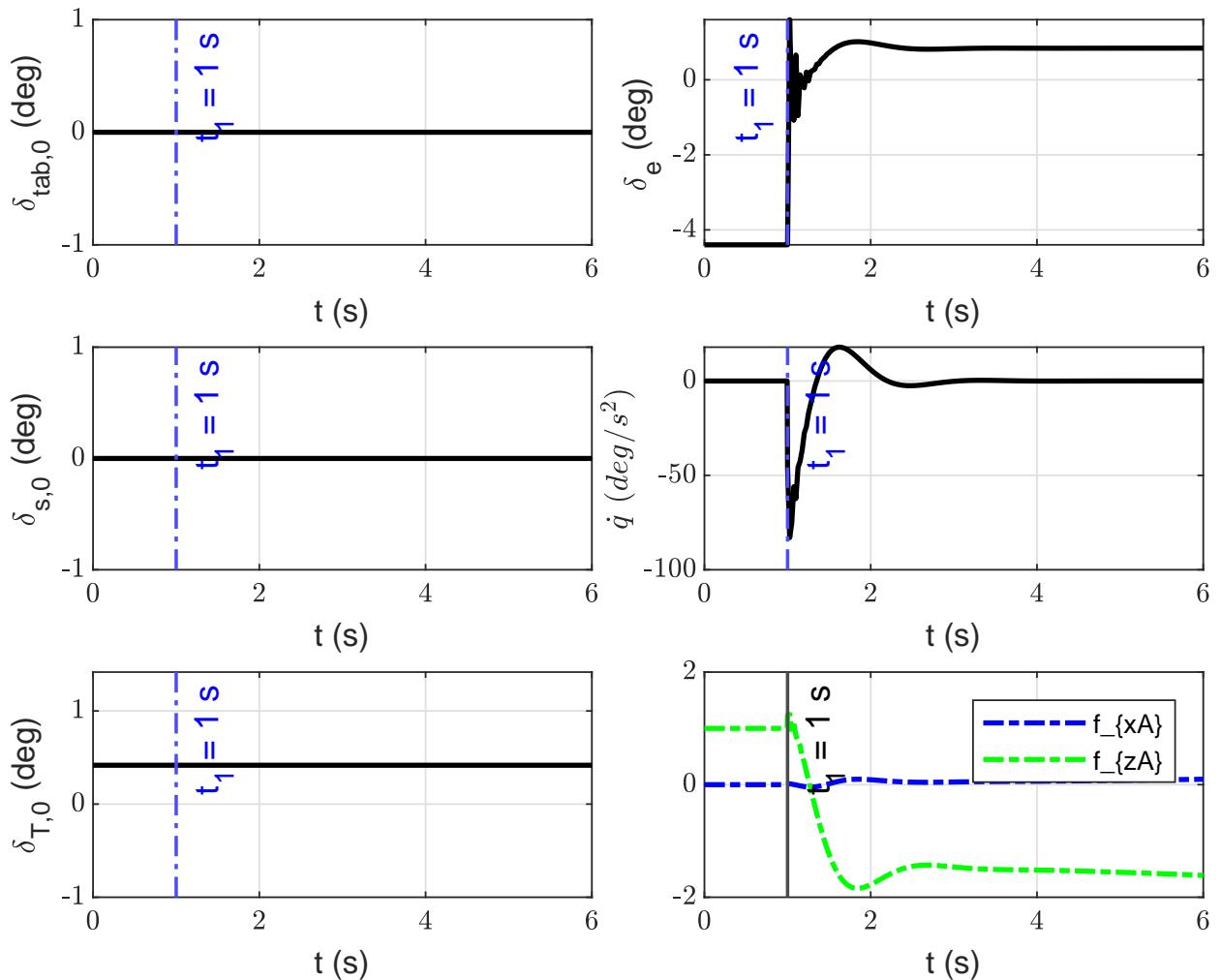


Figura 4.7: Esercizio 2 - Leggi di comando, angolo di deflessione, fattore di carico e accelerazione angolare

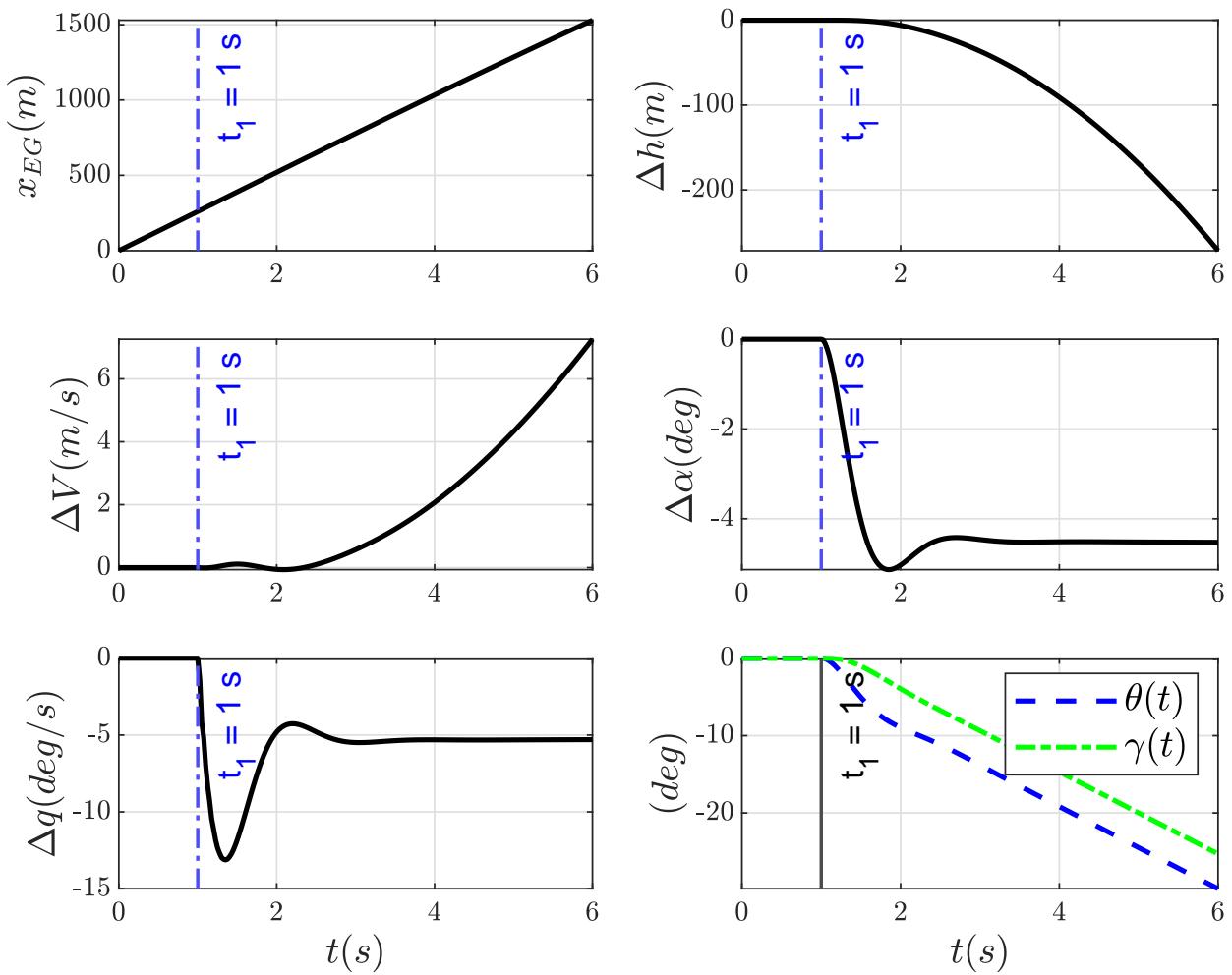


Figura 4.8: Esercizio 2 - Storie temporali delle variabili di stato

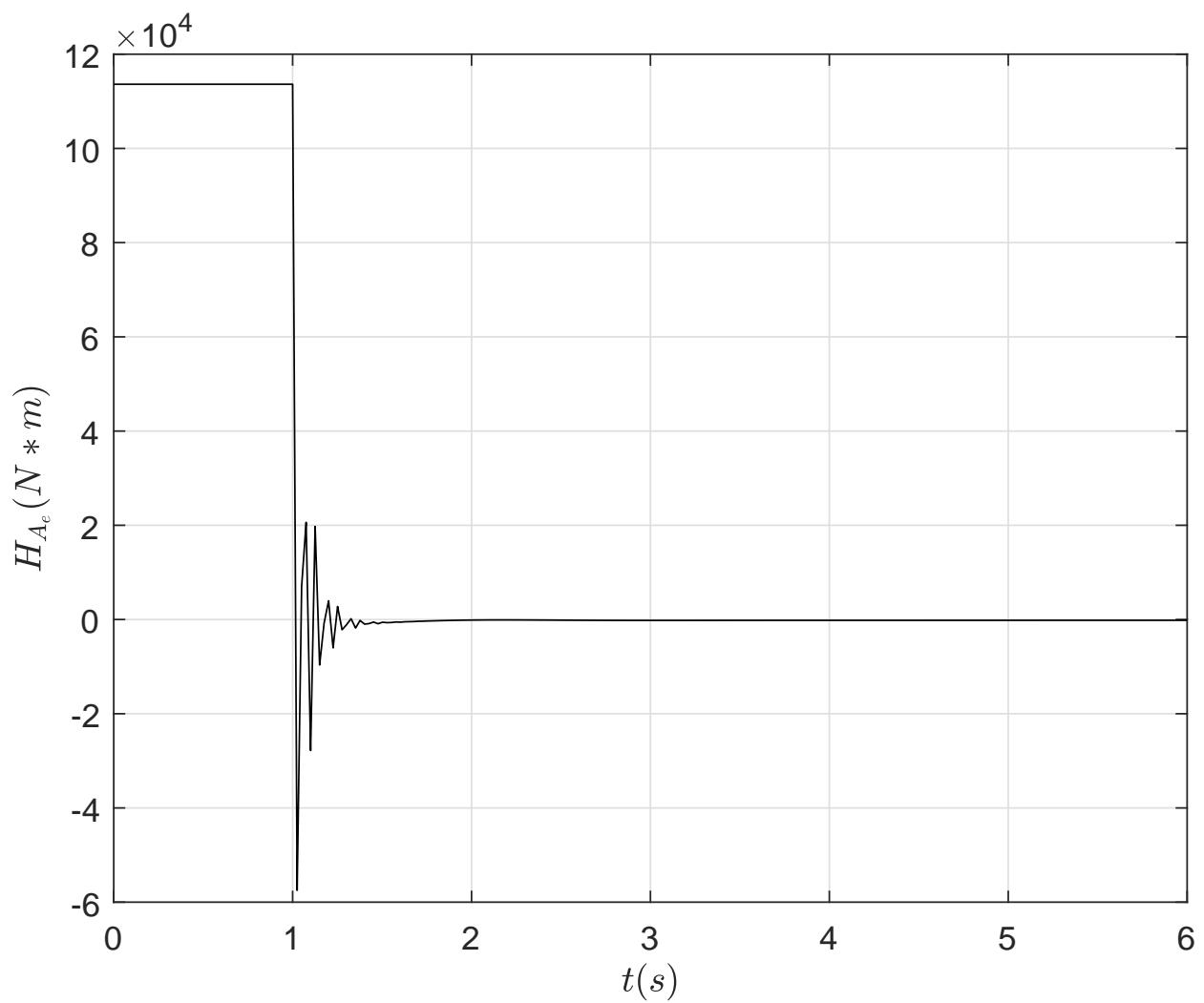


Figura 4.9: Esercizio 2 - Storie temporali del momento di cerniera

4.5 Esercizio 3 (11.3)

L'obiettivo di questo esercizio è determinare l'effetto della deflessione dell'aletta tab sull'angolo di flottaggio dell'equilibratore e di trovare la deflessione δ_t necessaria ad avere un angolo δ_e asintotico pari a $\delta_{e,0}$, cioè quello ottenuto imponendo le condizioni di trim. Nelle condizioni asintotiche agiscono sull'equilibratore solo il momento di cerniera aerodinamico e il momento dovuto all'eccentricità del baricentro rispetto all'asse di cerniera. Pertanto se si suppone che la condizione asintotica coincida con la condizione di trim iniziale, l'equazione di equilibrio alla rotazione diventa:

$$\begin{aligned} m_e(a_{G_{z_{B,0}}} - g_{z_{B,0}})e_e &= \mathcal{H}_{A,e} = \\ &= \bar{q}_\infty S_e \bar{c}_e [C_{H_0} + C_{H_\alpha} \alpha_H + C_{H_{\delta_e}} \delta_{e,0} + C_{H_{\delta_s}} \delta_s + C_{H_{\delta_t}} \delta_t^*] \end{aligned} \quad (4.38)$$

In cui $a_{G_{z_{B,0}}} - g_{z_{B,0}} = g \cos\theta$. Risolvendo questa equazione rispetto a δ_t^* si trova il valore ricercato, da cui:

$$C_{H_{\delta_t}} \delta_t^* = \frac{m_e(a_{G_{z_{B,0}}} - g_{z_{B,0}})e_e}{\bar{q}_\infty S_e \bar{c}_e} - [C_{H_0} + C_{H_\alpha} \alpha_H + C_{H_{\delta_e}} \delta_{e,0} + C_{H_{\delta_s}} \delta_s] \quad (4.39)$$

Dunque, dal codice si ottiene:

$$\delta_t^* = 0.037824 \text{ deg} \quad (4.40)$$

Il codice utilizzato è:

4.5.1 Risultati

Le condizioni di trim trovate sono:

Condizioni di Trim

Soluzione di un problema di trim ad una data altitudine e velocità di volo
File DSV1.txt aperto.
File DSV1.txt letto correttamente.

```
delta_tab_segnato = 0.037824°
Le condizioni di trim iniziali ottenute sono:
alpha_0 = 2.019°
delta_{e,0} = -4.3981°
delta_{s,0} = 4.3398e-22°
delta_{T,0} = 0.41716
```

Le successive figure mostrano le storie temporali delle leggi di comando bloccate, la storia temporale dei fattori di carico, le storie temporali delle variabili di stato per comandi liberi, la storia temporale della deflessione dell'eqilibratore e del momento di cerniera.

La linea verticale indica l'istante in cui si passa da comandi bloccati a comandi liberi.

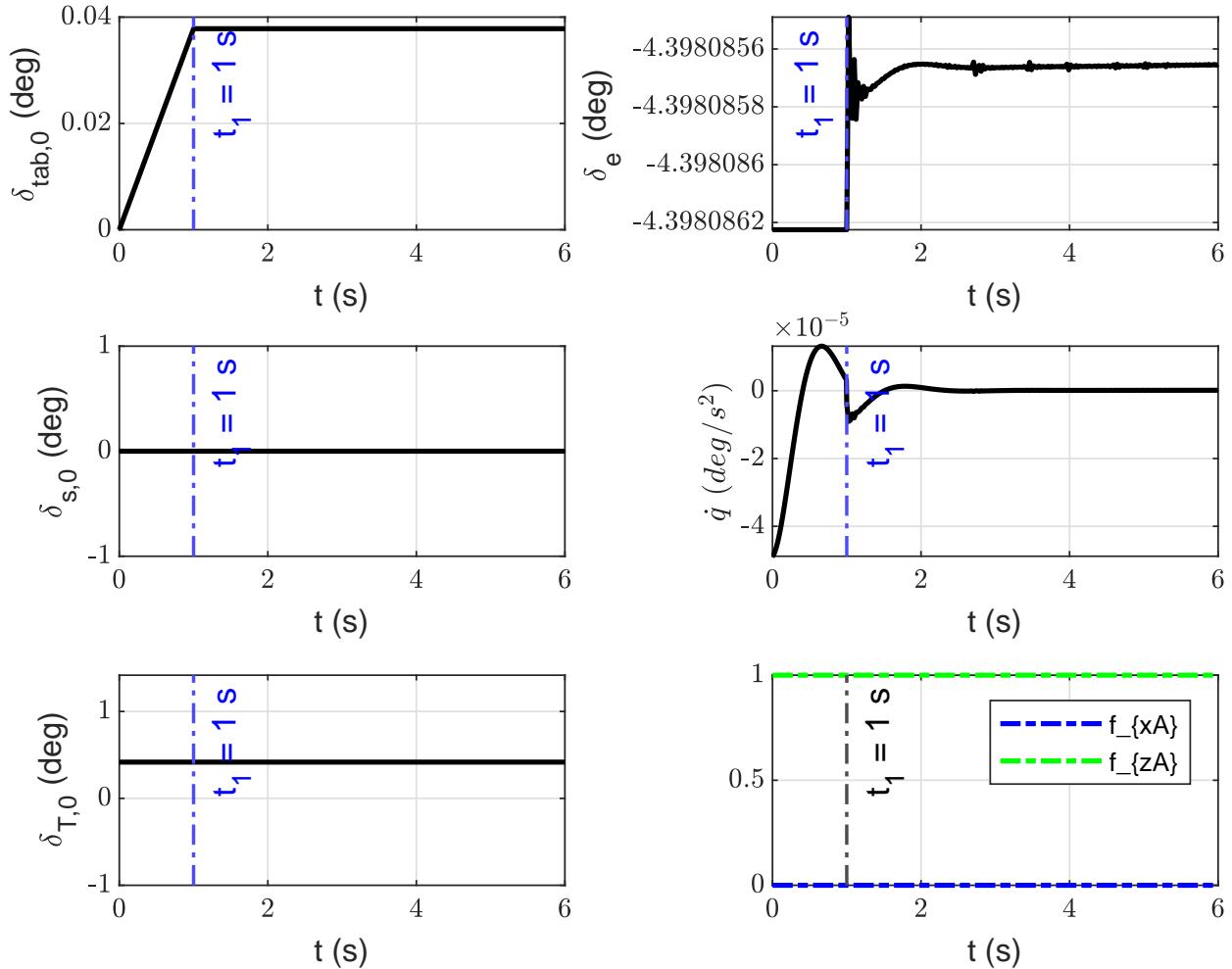


Figura 4.10: Esercizio 3 - Leggi di comando, angolo di deflessione, fattore di carico e accelerazione angolare

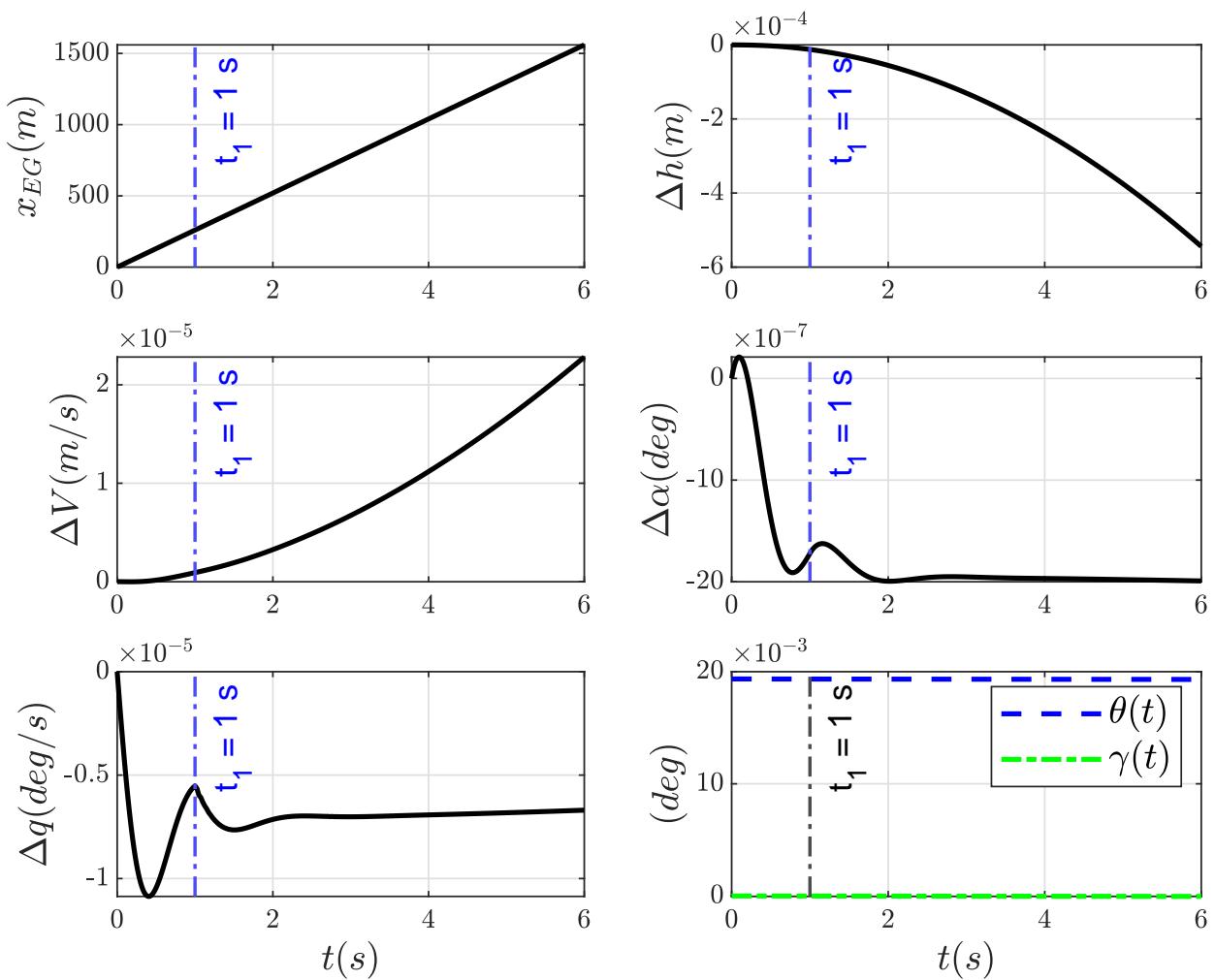


Figura 4.11: Esercizio 3 - Storie temporali delle variabili di stato

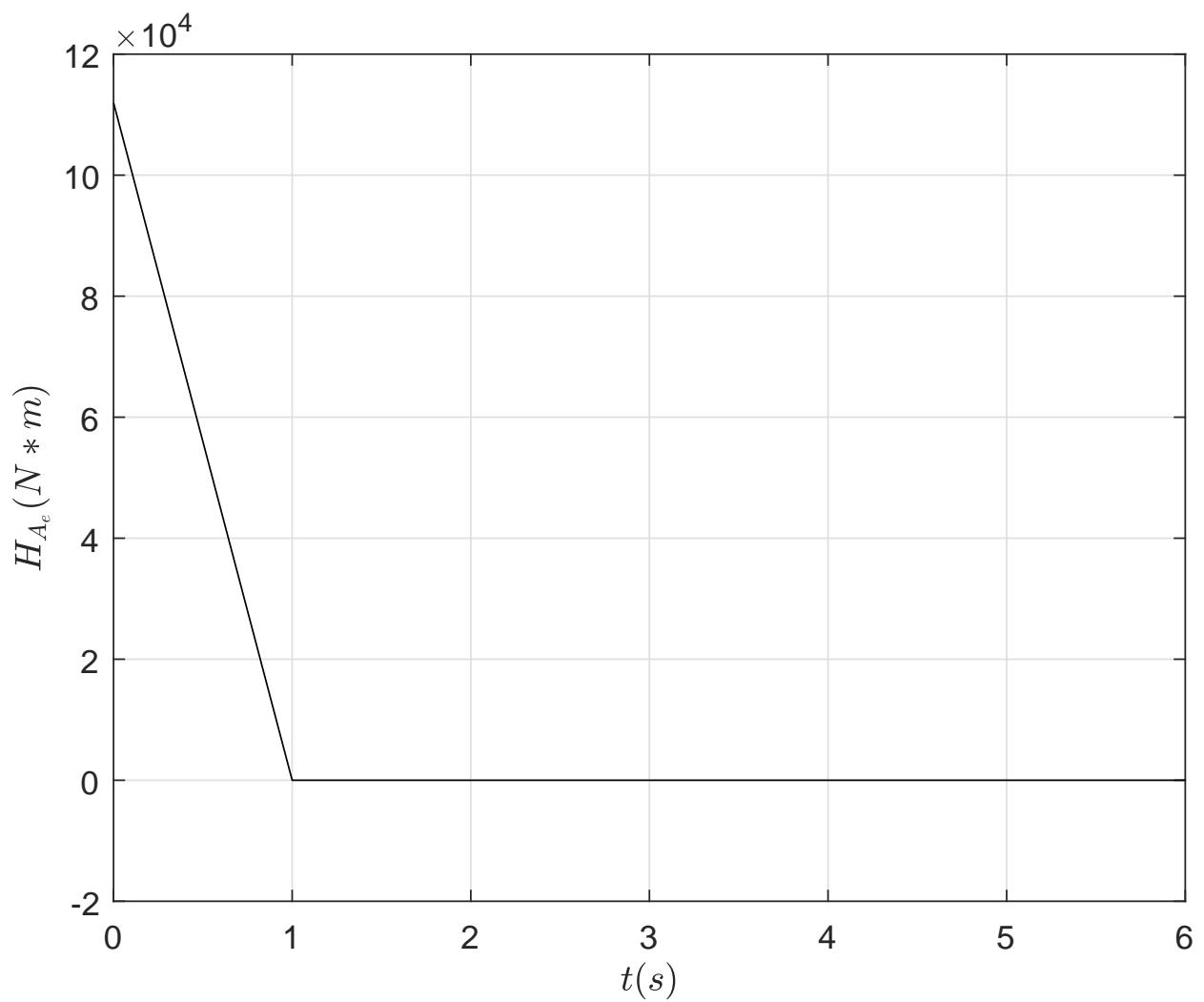


Figura 4.12: Esercizio 3 - Storie temporali del momento di cerniera

4.6 Esercizio 4 (11.4)

In questo esercizio si risolve il problema del volo longitudinal-simmetrico assegnando una legge del comando dell'equilibratore δ_e del tipo *cabra-picchia* (*elevator pulse*) comandata dal pilota. Ciò avviene al tempo t_1 . Nella fase successiva si suppone che il comando longitudinale venga rilasciato (comandi liberi). La storia dei comandi dell'equilibratore è in una delle prossime figure. Si suppone nullo e fissato l'angolo di deflessione dell'aletta tab. Il codice è:

```

1 function [] = func114(INex11_2)
2 % -----
3 % Esercizi Capitolo 11
4 % -----
5 % Esercizio N°4
6 % -----
7
8
9 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
10 %          CALCOLO CONDIZIONI DI TRIM %
11 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
12 diary('ex11_4.txt')
13 diary on
14
15 % Ricerca di un set completo di condizioni iniziali per un volo trimmato disp('Moto di un aeroplano a 3-DoF.');
16 disp('Soluzione di un problema di trim ad una data altitudine e velocità di volo');
17
18 %% INPUT
19 % Tipo di Periodo: 1=corto (6s)      2=lungo (50s)
20
21 INex11_1 = INex11_2;
22
23 periodo = INex11_1(10);
24 t_switch = 1;
25 % x_0 = ;                         % posizione longitudinale [m]
26 % z_0 = -4000.0;                   % a.s.l. (about sea level) altezza [m]
27 % VO = 260;                       % vettore velocità [m/s]
28 % q0 = 0.0;                        % velocità angolare di pitch [rad/s]
29 % gamma0 = 0.0;                    % angolo di salita [rad]
30
31 % % Vettore dei valori di primo tentativo
32 % x0 = [ ...
33 % 0; ...                           % 1) alpha_0
34 % 0; ...                           % 2) delta_e_0. Deflessione equilibratore
35 % 0; ...                           % 3) delta_s_0. Deflessione stabilizzatore
36 % 0.5];                            % 4) delta_T_0. Manette
37
38 %%
39 % Dichiarazione variabili globali
40
41 global ...
42 g ...                           % accelerazione di gravità
43 z_0 VO q0 gamma0 ... % condizioni iniziali
44 rho0 ...                         % densità dell'aria alla quota Z_0
45 delta_tab ...                     % deflessione aletta tab
46 delta_s ...                       % legge di comando stabilizzatore
47 delta_T ...                       % legge di comando manetta
48 delta_e_fix ...                  % oggetto aereo. (è una sorta di record con
49 myAC                                % diversi campi)
50
51
52 % Aircraft data file name
53 aircraftDataFileName = 'DSV1.txt';
54
55 % Aircraft object
56 myAC = DSVAircraft112(aircraftDataFileName); % viene passato il nome del file
57
58 if (myAC.err == -1)           %parametro di controllo per vedere se vi è un
59 disp('... terminazione.');//errore.
60 else
61 disp(['File ', aircraftDataFileName, ' letto correttamente.']);
62
63 % Condizioni iniziali. Condizioni di primo tentativo per la ricerca delle
64 % condizioni di trim
65 x_0 = INex11_1(1);             % posizione longitudinale [m]
66 z_0 = INex11_1(2);             % a.s.l. (about sea level) altezza [m]
67 VO = INex11_1(3);              % vettore velocità [m/s]
68 q0 = INex11_1(4);              % velocità angolare di pitch [rad/s]
69 gamma0 = INex11_1(5);          % angolo di salita [rad]
70
71 % Funzione matlab del modello ISA. Lo usiamo per il calcolo della densità
72 [air_Temp_0,sound_speed_0, air_pressure_0, rho0] = atmosisa(-z_0);
73
74 % Accelerazione di gravità
75 g = 9.81;                      % [m/s^2]
76
77 %% Input da DSV.AIRCRAFT
78
```

```

79 % Caratteristiche di massa
80 m = myAC.mass; %massa (kg)
81 k_y = myAC.k_y; %raggio di inerzia longitudinale (m)
82
83 % Caratteristiche geometriche
84 c = myAC.mac; % corda media aerodinamica (m)
85 b = myAC.b; % apertura alare (m)
86 S = myAC.S; % superficie alare (m^2)
87 Xcg_adim = myAC.Xcg_adim; % distanza adimensionale del baricentro dal l.e.
88 Xn_adim = myAC.Xn_adim; %distanza adimensionale del punto neutro dal l.e.
89
90 % Caratteristiche propulsive
91 T0 = myAC.T/g; % spinta netta (kgf)
92 mu_T = myAC.mu_T; % calettamento della spinta
93 Cm_T = myAC.Cm_T_0; %contributo al coefficiente mom becch
94
95 % Caratteristiche aerodinamiche
96 CD_0 = myAC.CD_0;
97 K = myAC.K;
98 m = myAC.m;
99
100 CL_alpha = myAC.CL_alpha;
101 CL_delta_e = myAC.CL_delta_e;
102 CL_delta_s = myAC.CL_delta_s;
103 CL_alpha_dot = myAC.CL_alpha_dot;
104 CL_q = myAC.CL_q;
105
106 Cm_0 = myAC.Cm_0;
107 Cm_alpha = myAC.Cm_alpha;
108 Cm_delta_e = myAC.Cm_delta_e;
109 Cm_delta_s = myAC.Cm_delta_s;
110 Cm_alpha_dot = myAC.Cm_alpha_dot;
111 Cm_q = myAC.Cm_q;
112 eps_0 = myAC.eps_0;
113 DepsDalpha = myAC.DepsDalpha;
114
115 % Caratteristiche aerodinamiche dell'equilibratore
116 Ch_e_0 = myAC.Ch_e_0;
117 Ch_e_alpha = myAC.Ch_e_alpha;
118 Ch_e_delta_e = myAC.Ch_e_delta_e;
119 Ch_e_delta_s = myAC.Ch_e_delta_s;
120 Ch_e_delta_tab = myAC.Ch_e_delta_tab;
121 Ch_e_alpha_dot = myAC.Ch_e_alpha_dot;
122 Ch_e_q = myAC.Ch_e_q;
123 Ch_e_delta_e_dot = myAC.Ch_e_delta_e_dot;
124
125
126 % Vettore dei valori di primo tentativo
127 x0 = [ ...
128 INex11_1(6); ... % 1) alpha_0
129 INex11_1(7); ... % 2) delta_e_0. Deflessione equilibratore
130 INex11_1(8); ... % 3) delta_s_0. Deflessione stabilatore
131 INex11_1(9)]; % 4) delta_T_0. Manette
132
133 %% TRIM
134 %%%%%% MINIMIZZAZIONE DELLA FUNZIONE DI COSTO %%%%%%
135
136 % Limiti inferiori (low bound)
137 lb =[convang(-15,'deg','rad'),...%minimum alpha
138 convang(-20,'deg','rad'),...%minimum elevator deflection (delta_e)
139 convang(-5,'deg','rad'),...%minimum stabilizer incidence(delta_s)
140 0.2 ... % minimum thrust fraction (delta_T)
141 ];
142
143 % Limiti superiori (upper bound)
144 ub =[convang( 15,'deg','rad'), ... % maximum alpha
145 convang( 13,'deg','rad'), ... % maximum elevator deflection
146 convang( 5,'deg','rad'), ... % maximum stabilizer incidence
147 1.0 ... % maximum thrust fraction
148 ];
149
150 %Vincolo lineare per delta_s_0 (ipostizziamo sia nullo)
151 Aeq = zeros(4,4);
152 beq = zeros(4,1); Aeq(3,3) = 1;
153 delta_s_0 = convang(0,'deg','rad'); % Ipozziamo deflessione nulla dello stabilatore
154 beq(3,1) = delta_s_0; % manteniamo fisso delta_s
155
156 options = optimset( ...
157 'tolfun',1e-9, ... % tolleranza sull'errore (tolleration function)
158 'Algorithm','interior-point'); %tipologia di algoritmo da utilizzare
159
160 % Funzione per ricavare la condizione di trim
161 [x,fval] = ...
162 fmincon(@costLongEquilibriumStaticStickFixed, ... % funzione per òa ricerca del minimo
163 x0, ... % valore di primo tentativo
164 [], ... % A, A*x=b
165 [], ... % b
166 Aeq, ... % Aeq, Aeq*x=beq
167 beq, ... % beq
```

```

168 lb,ub, ... % lower, upper bounds
169 @myNonLinearConstraint, ... % nonlinear const.
170 options); % opzioni di ricerca del minimo
171
172 alpha_0_rad = x(1);
173 alpha_0_deg = convang(x(1),'rad','deg'); delta_e_0_rad = x(2);
174 delta_e_0_deg = convang(x(2),'rad','deg'); delta_s_0_rad = x(3);
175 delta_s_0_deg = convang(x(3),'rad','deg'); delta_T_0 = x(4);
176
177 %% calcolo trim tab
178 theta_trim = gamma0+alpha_0_rad-myAC.mu_x;
179
180 % Definizione angoli di interesse
181 alpha_body_0 = alpha_0_rad-myAC.mu_x; % alpha body
182 alpha_H_0 = (1-myAC.DepsDalpa)*(alpha_body_0)... % alpha H
183 -myAC.eps_0 + delta_s_0_rad + myAC.mu_x;
184 % Poichè ci troviamo in condizioni di trim non sono presenti termini
185 % variabili nel tempo
186
187 CH_A_e_delta_tab = ((myAC.mass_e*myAC.ec_adim*myAC.mac_e)...
188 *g*cos(theta_trim))/(0.5*rho0*V0^2*myAC.S_e*myAC.mac_e)...
189 - (myAC.Ch_e_0 + myAC.Ch_e_alpha*alpha_H_0 ...
190 + myAC.Ch_e_delta_e*delta_e_0_rad ...
191 + myAC.Ch_e_delta_s*delta_s_0_rad);
192 delta_tab_segnato = CH_A_e_delta_tab/myAC.Ch_e_delta_tab;
193
194
195
196
197 % Mostro a display i risultati ottenuti
198 disp('Le condizioni di trim iniziali ottenute sono: ');
199 disp(['alpha_0 = ', num2str(alpha_0_deg), '°']);
200 disp(['delta_{e,0} = ', num2str(delta_e_0_deg), '°']);
201 disp(['delta_{s,0} = ', num2str(delta_s_0_deg), '°']);
202 disp(['delta_{T,0} = ', num2str(delta_T_0)]);
203 %%%%%% RISOLUZIONE EQUAZIONI DELLA DINAMICA %
204 %%%%%% %%%%%% %%%%%% %%%%%% %%%%%% %%%%%%
205 % Risolto il problema del trim si passa alla risoluzione delle equazioni
206 % della dinamica per le quali si fa riferimento alle eq. (7.61)
207
208
209 % Istanti di inizio e fine osservazione del fenomeno
210 t_0 = 0; % istante iniziale
211 t_1 = t_switch; % istante finale volo a comandi bloccati
212 t_2 = t_switch+1.5;
213 t_3 = t_2+1.5;
214 switch periodo
215 case 1
216 t_finale = 10; % istante finale di osservazione del fenomeno
217 case 2
218 t_finale = 50; % istante finale di osservazione del fenomeno
219 end
220
221 % Mesh
222 N = 200; % numero di punti
223 time1 = linspace (t_0, t_1, N); % punti mesh uniforme
224
225 % Leggi di comando
226 delta_delta_e = convang(-3,'deg','rad');
227
228 delta_e_fix = @(t) interp1([t_0 t_1 t_2 t_3],[delta_e_0_rad delta_e_0_rad (delta_e_0_rad+delta_delta_e) (
    delta_e_0_rad)], t);
229
230 delta_s = @(t) interp1([t_0 t_finale],[delta_s_0_rad delta_s_0_rad],t);
231 delta_T = @(t) interp1 ([t_0 t_finale], [delta_T_0 delta_T_0], t);
232
233 % legge della deflessione 'dellaletta tab
234 % delta_tab = @(t) interp1 ([t_0, t_1, t_finale], ...
235 % [0, delta_tab_segnato, delta_tab_segnato],...
236 % t, 'linear');
237 delta_tab = @(t) interp1 ([t_0 t_finale], [0 0], t);
238 %%%%%% CASO COMANDI BLOCCATI %
239 % Vettore condizioni iniziali
240 X01 = [ V0;
241 alpha_0_rad; q0;
242 x_0; z_0;
243 theta_trim];
244
245 % Risolvo il sistema di equazioni iniziali X0 mediante l'ode 45
246 options = odeset( 'RelTol', 1e-9, 'AbsTol',1e-9*ones(1,6));
247
248 [vTime1, X1] = ode45(@eqLongDynamicStickFixed114, ... % eq. (7.61)
249 time1, ... % Vettore dei tempi
250 X01', ... % Vettore iniziale
251 options); % Vettore delle opzioni

```

```

256 % Mesh
257 N = 200; % numero di punti
258 time2 = linspace (t_1, t_2, N); % punti mesh uniforme
259
260 X02 = X1(end,:); 1
261
262 % Risolvo il sistema di equazioni iniziali X0 mediante l'ode 45
263 options = odeset( 'RelTol', 1e-9, 'AbsTol',1e-9*ones(1,6));
264
265 [vTime2, X2] = ode45(@eqLongDynamicStickFixed114, ... % eq. (7.61)
266 time2, ... % Vettore dei tempi
267 X02', ... % Vettore iniziale
268 options); % Vettore delle opzioni
269
270 % Mesh
271 N = 200; % numero di punti
272 time3 = linspace (t_2, t_3, N); % punti mesh uniforme
273
274 X03 = X2(end,:);
275
276 % Risolvo il sistema di equazioni iniziali X0 mediante l'ode 45
277 options = odeset( 'RelTol', 1e-9, 'AbsTol',1e-9*ones(1,6));
278
279 [vTime3, X3] = ode45(@eqLongDynamicStickFixed114, ... % eq. (7.61)
280 time3, ... % Vettore dei tempi
281 X03', ... % Vettore iniziale
282 options); % Vettore delle opzioni
283
284
285 %%%%%% CASO COMANDI LIBERI %%%%%%
286 % Circa il vettore dei comandi liberi occorre aggiungere un'ulteriore
287 % incognita
288 % quale, appunto, la deflessione degli equilibratori. Inoltre occorre
289 % considerare come posizione iniziale quelle che riguardano le condizioni
290 % finali precedenti. Per cui X02 = X1 + altre due termini
291 % Istanti di inizio e fine osservazione del fenomeno
292
293 % Mesh
294 N = 200; % numero di punti
295 time4 = linspace (t_3, t_finale, N); % punti mesh uniforme
296
297 %Vettore condizioni iniziali che si ottiene a partire da quelle finalinel
298 %caso di comandi bloccati. Si badi che il vettore X1 ottenuto dalla
299 %funzione ode è un vettore ove ogni riga indica il valore del vettore di
300 %stato X1 ad un dato istante. Noi siamo interessati all'istante finale
301 %per cui consideriamo l'ultima riga.
302
303 X04 = [X3(end,:)];
304 O; % derivata prima deflessione equilibratore nullo
305 delta_e_fix(t_3)]; % deflessione iniziale pari a quella all'istante t_1
306
307 % Risolvo il sistema di equazioni iniziali X02 mediante l'ode 45
308 options = odeset( 'RelTol', 1e-9, 'AbsTol',1e-9*ones(1,8));
309 [vTime4, X4] = ode45(@eqLongDynamicStickFree, ... % eq. (7.61)
310 time4, ... % Vettore dei tempi
311 X04', ... % Vettore iniziale
312 options); % Vettore delle opzioni
313
314 % Sommo i due vettori così ottenuti. Per far ciò, dato che X1 ha solo 6
315 % colonne dato che 6 sono le incognite, occorre aggiungere altre due
316 % riguardanti la derivata di delta_e e delta_e
317 X1(:,7) = zeros(numel(vTime1),1); % colonna di valori nulli. La funzione numel
318 % restituisce il numero di elementi di vTime1
319 X1(:,8) = delta_e_fix(vTime1); % aggiungo il vettore
320 %dei valori di delta_e fissi
321 X2(:,7) = zeros(numel(vTime2),1); % colonna di valori nulli. La funzione numel
322 % restituisce il numero di elementi di vTime1
323 X2(:,8) = delta_e_fix(vTime2); % aggiungo il vettore
324 %dei valori di delta_e fissi
325 X3(:,7) = zeros(numel(vTime3),1); % colonna di valori nulli. La funzione numel
326 % restituisce il numero di elementi di vTime1
327 X3(:,8) = delta_e_fix(vTime3); % aggiungo il vettore
328 %dei valori di delta_e fissi
329
330
331
332 time = [vTime1; vTime2(2:end); vTime3(2:end); vTime4(2:end)]; % sommo i due vettori tempo
333 X = [X1; X2(2:end, :); X3(2:end, :); X4(2:end, :)]; % sommo al vettore di stato X1 il vettore X2
334 % ma elimino la prima riga che coincide con
335 % quella finale del vettore X1
336
337
338 % Incrementi rispetto i valori iniziali
339 DELTA_V = X(:,1) - V0;
340 DELTA_alpha = convang (X(:,2), 'rad', 'deg') - alpha_0_deg;
341 DELTA_q = convangvel (X(:,3), 'rad/s', 'deg/s') - q0;
342 X_EG = X(:,4);
343 DELTA_h = -(X(:,5) - z_0);
344 THETA = X(:,6);

```

```

345 GAMMA      = THETA + myAC.mu_x - X(:,2);
346 DELTA_E    = X(:,8);
347
348
349 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
350 %          FATTORI DI CARICO          %
351 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
352
353 %Per calcolare q_punto, f_xA e f_zA, occorre riprendere le equazioni 7.61
354 %e calcolare i vettori V_punto e gamma_punto.Si osservi che gamma_punto
355 %= theta_punto - alpha_punto (è sufficiente derivare l'equazione 7.64)
356 %eq. (7.61a)
357
358 V      = X(:,1);
359 alpha  = X(:,2);
360 q      = X(:,3);
361 z_EG   = X(:,5);
362 theta  = X(:,6);
363 delta_e = X(:,8);
364
365
366 % Funzione matlab del modello ISA. Lo usiamo per il calcolo della densità
367 % Serve per aggiornare i parametri atmosferici dato che z_EG può variare
368 [air_Temp_0,sound_speed_0, air_pressure_0, rho] = atmosisa(-z_EG);
369 mu_rel = (myAC.W/g)./(rho*myAC.S*myAC.b); % m/(rho*S*b)
370
371
372 % Calcolo delle derivate mediante la funzione gradient
373 V_punto = gradient(X(:,1), time);
374 alpha_punto = gradient(X(:,2), time);
375 q_punto = gradient(X(:,3), time);
376
377
378 % eq. (7.61f)
379 theta_punto = q;
380 % Si ricavano, di seguito, le ulteriori funzioni di interesse, quali
381 % gamma_punto e le funzioni fattore di carico lungo gli assi aerodinamici
382 % xA e zA
383 gamma_punto = theta_punto - alpha_punto;
384 f_xa = -(sin(GAMMA)+V_punto/g);           % fattore di carico lungo xa
385 f_zz = cos(GAMMA) + (V.*gamma_punto)/g;   % fattore di carico lungo za
386
387
388
389 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
390 %          MOMENTO DI CERNIERA          %
391 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
392 vTime=time;
393 CH_A_e = myAC.Ch_e_0 + ...
394 myAC.Ch_e_alpha*((1-myAC.DepsDalpha).* (alpha-myAC.mu_x)- ...
395 myAC.eps_0+delta_s([vTime])+myAC.mu_x)+...
396 myAC.Ch_e_delta_e*delta_e+...
397 myAC.Ch_e_delta_s*delta_s([vTime])+...
398 myAC.Ch_e_delta_tab*delta_tab([vTime])+...
399 myAC.mac_e*myAC.Ch_e_q*q'/(2*V);
400
401 H_A_e = 0.5*rho'*V.^2*myAC.S_e*myAC.mac_e*CH_A_e;
402
403 %% Grafica 1
404
405 figure("NumberTitle","off","Name","Leggi di comando, angolo di deflessione, fattore di carico e accelerazione
        angolare")
406 tiledlayout(3,2,"TileSpacing","Tight");
407
408 nexttile
409 plot(time,delta_tab(time),'-k','LineWidth',1.5)
410 xlabel("t (s)"); ylabel("\delta_{tab,0} (deg)"); grid on;
411 label = {sprintf('t_1 = %g s', t_1)};
412 label2 = {sprintf('t_2 = %g s', t_2)};
413 label3 = {sprintf('t_3 = %g s', t_3)};
414
415 xline(t_1,'-b',label,'LineWidth',1);
416 xline(t_2,'-r',label2,'LineWidth',1);
417 xline(t_3,'-g',label3,'LineWidth',1);
418
419 nexttile
420 plot(time,convang(DELTA_E,'rad','deg'),'-k','LineWidth',1.5)
421 xlabel("t (s)"); ylabel("\delta_{e} (deg)"); grid on;
422 xl=xline(t_1,'-b',label,'LineWidth',1);
423 xl.LabelHorizontalAlignment = 'left';
424 xline(t_2,'-r',label2,'LineWidth',1);
425 xline(t_3,'-g',label3,'LineWidth',1);
426
427 nexttile
428 plot(time,delta_s(time),'-k','LineWidth',1.5)
429 xlabel("t (s)"); ylabel("\delta_{s,0} (deg)"); grid on;
430
431 xline(t_1,'-b',label,'LineWidth',1);
432 xline(t_2,'-r',label2,'LineWidth',1);

```

```

433 xline(t_3,'-.g',label3,'LineWidth',1);
434
435 nexttile
436 plot(time,convang(q_punto,'rad','deg'),'-k','LineWidth',1.5)
437 xlabel("t (s)"); ylabel("$\dot{q} \cdot (\deg/s^2)$", 'Interpreter', 'latex'); grid on;
438
439 xline(t_1,'-.b',label,'LineWidth',1);
440 xline(t_2,'-.r',label2,'LineWidth',1);
441 xline(t_3,'-.g',label3,'LineWidth',1);
442
443 nexttile
444 plot(time,delta_T(time),'-k','LineWidth',1.5)
445 xlabel("t (s)"); ylabel("\delta_{T,0} (deg)"); grid on;
446
447 xline(t_1,'-.b',label,'LineWidth',1);
448 xline(t_2,'-.r',label2,'LineWidth',1);
449 xline(t_3,'-.g',label3,'LineWidth',1);
450
451 nexttile
452 plot(time,f_xa,'-.b',time,f_za,'-.g','LineWidth',1.5)
453 xlabel("t (s)"); grid on;
454 xline(t_1,'-.k',label,'LineWidth',1);
455 xline(t_2,'-.r',label2,'LineWidth',1);
456 xline(t_3,'-.m',label3,'LineWidth',1);
457 legend("f_{xA}", "f_{zA}", '');
458
459
460
461 %% Grafica 2
462
463 figure("NumberTitle","off","Name","Storie temporali delle variabili di stato")
464
465 h(1) = subplot(3,2,1);
466 plot(time,X_EG,'-k','LineWidth',1.5);
467 grid on
468 xline(t_1,'-.b',label,'LineWidth',1);
469 xline(t_2,'-.r',label2,'LineWidth',1);
470 xline(t_3,'-.g',label3,'LineWidth',1);
471
472 ylabel('$x_{EG} (m)$', 'interpreter', 'latex', 'fontsize', 11)
473 h(2) = subplot(3,2,2);
474 plot(vTime,DELTA_h,'-k','LineWidth',1.5);
475 grid on
476 xline(t_1,'-.b',label,'LineWidth',1);
477 xline(t_2,'-.r',label2,'LineWidth',1);
478 xline(t_3,'-.g',label3,'LineWidth',1);
479
480 ylabel('$\Delta h (m)$', 'interpreter', 'latex', 'fontsize', 11)
481 h(3) = subplot(3,2,3);
482 plot(vTime,DELTA_V,'-k','LineWidth',1.5);
483 grid on
484 xline(t_1,'-.b',label,'LineWidth',1);
485 xline(t_2,'-.r',label2,'LineWidth',1);
486 xline(t_3,'-.g',label3,'LineWidth',1);
487
488 ylabel('$\Delta V (m/s)$', 'interpreter', 'latex', 'fontsize', 11)
489 h(4) = subplot(3,2,4);
490 plot(vTime,DELTA_alpha,'-k','LineWidth',1.5);
491 grid on
492 xline(t_1,'-.b',label,'LineWidth',1);
493 xline(t_2,'-.r',label2,'LineWidth',1);
494 xline(t_3,'-.g',label3,'LineWidth',1);
495
496 ylabel('$\Delta \alpha (deg)$', 'interpreter', 'latex', 'fontsize', 11)
497 h(5) = subplot(3,2,5);
498 plot(vTime,DELTA_q,'-k','LineWidth',1.5);
499 grid on
500 xline(t_1,'-.b',label,'LineWidth',1);
501 xline(t_2,'-.r',label2,'LineWidth',1);
502 xline(t_3,'-.g',label3,'LineWidth',1);
503
504 xlabel('$t (s)$', 'interpreter', 'latex', 'fontsize', 11);
505 ylabel('$\Delta q (deg/s)$', 'interpreter', 'latex', 'fontsize', 11)
506 h(6) = subplot(3,2,6);
507 plot(vTime,convang(THETA,'rad','deg'),"--b",'LineWidth',1.5);
508 hold on;
509 plot(vTime,convang(GAMMA,'rad','deg'),'-g','LineWidth',1.5);
510 grid on
511 xline(t_1,'-.k',label,'LineWidth',1);
512 xline(t_2,'-.r',label2,'LineWidth',1);
513 xline(t_3,'-.m',label3,'LineWidth',1);
514 lgd = legend('$\theta(t)$', '$\gamma(t)$', '');
515 lgd.Interpreter = 'latex';
516 lgd.FontSize = 11;
517
518 xlabel('$t (s)$', 'interpreter', 'latex', 'fontsize', 11);
519
520 ylabel('$(deg)$', 'interpreter', 'latex', 'fontsize', 11)

```

```

522 %%
523 figure("NumberTitle","off","Name","Storie temporali del momento di cerniera")
524 plot(vTime,H_A_e,'-k','LineWidth',.5);
525 grid on
526 xlabel('$t~(s)$','interpreter','latex','fontsize',11);
528 ylabel('$H_{\{e\}}(N*m)$','interpreter','latex','fontsize',11)
529
530 %figure
531 %plot(delta_e_fix(vTime))
532 diary off
533 end
534 end

```

4.6.1 Risultati

I risultati ottenuti sono:

Le condizioni di trim trovate sono:

Condizioni di Trim

Soluzione di un problema di trim ad una data altitudine e velocità di volo

File DSV1.txt aperto.

File DSV1.txt letto correttamente.

Le condizioni di trim iniziali ottenute sono:

$\alpha_0 = 2.019^\circ$

$\delta_e,0 = -4.3981^\circ$

$\delta_s,0 = 4.3398e-22^\circ$

$\delta_T,0 = 0.41716$

Le successive figure mostrano le storie temporali delle leggi di comando bloccate, la storia temporale dei fattori di carico, le storie temporali delle variabili di stato per comandi liberi, la storia temporale della deflessione dell'equilibratore e del momento di cerniera.

La linea verticale indica l'istante in cui si passa da comandi bloccati a comandi liberi.

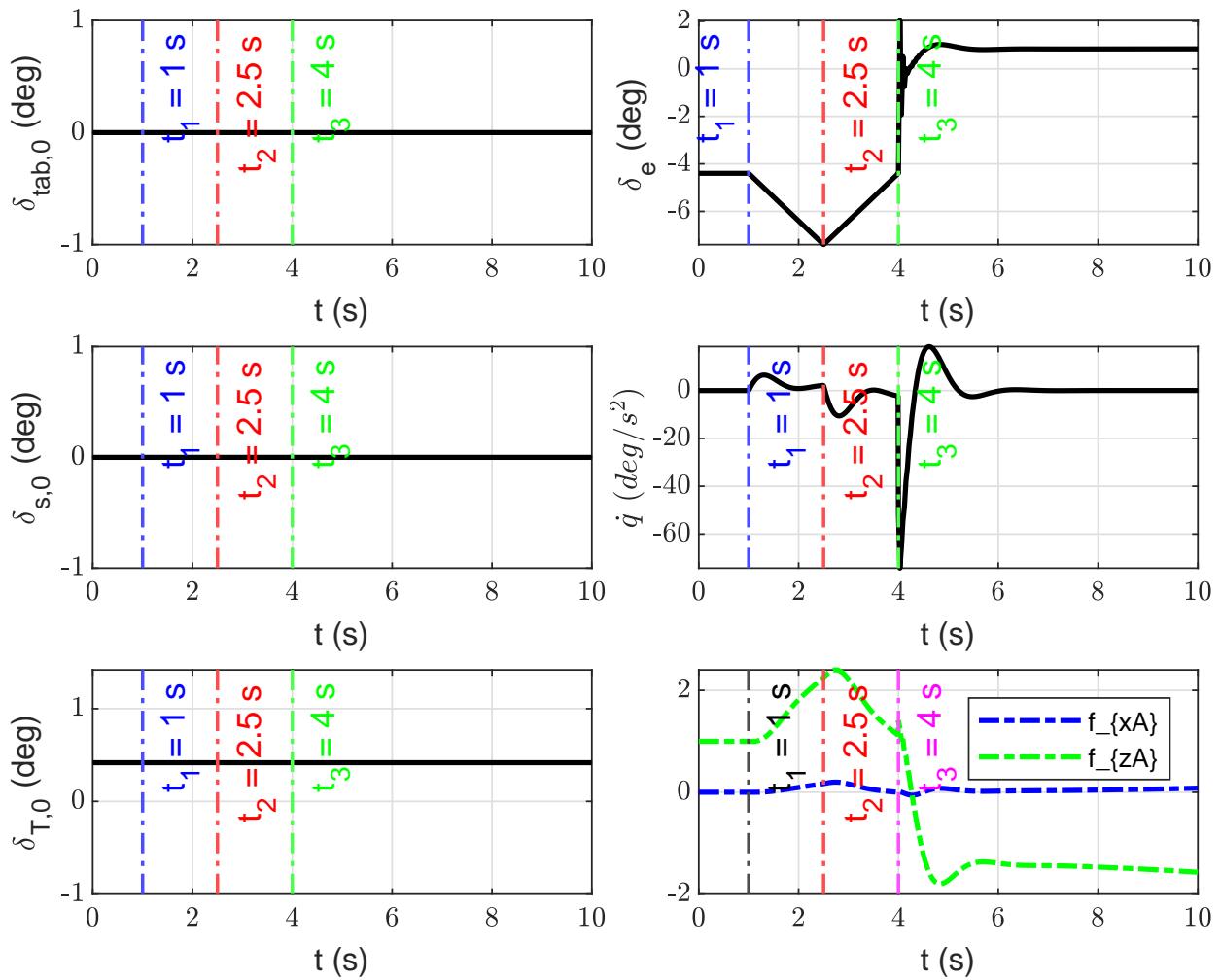


Figura 4.13: Esercizio 4 - Leggi di comando, angolo di deflessione, fattore di carico e accelerazione angolare

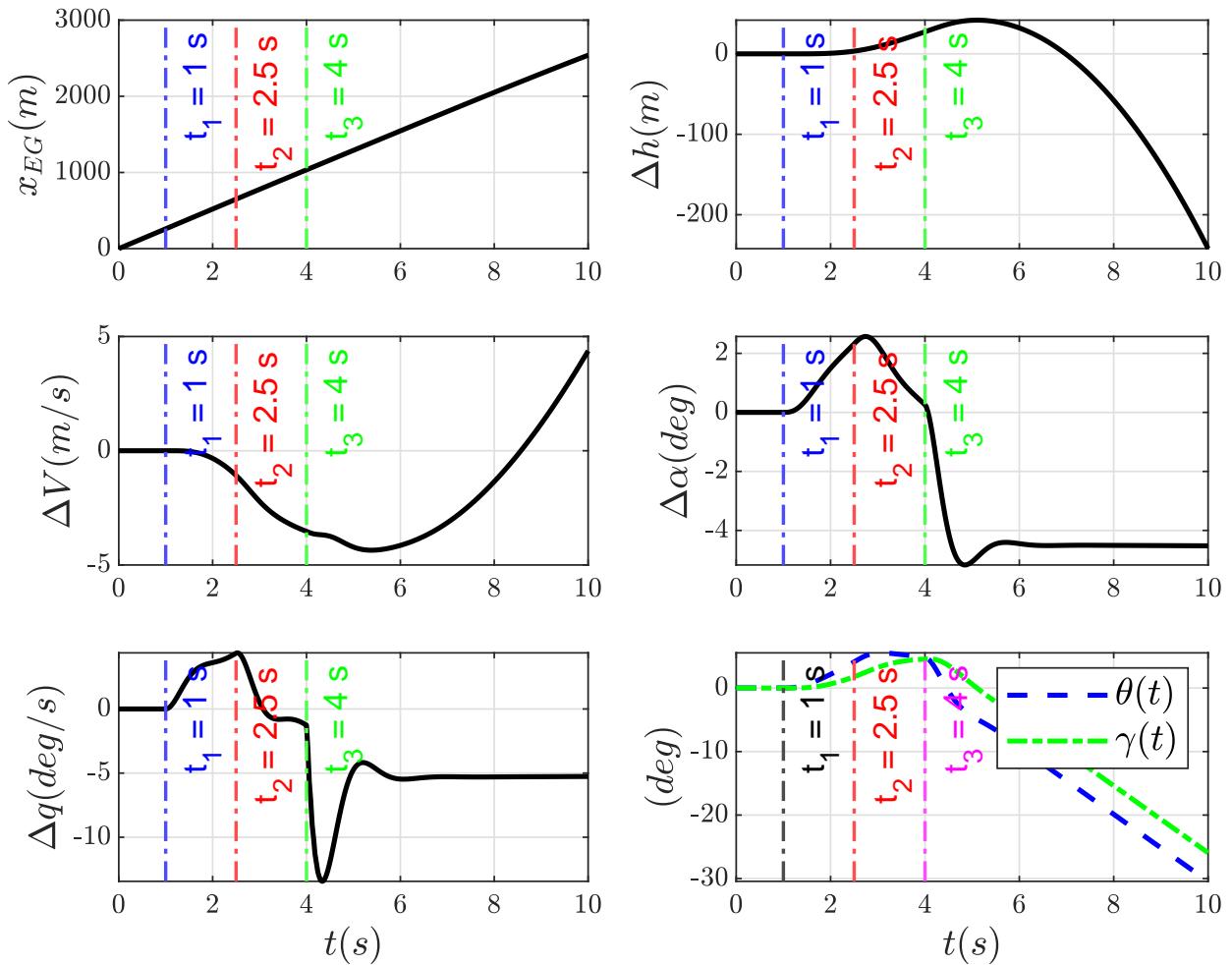


Figura 4.14: Esercizio 4 - Storie temporali delle variabili di stato

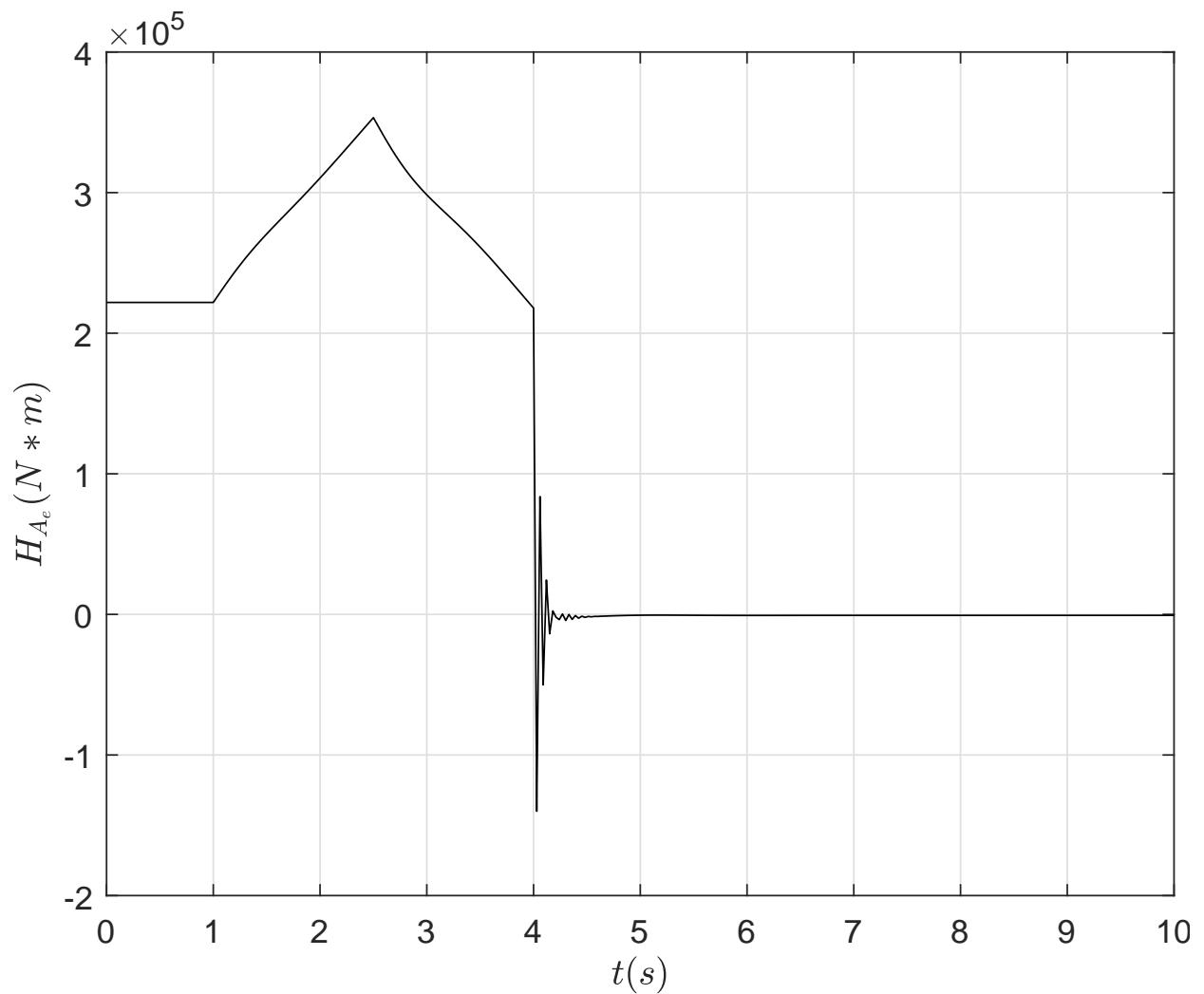


Figura 4.15: Esercizio 4 - Storie temporali del momento di cerniera

4.7 Esercizio 5 (11.6)

Analogamente allo scorso esercizio, viene mostrata una manovra *rampa*.

```

1 function [] = func116(INex11_2)
2 % -----
3 % Esercizi Capitolo 11
4 % -----
5 % Esercizio N°6
6 % -----
7
8
9 %%%%%%CALCOLO CONDIZIONI DI TRIM%%%%%
10 % CALCOLO CONDIZIONI DI TRIM %
11 %%%%%%
12
13 diary('ex11_6.txt')
14 diary on
15 % Ricerca di un set completo di condizioni iniziali per un volo trizzato disp('Moto di un aeroplano a 3-DoF.');
16 disp('Soluzione di un problema di trim ad una data altitudine e velocità di volo');
17
18 %% INPUT
19 % Tipo di Periodo: 1=corto (6s) 2=lungo (50s)
20 INex11_1 = INex11_2;
21 periodo = INex11_1(10);
22 t_switch = 1;
23 % x_0 = ; % posizione longitudinale [m]
24 % z_0 = -4000.0; % a.s.l. (about sea level) altezza [m]
25 % V0 = 260; % vettore velocità [m/s]
26 % q0 = 0.0; % velocità angolare di pitch [rad/s]
27 % gamma0 = 0.0; % angolo di salita [rad]
28
29 % % Vettore dei valori di primo tentativo
30 % x0 = [ ...
31 % 0; ... % 1) alpha_0
32 % 0; ... % 2) delta_e_0. Deflessione equilibratore
33 % 0; ... % 3) delta_s_0. Deflessione stabilizzatore
34 % 0.5]; % 4) delta_T_0. Manette
35
36 %%
37 % Dichiarazione variabili globali
38
39 global ...
40 g ... % accelerazione di gravità
41 z_0 V0 q0 gamma0 ... % condizioni iniziali
42 rho0 ... % densità dell'aria alla quota Z_0
43 delta_tab ... % deflessione aletta tab
44 delta_s ... % legge di comando stabilizzatore
45 delta_T ... % legge di comando manetta
46 delta_e_fix ...
47 myAC % oggetto aereo. (è una sorta di record con
        % diversi campi)
48
49
50 % Aircraft data file name
51 aircraftDataFileName = 'DSV1.txt';
52
53 % Aircraft object
54 myAC = DSVAircraft112(aircraftDataFileName); % viene passato il nome del file
55
56 if (myAC.err == -1) %parametro di controllo per vedere se vi è un
    disp('... terminazione. ');%errore.
57 else
58 disp(['File ', aircraftDataFileName, ' letto correttamente.']);
59
60
61 % Condizioni iniziali. Condizioni di primo tentativo per la ricerca delle
62 % condizioni di trim
63 x_0 = INex11_1(1); % posizione longitudinale [m]
64 z_0 = INex11_1(2); % a.s.l. (about sea level) altezza [m]
65 V0 = INex11_1(3); % vettore velocità [m/s]
66 q0 = INex11_1(4); % velocità angolare di pitch [rad/s]
67 gamma0 = INex11_1(5); % angolo di salita [rad]
68
69 % Funzione matlab del modello ISA. Lo usiamo per il calcolo della densità
70 [air_Temp_0,sound_speed_0, air_pressure_0, rho0] = atmosisa(-z_0);
71
72 % Accelerazione di gravità
73 g = 9.81; % [m/s^2]
74
75 %% Input da DSV.AIRCRAFT
76
77 % Caratteristiche di massa
78 m = myAC.mass; %massa (kg)
79 k_y = myAC.k_y; %raggio di inerzia longitudinale (m)
80
81 % Caratteristiche geometriche
82 c = myAC.mac; % corda media aerodinamica (m)
83 b = myAC.b; % apertura alare (m)
84 S = myAC.S; % superficie alare (m^2)

```

```

85 Xcg_adim = myAC.Xcg_adim; % distanza adimensionale del baricentro dal l.e.
86 Xn_adim = myAC.Xn_adim; %distanza adimensionale del punto neutro dal l.e.
87
88 % Caratteristiche propulsive
89 TO = myAC.T/g; % spinta netta (kgf)
90 mu_T = myAC.mu_T; % calettamento della spinta
91 Cm_T = myAC.Cm_T_0; %contributo al coefficiente mom becch
92
93 % Caratteristiche aerodinamiche
94 CD_0 = myAC.CD_0;
95 K = myAC.K;
96 m = myAC.m;
97
98 CL_alpha = myAC.CL_alpha;
99 CL_delta_e = myAC.CL_delta_e;
100 CL_delta_s = myAC.CL_delta_s;
101 CL_alpha_dot = myAC.CL_alpha_dot;
102 CL_q = myAC.CL_q;
103
104 Cm_0 = myAC.Cm_0;
105 Cm_alpha = myAC.Cm_alpha;
106 Cm_delta_e = myAC.Cm_delta_e;
107 Cm_delta_s = myAC.Cm_delta_s;
108 Cm_alpha_dot = myAC.Cm_alpha_dot;
109 Cm_q = myAC.Cm_q;
110 eps_0 = myAC.eps_0;
111 DepsDalpha = myAC.DepsDalpha;
112
113 % Caratteristiche aerodinamiche dell'equilibratore
114 Ch_e_0 = myAC.Ch_e_0;
115 Ch_e_alpha = myAC.Ch_e_alpha;
116 Ch_e_delta_e = myAC.Ch_e_delta_e;
117 Ch_e_delta_s = myAC.Ch_e_delta_s;
118 Ch_e_delta_tab = myAC.Ch_e_delta_tab;
119 Ch_e_alpha_dot = myAC.Ch_e_alpha_dot;
120 Ch_e_q = myAC.Ch_e_q;
121 Ch_e_delta_e_dot = myAC.Ch_e_delta_e_dot;
122
123
124 % Vettore dei valori di primo tentativo
125 x0 = [ ...
126 INex11_1(6); ... % 1) alpha_0
127 INex11_1(7); ... % 2) delta_e_0. Deflessione equilibratore
128 INex11_1(8); ... % 3) delta_s_0. Deflessione stabilatore
129 INex11_1(9)]; % 4) delta_T_0. Manette
130
131 %% TRIM
132 %%%%%%%% MINIMIZZAZIONE DELLA FUNZIONE DI COSTO %%%%%%%
133
134 % Limiti inferiori (low bound)
135 lb =[convang(-15,'deg','rad'),...%minimum alpha
136 convang(-20,'deg','rad'),...%minimum elevator deflection (delta_e)
137 convang(-5,'deg','rad'),...%minimum stabilizer incidence(delta_s)
138 0.2 ... % minimum thrust fraction (delta_T)
139 ];
140
141 % Limiti superiori (upper bound)
142 ub =[convang( 15,'deg','rad'), ... % maximum alpha
143 convang( 13,'deg','rad'), ... % maximum elevator deflection
144 convang( 5,'deg','rad'), ... % maximum stabilizer incidence
145 1.0 ... % maximum thrust fraction
146 ];
147
148 %Vincolo lineare per delta_s_0 (ipostizziamo sia nullo)
149 Aeq = zeros(4,4);
150 beq = zeros(4,1); Aeq(3,3) = 1;
151 delta_s_0 = convang(0,'deg','rad'); % Ipozziamo deflessione nulla dello stabilatore
152 beq(3,1) = delta_s_0; % manteniamo fisso delta_s
153
154 options = optimset( ...
155 'tolfun',1e-9, ... % tolleranza sull'errore (tolleration function)
156 'Algorithm','interior-point'); %tipologia di algoritmo da utilizzare
157
158 % Funzione per ricavare la condizione di trim
159 [x,fval] = ...
160 fmincon(@costLongEquilibriumStaticStickFixed, ... % funzione per òa ricerca del minimo
161 x0, ... % valore di primo tentativo
162 [], ... % A, A*x=b
163 [], ... % b
164 Aeq, ... % Aeq, Aeq*x=beq
165 beq, ... % beq
166 lb,ub, ... % lower, upper bounds
167 @myNonLinearConstraint, ... % nonlinear const.
168 options); % opzioni di ricerca del minimo
169
170 alpha_0_rad = x(1);
171 alpha_0_deg = convang(x(1),'rad','deg'); delta_e_0_rad = x(2);
172 delta_e_0_deg = convang(x(2),'rad','deg'); delta_s_0_rad = x(3);
173 delta_s_0_deg = convang(x(3),'rad','deg'); delta_T_0 = x(4);

```

```

174 %% calcolo trim tab
175 theta_trim = gamma0+alpha_0_rad-myAC.mu_x;
177
178 % Definizione angoli di interesse
179 alpha_body_0 = alpha_0_rad-myAC.mu_x; % alpha body
180 alpha_H_0 = (1-myAC.DepsAlpha)*(alpha_body_0)... % alpha H
181 -myAC.eps_0 + delta_s_0_rad + myAC.mu_x;
182 % Poichè ci troviamo in condizioni di trim non sono presenti termini
183 % variabili nel tempo
184
185 CH_A_e_delta_tab = ((myAC.mass_e*myAC.ec_adim*myAC.mac_e)...
186 *g*cos(theta_trim))/(0.5*rho0*V0^2*myAC.S_e*myAC.mac_e)...
187 - (myAC.Ch_e_0 + myAC.Ch_e_alpha*alpha_H_0 ...
188 + myAC.Ch_e_delta_e*delta_e_0_rad...
189 + myAC.Ch_e_delta_s*delta_s_0_rad);
190 delta_tab_segnato = CH_A_e_delta_tab/myAC.Ch_e_delta_tab;
191
192
193
194
195 % Mostro a display i risultati ottenuti
196 disp('Le condizioni di trim iniziali ottenute sono: ');
197 disp(['alpha_0 = ', num2str(alpha_0_deg), '°']);
198 disp(['delta_e_0 = ', num2str(delta_e_0_deg), '°']);
199 disp(['delta_s_0 = ', num2str(delta_s_0_deg), '°']);
200 disp(['delta_T_0 = ', num2str(delta_T_0)]);
201 %%%%%%
202 %% RISOLUZIONE EQUAZIONI DELLA DINAMICA %
203 %%%%%%
204 % Risolto il problema del trim si passa alla risoluzione delle equazioni
205 % della dinamica per le quali si fa riferimento alle eq. (7.61)
206
207 % Istanti di inizio e fine osservazione del fenomeno
208 t_0 = 0; % istante iniziale
209 t_1 = t_switch; % istante finale volo a comandi bloccati
210 t_2 = t_switch+2;
211 switch periodo
212 case 1
213 t_finale = 15; % istante finale di osservazione del fenomeno
214 case 2
215 t_finale = 50; % istante finale di osservazione del fenomeno
216 end
217
218 % Mesh
219 N = 200; % numero di punti
220 time1 = linspace (t_0, t_1, N); % punti mesh uniforme
221
222 % Leggi di comando
223 delta_e_fix = @(t) interp1([t_0, t_1, t_2, t_finale],[delta_e_0_rad delta_e_0_rad delta_e_0_rad delta_e_0_rad],t);
224 delta_s = @(t) interp1([t_0, t_1, t_2, t_finale],[delta_s_0_rad delta_s_0_rad delta_s_0_rad delta_s_0_rad],t);
225 delta_T = @(t) interp1 ([t_0, t_1, t_2, t_finale], [delta_T_0 delta_T_0 delta_T_0 delta_T_0], t);
226
227 delta_delta_tab= convang(1.2,'deg','rad');
228 % legge della deflessione 'dellaletta tab'
229 delta_tab = @(t) interp1 ([t_0, t_1, t_2, t_finale], ...
230 [0, 0, delta_tab_segnato+delta_delta_tab, delta_tab_segnato+delta_delta_tab],...
231 t, 'linear');
232
233 %%%%%%
234 % CASO COMANDI BLOCCATI %
235 %%%%%%
236 % Vettore condizioni iniziali
237 X01 = [ VO;
238 alpha_0_rad; q0;
239 x_0; z_0;
240 theta_trim];
241
242 % Risolvo il sistema di equazioni iniziali X0 mediante l'ode 45
243 options = odeset( 'RelTol', 1e-9, 'AbsTol',1e-9*ones(1,6));
244
245 [vTime1, X1] = ode45(@eqLongDynamicStickFixed114, ... % eq. (7.61)
246 time1, ... % Vettore dei tempi
247 X01, ... % Vettore iniziale
248 options); % Vettore delle opzioni
249
250
251 % Mesh
252 N = 200; % numero di punti
253 time2 = linspace (t_1, t_2, N); % punti mesh uniforme
254
255 X02 = X1(end,:);
256
257 % Risolvo il sistema di equazioni iniziali X0 mediante l'ode 45
258 options = odeset( 'RelTol', 1e-9, 'AbsTol',1e-9*ones(1,6));
259
260 [vTime2, X2] = ode45(@eqLongDynamicStickFixed114, ... % eq. (7.61)
261 time2, ... % Vettore dei tempi

```

```

262             X02', ... % Vettore iniziale
263             options); % Vettore delle opzioni
264
265 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
266 %%                                CASO COMANDI LIBERI %
267 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
268 % Circa il vettore dei comandi liberi occorre aggiungere un'ulteriore
269 % incognita
270 % quale, appunto, la deflessione degli equilibratori. Inoltre occorre
271 % considerare come posizione iniziale quelle che riguardano le condizioni
272 % finali precedenti. Per cui X02 = X1 + altre due termini
273 % Istanti di inizio e fine osservazione del fenomeno
274
275 % Mesh
276 N = 200;                               % numero di punti
277 time3 = linspace (t_2, t_finale, N);% punti mesh uniforme
278
279 %Vettore condizioni iniziali che si ottiene a partire da quelle finalinel
280 %caso di comandi bloccati. Si badi che il vettore X1 ottenuto dalla
281 %funzione ode è un vettore ove ogni riga indica il valore del vettore di
282 %stato X1 ad un dato istante. Noi siamo interessati all'istante finale
283 %per cui consideriamo l'ultima riga.
284
285 X03 = [X2(end,:)];
286 0;                                     % derivata prima deflessione equilibratore nullo
287 delta_e_fix(t_2)]; % deflessione iniziale pari a quella all'istante t_1
288
289 % Risolvo il sistema di equazioni iniziali X02 mediante l'ode 45
290 options = odeset( 'RelTol', 1e-9,'AbsTol',1e-9*ones(1,8));
291 [vTime3, X3] = ode45(@eqLongDynamicStickFree, ... % eq. (7.61)
292 time3, ... % Vettore dei tempi
293 X03', ... % Vettore iniziale
294 options); % Vettore delle opzioni
295
296 % Sommo i due vettori così ottenuti. Per far ciò, dato che X1 ha solo 6
297 % colonne dato che 6 sono le incognite, occorre aggiungere altre due
298 % riguardanti la derivata di delta_e e delta_e
299 X1(:,7) = zeros(numel(vTime1),1); % colonna di valori nulli. La funzione numel
300 % restituisce il numero di elementi di vTime1
301 X1(:,8) = delta_e_fix(vTime1); % aggiungo il vettore
302 %dei valori di delta_e fissi
303 X2(:,7) = zeros(numel(vTime2),1); % colonna di valori nulli. La funzione numel
304 % restituisce il numero di elementi di vTime1
305 X2(:,8) = delta_e_fix(vTime2); % aggiungo il vettore
306 %dei valori di delta_e fissi
307
308 time = [vTime1; vTime2(2:end); vTime3(2:end)]; % sommo i due vettori tempo
309 X = [X1; X2(2:end, :);X3(2:end, :)]; % sommo al vettore di stato X1 il vettore X2
310 % ma elimino la prima riga che coincide con
311 % quella finale del vettore X1
312
313 % Incrementi rispetto i valori iniziali
314 DELTA_V      = X(:,1) - VO;
315 DELTA_alpha   = convang (X(:,2), 'rad', 'deg') - alpha_0_deg;
316 DELTA_q       = convangvel (X(:,3), 'rad/s', 'deg/s') - q0;
317 X_EG         = X(:,4);
318 DELTA_h       = -(X(:,5) - z_0);
319 THETA        = X(:,6);
320 GAMMA        = THETA + myAC.mu_x - X(:,2);
321 DELTA_E       = X(:,8);
322
323
324 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
325 %%                                FATTORI DI CARICO %
326 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
327
328 %Per calcolare q_punto, f_xA e f_zA, occorre riprendere le equazioni 7.61
329 %e calcolare i vettori V_punto e gamma_punto.Si osservi che gamma_punto
330 %= theta_punto - alpha_punto (è sufficiente derivare l'equazione 7.64)
331 %eq. (7.61a)
332
333 V            = X(:,1);
334 alpha        = X(:,2);
335 q            = X(:,3);
336 z_EG         = X(:,5);
337 theta        = X(:,6);
338 delta_e     = X(:,8);
339
340
341 % Funzione matlab del modello ISA. Lo usiamo per il calcolo della densità
342 % Serve per aggiornare i parametri atmosferici dato che z_EG può variare
343 [air_Temp_0,sound_speed_0, air_pressure_0, rho] = atmossisa(-z_EG);
344 mu_rel = (myAC.W/g)./(rho*myAC.S*myAC.b); % m/(rho*S*b)
345
346
347 % Calcolo delle derivate mediante la funzione gradient
348 V_punto = gradient(X(:,1), time);
349 alpha_punto = gradient(X(:,2), time);
350 q_punto = gradient(X(:,3), time);

```

```

351
352
353 % eq. (7.61f)
354 theta_punto = q;
355 % Si ricavano, di seguito, le ulteriori funzioni di interesse, quali
356 % gamma_punto e le funzioni fattore di carico lungo gli assi aerodinamici
357 % xA e zA
358 gamma_punto = theta_punto - alpha_punto;
359 f_xa = -(sin(GAMMA)+V_punto/g); % fattore di carico lungo xa
360 f_zz = cos(GAMMA) + (V.*gamma_punto)/g; % fattore di carico lungo za
361
362
363
364 %%%%%%%% MOMENTO DI CERNIERA %%%%%%
365 %
366 %%%%%%%% %%%%%%
367 vTime=time;
368 DELTA_E=convang(DELTA_E,"rad","deg");
369 V=convvel(V,"km/h",'m/s');
370 CH_A_e = myAC.Ch_e_0 + ...
371 myAC.Ch_e_alpha*(1-myAC.DepsDalpha).*(alpha-myAC.mu_x)- ...
372 myAC.eps_0+delta_s([vTime])+myAC.mu_x)+...
373 myAC.Ch_e_delta_e*delta_e+...
374 myAC.Ch_e_delta_s*delta_s([vTime])+ ...
375 myAC.Ch_e_delta_tab*delta_tab([vTime])+ ...
376 myAC.mac_e*myAC.Ch_e_q*q'/(2*V');
377
378 H_A_e = 0.5*rho'*V.^2*myAC.S_e*myAC.mac_e*CH_A_e;
379
380 %% Grafica 1
381
382 figure("NumberTitle","off","Name","Leggi di comando, angolo di deflessione, fattore di carico e accelerazione
    angolare")
383 tiledlayout(3,2,"TileSpacing","Tight");
384
385 nexttile
386 plot(time,delta_tab(time),'-k','LineWidth',1.5)
387 xlabel("t (s)"); ylabel("\delta_{tab,0} (deg)"); grid on;
388 label = [sprintf('t_1 = %g s', t_1)];
389 label2 = [sprintf('t_2 = %g s', t_2)];
390
391 xline(t_1,'-.b',label,'LineWidth',1);
392 xline(t_2,'-.r',label2,'LineWidth',1);
393
394 nexttile
395 plot(time,convang(DELTA_E,'rad','deg'),'-k','LineWidth',1.5)
396 xlabel("t (s)"); ylabel("\delta_e (deg)"); grid on;
397 xl=xline(t_1,'-.b',label,'LineWidth',1);
398 xl.LabelHorizontalAlignment = 'left';
399 xline(t_2,'-.r',label2,'LineWidth',1);
400
401 nexttile
402 plot(time,delta_s(time),'-k','LineWidth',1.5)
403 xlabel("t (s)"); ylabel("\delta_{s,0} (deg)"); grid on;
404
405 xline(t_1,'-.b',label,'LineWidth',1);
406 xline(t_2,'-.r',label2,'LineWidth',1);
407
408 nexttile
409 plot(time,convang(q_punto,'rad','deg'),'-k','LineWidth',1.5)
410 xlabel("t (s)"); ylabel("$\dot{q} \cdot (deg/s^2)$", 'Interpreter','latex'); grid on;
411
412 xline(t_1,'-.b',label,'LineWidth',1);
413 xline(t_2,'-.r',label2,'LineWidth',1);
414
415 nexttile
416 plot(time,delta_T(time),'-k','LineWidth',1.5)
417 xlabel("t (s)"); ylabel("\delta_{T,0} (deg)"); grid on;
418
419 xline(t_1,'-.b',label,'LineWidth',1);
420 xline(t_2,'-.r',label2,'LineWidth',1);
421
422 nexttile
423 plot(time,f_xa,'-.b',time,f_zz,'-.g','LineWidth',1.5)
424 xlabel("t (s)"); grid on;
425 xline(t_1,'-.k',label,'LineWidth',1);
426 xline(t_2,'-.r',label2,'LineWidth',1);
427
428 legend("f_{xA}","f_{zA}", '');
429
430
431
432 %% Grafica 2
433
434 figure("NumberTitle","off","Name","Storie temporali delle variabili di stato")
435
436 h(1) = subplot(3,2,1);
437 plot(time,X_EG,'-k','LineWidth',1.5);
438 grid on

```

```

439 xline(t_1,'-.b',label,'LineWidth',1);
440 xline(t_2,'-.r',label2,'LineWidth',1);
441
442 ylabel('$x_{EG} (m)$','interpreter','latex','fontsize',11)
443 h(2) = subplot(3,2,2);
444 plot(vTime,DELTAt_h,'-k','LineWidth',1.5);
445 grid on
446 xline(t_1,'-.b',label,'LineWidth',1);
447 xline(t_2,'-.r',label2,'LineWidth',1);
448
449 ylabel('$\Delta h (m)$','interpreter','latex','fontsize',11)
450 h(3) = subplot(3,2,3);
451 plot(vTime,DELTAt_V,'-k','LineWidth',1.5);
452 grid on
453 xline(t_1,'-.b',label,'LineWidth',1);
454 xline(t_2,'-.r',label2,'LineWidth',1);
455
456 ylabel('$\Delta V (m/s)$','interpreter','latex','fontsize',11)
457 h(4) = subplot(3,2,4);
458 plot(vTime,DELTAt_alpha,'-k','LineWidth',1.5);
459 grid on
460 xline(t_1,'-.b',label,'LineWidth',1);
461 xline(t_2,'-.r',label2,'LineWidth',1);
462
463 ylabel('$\Delta \alpha (deg)$','interpreter','latex','fontsize',11)
464 h(5) = subplot(3,2,5);
465 plot(vTime,DELTAt_q,'-k','LineWidth',1.5);
466 grid on
467 xline(t_1,'-.b',label,'LineWidth',1);
468 xline(t_2,'-.r',label2,'LineWidth',1);
469
470 xlabel('$t (s)$','interpreter','latex','fontsize',11);
471 ylabel('$\Delta q (deg/s)$','interpreter','latex','fontsize',11)
472 h(6) = subplot(3,2,6);
473 plot(vTime,convang(THETA,'rad','deg'),'--b','LineWidth',1.5);
474 hold on;
475 plot(vTime,convang(GAMMA,'rad','deg'),'-g','LineWidth',1.5);
476 grid on
477 xline(t_1,'-.k',label,'LineWidth',1);
478 xline(t_2,'-.r',label2,'LineWidth',1);
479
480 lgd = legend('$\theta(t)$','$\gamma(t)$','');
481 lgd.Interpreter = 'latex';
482 lgd.FontSize = 11;
483
484 xlabel('$t (s)$','interpreter','latex','fontsize',11);
485
486 ylabel('$(deg)$','interpreter','latex','fontsize',11)
487
488 %%
489 figure("NumberTitle","off","Name","Storie temporali del momento di cerniera")
490 plot(vTime,H_A_e,'-k','LineWidth',.5);
491 grid on
492
493 xlabel('$t (s)$','interpreter','latex','fontsize',11);
494 ylabel('$H_{A_e}(N*m)$','interpreter','latex','fontsize',11)
495 diary off
496 end
497 end

```

Condizioni di Trim

Soluzione di un problema di trim ad una data altitudine e velocità di volo
 File DSV1.txt aperto.
 File DSV1.txt letto correttamente.

Le condizioni di trim iniziali ottenute sono:
 $\alpha_0 = 2.019^\circ$
 $\delta_e,0 = -4.3981^\circ$
 $\delta_s,0 = 4.3398e-22^\circ$
 $\delta_T,0 = 0.41716$

Le successive figure mostrano le storie temporali delle leggi di comando bloccate, la storia temporale dei fattori di carico, le storie temporali delle variabili di stato per comandi liberi, la storia temporale della deflessione dell'equilibratore e del momento di cerniera.

La linea verticale indica l'istante in cui si passa da comandi bloccati a comandi liberi.

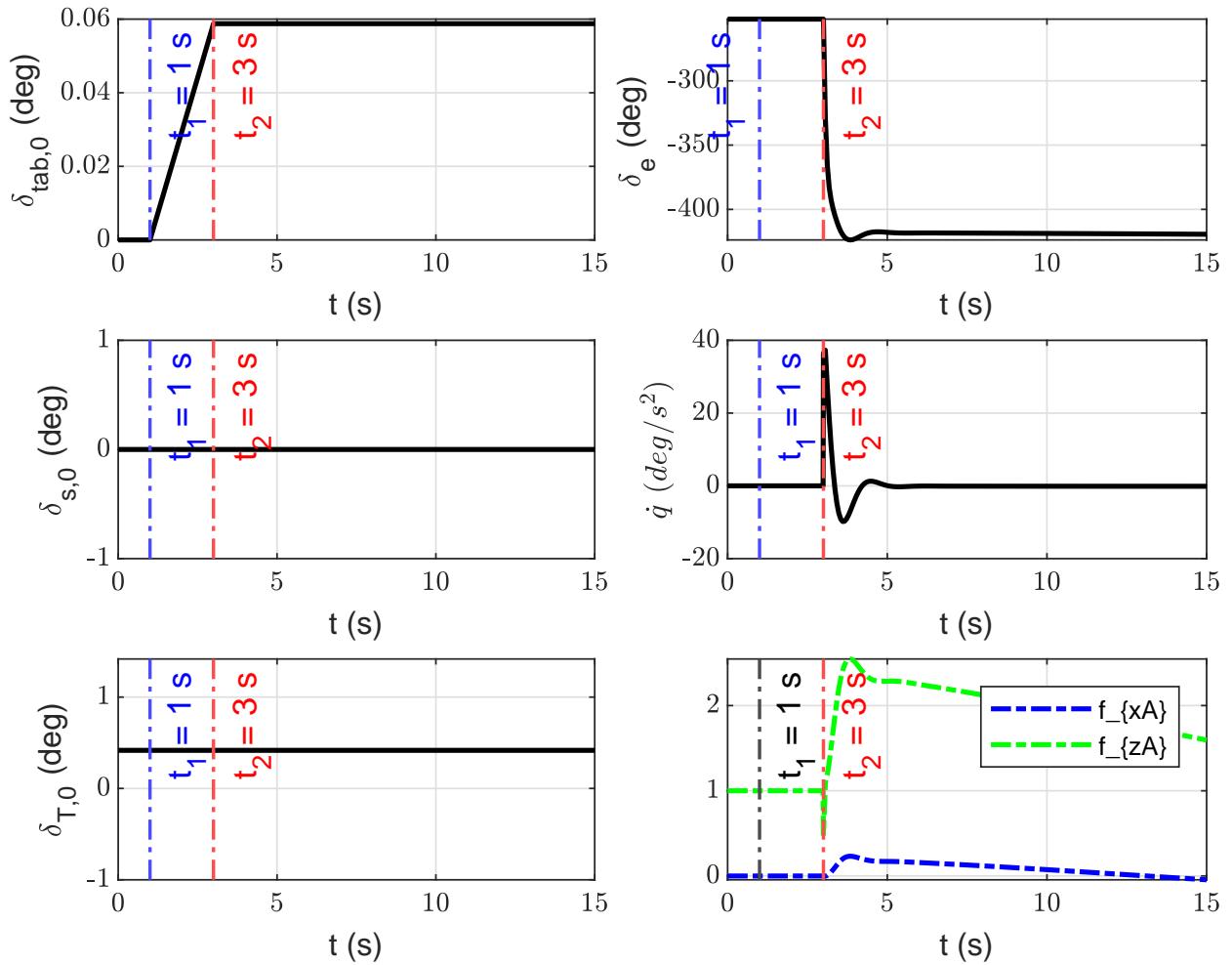


Figura 4.16: Esercizio 5 - Leggi di comando, angolo di deflessione, fattore di carico e accelerazione angolare

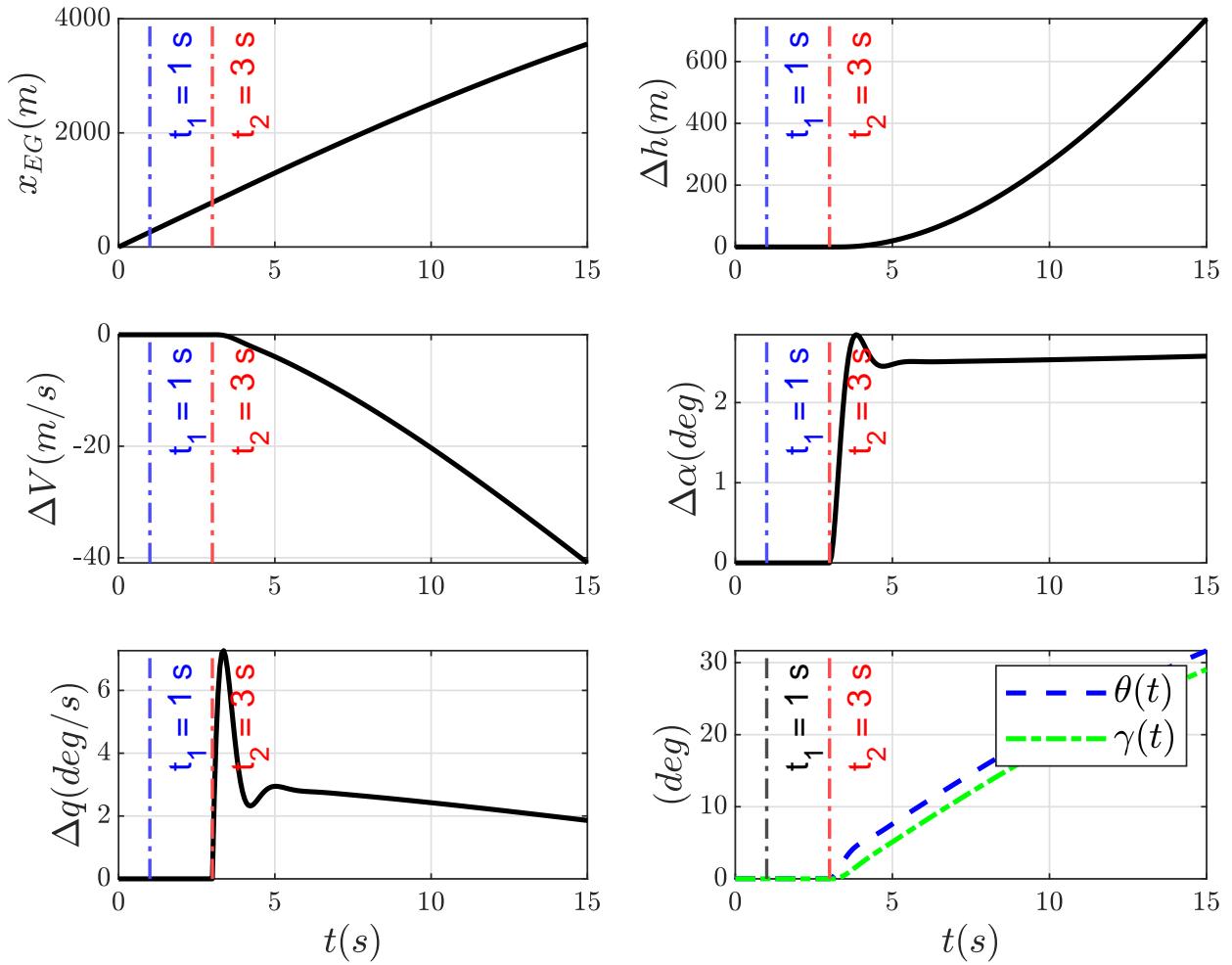


Figura 4.17: Esercizio 5 - Storie temporali delle variabili di stato

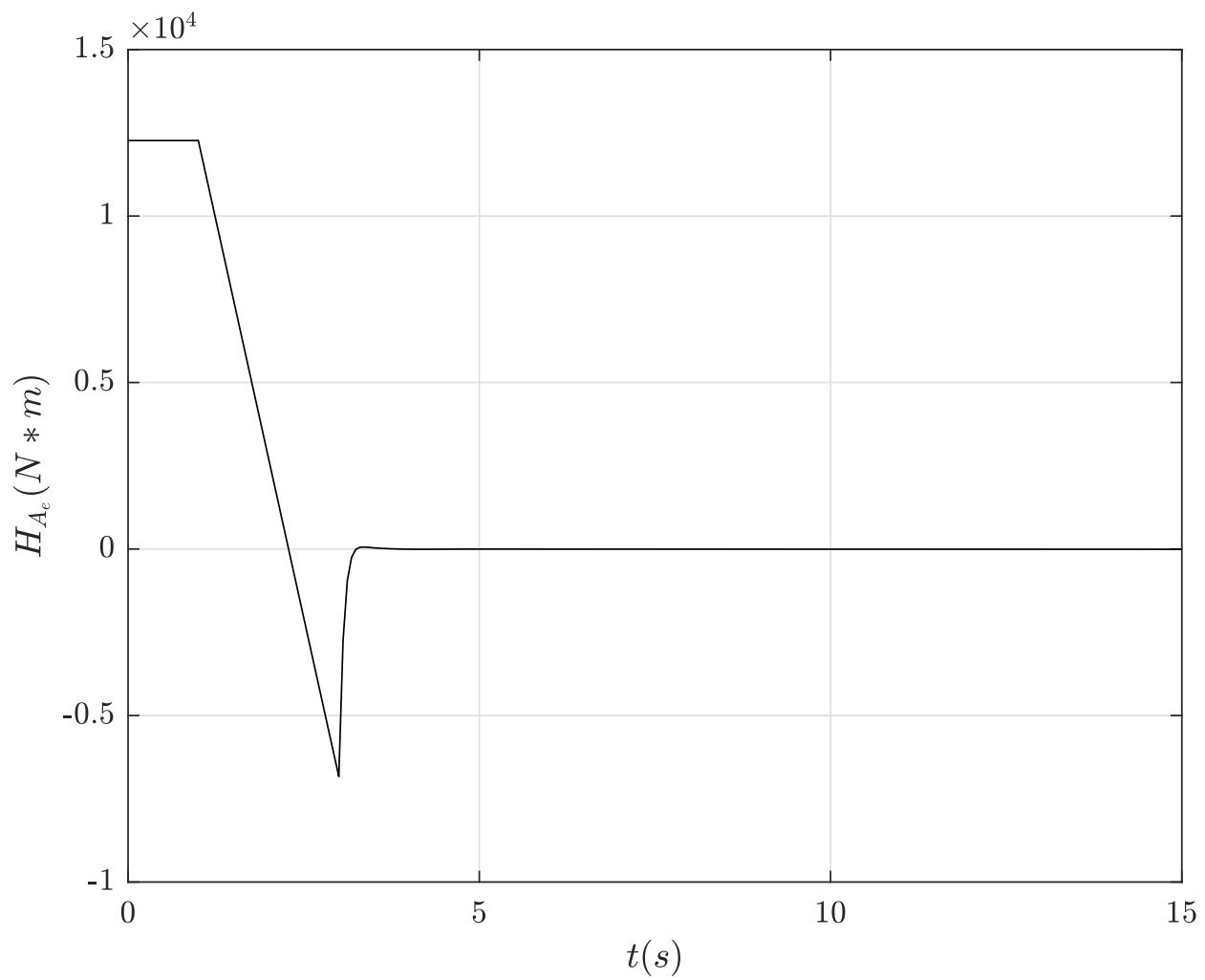


Figura 4.18: Esercizio 5 - Storie temporali del momento di cerniera

Capitolo 5

Piccole perturbazioni nel moto di un velivolo

5.1 Introduzione

L'obiettivo di questa esercitazione è l'analisi della dinamica longitudinale del velivolo, ossia dell'evoluzione di piccole perturbazioni delle variabili simmetriche intorno a una condizione di moto equilibrato, ad ali livellate e a velocità angolare nulla. Si parte dalle seguenti equazioni. Equazioni generali della traslazione del baricentro

$$\begin{cases} m(\dot{u} + qw - rv) = X_G + X_A + X_T \\ m(\dot{v} + ru - pw) = Y_G + Y_A + Y_T \\ m(\dot{w} + pv - qu) = Z_G + Z_A + Z_T \end{cases} \quad (5.1)$$

Sistema di equazioni alla rotazione intorno al baricentro:

$$\begin{cases} I_{xx}\dot{p} - I_Pxz(\dot{r} + pq) - (I_{yy} - I_{zz})qr = \mathcal{L}_A + \mathcal{L}_T \\ I_{yy}\dot{q} - I_Pxz(r^2 - p^2) - (I_{zz} - I_{xx})qr = \mathcal{M}_A + \mathcal{M}_T \\ I_{zz}\dot{r} - I_Pxz(\dot{r} - qr) - (I_{xx} - I_{yy})qr = \mathcal{N}_A + \mathcal{N}_T \end{cases} \quad (5.2)$$

E' necessario anche riprendere le gimbal equation:

$$\begin{cases} \dot{\phi} = p + (q \sin\phi + r \cos\phi) \frac{\sin\theta}{\cos\theta} \\ \dot{\theta} = q \cos\phi - r \sin\phi \\ \dot{\psi} = (q \sin\phi + r \cos\phi) \frac{1}{\cos\theta} \end{cases} \quad (5.3)$$

e le equazioni della traiettoria del baricentro in assi Terra:

$$\begin{Bmatrix} \dot{x}_{E,G} \\ \dot{y}_{E,G} \\ \dot{z}_{E,G} \end{Bmatrix} = [T(\phi, \theta, \psi)]_{EB} \begin{Bmatrix} u \\ v \\ w \end{Bmatrix} \quad (5.4)$$

Le precedenti equazioni costituiscono un sistema di 12 equazioni differenziali ordinarie nelle seguenti 12 incognite:

$$[u, v, w, p, q, r, x_{E,G}, y_{E,G}, z_{E,G}, \phi, \theta, \psi]^T \triangleq \mathbf{x} = [x_1, \dots, x_{12}]^T \quad (5.5)$$

Gli ingressi sono invece:

$$\mathbf{u} \triangleq [\delta_T, \delta_e, \delta_a, \delta_r]^T \quad (5.6)$$

Le prime 6 sono dinamiche, le altre cinematiche.

$$\begin{aligned} \dot{x}_d &= f_d(x_d, x_k, u) \\ \dot{x}_k &= f_d(x_d, x_k) \end{aligned} \quad (5.7)$$

Quindi:

$$\dot{x} = f(x, u) \quad \text{con } f = [f_d^T, f_l^T]^T \quad (5.8)$$

Si parte da una condizione detta *nominal*e, corrispondente al volo equilibrato in cui c'è la stazionarietà delle variabili dinamiche. ($\dot{x}_d = 0$). Quindi all'equilibrio:

$$[u_0, v_0, w_0, p_0, q_0, r_0]^T \triangleq x_{d0} \quad (5.9)$$

Si fanno le ulteriori semplificazioni:

- volo inizialmente traslato $p_0 = q_0 = r_0 = 0$;
- volo simmetrico $v_0 = \phi_0 = 0$;
- angolo di prua iniziale $\psi_0 = 0$;
- assi stabilità ($\alpha_0 = 0$), quindi $\theta_0 = \gamma_0$.

Si fa l'ipotesi di piccoli perturbazioni (piccole variazioni della condizione nominale), cioè:

$$x(t) = x_N(t) + \Delta x(t) \quad (5.10)$$

ad esempio $U = U_0 + u$, in cui U_0 è la condizione iniziale e u è la variazione. Si sostituisce quindi nelle precedenti equazioni. Ad esempio prendiamo la terza delle (5.1).

$$m((0 + \dot{w}) + (P_0 + p)(V_0 + v) - (Q_0 + q)(U_0 + u)) = (Z_{G0} + \Delta Z_G) + (Z_{A0} + \Delta Z_A) + (Z_{T0} + \Delta Z_T) \quad (5.11)$$

e riconoscendo:

$$m(P_0 V_0 - Q_0 U_0) = Z_{G0} + (Z_{A0} + (Z_{T0}) \quad (5.12)$$

Si ottiene:

$$m(\dot{w} + (P_0 v)(p V_0) + p v - (Q_0 u)(q U_0) - q u) = (\Delta Z_G) + (\Delta Z_A) + (\Delta Z_T) \quad (5.13)$$

In cui si possono ignorare i termini $p v$ e $q u$.

$$m(\dot{w} + (P_0 v)(p V_0) - (Q_0 u)(q U_0)) = \Delta Z_G + \Delta Z_A + \Delta Z_T \quad (5.14)$$

Ragionamento analogo per le equazioni alla rotazione. Si ottengono così delle variazioni delle azioni aerodinamiche, propulsive e gravitazionali che sono esprimibili, per ipotesi di piccole perturbazioni, come combinazione lineare dei parametri del moto e di comando.

Perturbazioni delle componenti del peso

Esplicitando la matrice di trasformazione $[T]_{BE}$ nel prodotto $[T]_{BE}\{W\}_E$ si ottengono le *perturbazioni delle componenti del peso in assi velivolo*:

$$\begin{cases} \Delta X_G = -mg(\cos\Theta_0)\theta \\ \Delta Y_G = mg[-(\sin\Theta_0\sin\Phi_0)\theta + (\cos\Theta_0\cos\Phi_0)\phi] \\ \Delta Z_G = mg[-(\sin\Theta_0\cos\Phi_0)\theta - (\cos\Theta_0\sin\Phi_0)\phi] \end{cases} \quad (5.15)$$

Che, con $\cos\phi \approx 1$, $\sin\phi \approx \phi$, $\cos\theta \approx 1$, $\sin\theta \approx \theta$, diventa:

$$\begin{cases} \Delta X_G = -mg(\cos\Theta_0)\theta \\ \Delta Y_G = mg(\cos\Theta_0)\phi \\ \Delta Z_G = -mg(\sin\Theta_0)\theta \end{cases} \quad (5.16)$$

Equazioni cinematiche dell'orientamento

Prendendo come esempio la prima delle (5.3), si può riscrivere come:

$$(\dot{\Phi}_0 + \dot{\phi}) - (\dot{\Psi}_0 + \dot{\psi})\sin(\Theta_0 + \theta) = P_0 + p \quad (5.17)$$

Riscrivibile come:

$$[E]\{\dot{x}\} = [A]\{x\} + [B]\{u\} \quad (5.18)$$

In cui $[B] = [I]$ (matrice identica), $[E]$ ed $[A]$ contengono i termini linearizzati. Si perviene con alcuni passaggi ad una scrittura del tipo $\{\dot{x}\} = [\mathcal{A}]\{x\} + [\mathcal{B}]\{u\}$.

In una condizione di volo traslato, simmetrico e ad ali livellate si ottiene la seguente:

$$\begin{Bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{Bmatrix} = \begin{bmatrix} 1 & 0 & \frac{S_{\Theta_0}}{C_{\Theta_0}} \\ 0 & 1 & 0 \\ 0 & 0 & \frac{1}{C_{\Theta_0}} \end{bmatrix} \begin{Bmatrix} p \\ q \\ r \end{Bmatrix} \quad (5.19)$$

Per cui:

$$\dot{\phi} = p + \tan \Theta_0 r, \quad \dot{\theta} = q, \quad \dot{\psi} = r / \cos \Theta_0 \quad (5.20)$$

Queste espressione mostrano il *disaccoppiamento cinematico* tra le perturbazioni rotazionali del moto longitudinale e di quelle del moto latero-direzionale. Se la traiettoria è orizzontale ($W_0 = \Theta_0 = \Gamma_0 = 0$) diventano:

$$\dot{\phi} = p, \quad \dot{\theta} = q, \quad \dot{\psi} = r \quad (5.21)$$

Equazioni cinematiche della navigazione

In maniera analoga si giunge alle:

$$\begin{Bmatrix} \dot{x}_{E,G} \\ \dot{y}_{E,G} \\ \dot{z}_{E,G} \end{Bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{Bmatrix} u \\ v \\ w \end{Bmatrix} + \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & U_0 \\ 0 & -U_0 & 0 \end{bmatrix} \begin{Bmatrix} \phi \\ \theta \\ \psi \end{Bmatrix} \quad (5.22)$$

Quindi:

$$\dot{X}_{E,G} = U_0 + u, \quad \dot{Y}_{E,G} = v + U_0 \psi, \quad \dot{Z}_{E,G} = w - U_0 \theta \quad (5.23)$$

Che sono le equazioni della navigazione linearizzate, in assi stabilità, per volo iniziale traslato ad ali livellate e quota costante. Si evidenzia il disaccoppiamento cinematico tra le variabili.

5.1.1 Condizioni di Trim

Il sistema completo di equazioni linearizzate risulta molto semplificato se la condizione nominale corrisponde al volo traslato, a velocità costante, ad ali livellate e a quota costante, e per assi velivolo coincidenti con gli assi stabilità della condizione iniziale. Tale condizione viene detta *condizioni di trim ad ali livellate*.

Ponendo

$$V_G = V_0 + v \quad \Omega_B = \Omega_0 + \omega \quad F = F_0 + \Delta F \quad \mathcal{M} = \mathcal{M}_0 + \Delta \mathcal{M} \quad (5.24)$$

si ottengono (dalle equazioni generali del moto):

$$\begin{aligned} m(\{\dot{v}_G\}_B + [\tilde{\Omega}_0]_B \{v\}_B + [\tilde{\omega}_0]_B \{V_0\}_B) &= \{\Delta F\}_B \\ [I]_B \{\dot{\omega}\} + [\tilde{\Omega}_0]_B ([I]_B \{\omega\}_B) + [\tilde{\omega}_0]_B ([I]_B \{\Omega\}_B) &= \{\Delta \mathcal{M}\}_B \end{aligned} \quad (5.25)$$

si verificano le identità di equilibrio:

$$\begin{aligned} m(\{\dot{v}_G\}_B + [\tilde{\omega}_0]_B \{V_0\}_B) &= \{F_0\}_B \\ [\tilde{\Omega}_0]_B ([I]_B \{\Omega\}_B) &= \{\mathcal{M}_0\}_B \end{aligned} \quad (5.26)$$

e poi, in condizioni di trim:

$$\begin{aligned} (\{\dot{v}_G\}_B + [\tilde{\omega}_0]_B \{V_0\}_B) &= \frac{1}{m} \{\Delta F\}_B \\ [I]_B \{\dot{\omega}\} &= \{\Delta \mathcal{M}\}_B \end{aligned} \quad (5.27)$$

Che scritta in forma estesa è:

$$\begin{cases} m(\dot{u} + W_0 q) = -mg C_{\Theta_0} \theta + \Delta X_A + \Delta X_T \\ m(\dot{v} + U_0 r - W_0 p) = mg C_{\Theta_0} \phi + \Delta Y_A + \Delta Y_T \\ m(\dot{w} - U_0 q) = -mg S_{\Theta_0} \theta + \Delta Z_A + \Delta Z_T \end{cases} \quad (5.28)$$

Analogamente:

$$\begin{cases} I_{xx} \dot{p} - I_{xz} \dot{r} = \Delta \mathcal{L}_A + \Delta \mathcal{L}_T \\ I_{yy} \dot{q} = \Delta \mathcal{M}_A + \Delta \mathcal{M}_T \\ I_{zz} \dot{r} - I_{xz} \dot{p} = \Delta \mathcal{N}_A + \Delta \mathcal{N}_T \end{cases} \quad (5.29)$$

5.1.2 Azioni aerodinamiche linearizzate

E' possibile, con Taylor, scrivere una funzione nella seguente forma:

$$\begin{aligned} f(\Delta x_1, \Delta x_2) &= f_0 + f_{x_1} \Delta x_1 + f_{x_2} \Delta x_2 \\ &+ \frac{1}{2} (f_{x_1 x_1} \Delta x_1^2 + 2 f_{x_1 x_2} \Delta x_1 \Delta x_2 + f_{x_2 x_2} \Delta x_2^2) + \dots + R_n \end{aligned} \quad (5.30)$$

cioè stiamo linearizzando f intorno alla condizione di riferimento (\bar{x}_1, \bar{x}_2) . Ci fermiamo al primo ordine, dunque:

$$f(\Delta x_1, \Delta x_2) \approx f_0 + f_{x_1} \Delta x_1 + f_{x_2} \Delta x_2 \quad (5.31)$$

Che utilizzeremo per linearizzare le azioni aerodinamiche.

$$\begin{aligned} \Delta f_A &= C_1 u + C_2 \dot{u} + C_3 v + C_4 \dot{v} + C_5 w + C_6 \dot{w} \\ &+ C_7 p + C_8 \dot{p} + C_9 q + C_{10} \dot{q} + C_{11} r + C_{12} \dot{r} \\ &+ C_{13} z_E \\ &+ C_{14} \delta_a + C_{15} \dot{\delta}_a + C_{16} \delta_e + C_{17} \dot{\delta}_e + C_{18} \delta_r + C_{19} \dot{\delta}_r \\ &+ C_{20} \delta_T + C_{21} \dot{\delta}_T \end{aligned} \quad (5.32)$$

Le derivate di stabilità ottenute con questa logica sono riportate nella seguente tabella:

$\frac{\partial \rightarrow}{\partial(\downarrow)}$	X_A	Y_A	Z_A	\mathcal{L}_A	\mathcal{M}_A	\mathcal{N}_A	
u	□	0	□	0	□	0	
v	0	•	0	•	0	•	
w	□	0	□	0	□	0	
\dot{w}	≈ 0	0	□	0	□	0	
p	0	≈ 0	0	•	0	•	
q	≈ 0	0	□	0	□	0	
r	0	•	0	•	0	•	
δ_a	0	≈ 0	0	•	0	•	
δ_e	≈ 0	0	□	0	□	0	
δ_r	0	•	0	•	0	•	

Si vede anche qui il disaccoppiamento tra le equazioni linearizzate longitudinali e latero-direzionali.

5.1.3 Azioni Propulsive Linearizzate

Le azioni propulsive vengono espresse come:

$$T_0 = \delta_{T0} T_{\max}(z_0), \quad X_{T0} = T_0 \cos \mu_T, \quad Z_{T0} = -T_0 \sin \mu_T, \quad \mathcal{M}_{T0} = T_0 e_T \quad (5.34)$$

in cui:

- T_{\max} è la spinta massima
- δ_{T0} è la manetta
- μ_T è un angolo costante (se la spinta non è orientabile)
- e_T è la distanza dell'asse di spinta dal baricentro

La spinta viene generalmente modellata con la seguente espressione:

$$T = C_T(|V|, \delta_{T0} + \delta_T) \frac{1}{2} \rho |V|^2 S \quad (5.35)$$

Ipotizzando spinta baricentrica ($e_T = 0$) e μ_T nullo:

$$T_0 = \delta_{T0} T_{\max}(z_0), \quad X_{T0} = T_0, \quad Z_{T0} = \mathcal{M}_{T0} = 0 \quad (5.36)$$

Il coefficiente di spinta è modellato come segue:

$$C_T = (C_{T_{F1}} + k_V |V|^\eta) (\delta_{T0} + \delta_T) \approx [C_{T_{Fx}} + k_V (U_0 + u)^\eta] (\delta_{T0} + \delta_T) \quad (5.37)$$

Dove:

- il numero costante $C_{T\text{FF}}$ è il coefficiente di spinta a punto fisso, cioè quando $U_0 = u = 0$ e $\delta_{T0} + \delta_T = 1$,
- la quantità costante k_V è un fattore di scala dell'effetto dovuto alla velocità di volo,
- $,\eta$ è un esponente, tipicamente negativo, funzione del tipo di motore.

Questa espressione va derivata rispetto a u/U_0 e rispetto a δ_T .

Il modello di spinta (5.34) permette di formulare la seguente espressione linearizzata:

$$\Delta T = \bar{q}_0 S \left[2C_T|_0 + C_{T_u}|_0 \right] \frac{u}{U_0} + C_{T_{\delta_T}}|_0 \delta_T \quad (5.38)$$

5.1.4 Sistema LTI

L'ipotesi di piccole perturbazioni permette di linearizzare il sistema delle equazioni del moto costituito dall'insieme delle *equazioni dinamiche* e delle *relazioni cinematiche ausiliarie*, arrivando a un sistema di equazioni lineari che può essere studiato con i metodi classici dei *sistemi lineari tempo-invarianti* (LTI). Il sistema lineare può essere scomposto in due sistemi:

- il sistema di equazioni della dinamica longitudinale;
- il sistema di equazioni della dinamica latero-direzionale.

Parliamo dunque di separazione tra dinamica tra i moti longitudinale e latero-direzionale. Andremo ad affrontare la dinamica longitudinale del velivolo. Le equazioni linearizzate sono:

$$\begin{Bmatrix} \dot{x}_{LON} \\ \dot{x}_{LD} \end{Bmatrix} = \begin{bmatrix} [F_{LON}] & 0 \\ 0 & [F_{LD}] \end{bmatrix} \begin{Bmatrix} x_{LON} \\ x_{LD} \end{Bmatrix} + \begin{bmatrix} [G_{LON}] & 0 \\ 0 & [G_{LD}] \end{bmatrix} \begin{Bmatrix} u_{LON} \\ u_{LD} \end{Bmatrix}$$

Che sono disaccoppiabili nelle:

$$\begin{aligned} \{\dot{x}_{LON}\} &= [F_{LON}] \{x_{LON}\} + [G_{LON}] \{u_{LON}\} \\ \{\dot{x}_{LD}\} &= [F_{LD}] \{x_{LD}\} + [G_{LD}] \{u_{LD}\} \end{aligned} \quad (5.39)$$

Identificabili come due equazioni differenziali ordinarie a coefficienti costanti (LTI), del tipo:

$$\{\dot{x}\} = [A] \{x\} + [B] \{u\} \quad (5.40)$$

Nelle incognite $x_{LON}(t)$ e $x_{LD}(t)$ per assegnate leggi di comando $u_{LON}(t)$ e $u_{LD}(t)$. Ciascuno dei due sistemi ha una matrice $[A]$ di dimensioni 6×6 , mentre la matrice $[B]$ ha dimensioni $6 \times n$, in cui n è il numero dei comandi (in genere 2, ma variabile ad esempio con aggiunta di canard o spoiler). Assumiamo condizioni iniziali omogenee:

$$\begin{aligned} \{x_{LON}\}_0 &= \{u_{LON}\}_0 = 0 \\ \{x_{LD}\}_0 &= \{u_{LD}\}_0 = 0 \end{aligned} \quad (5.41)$$

Per definizione di moto perturbato, per leggi di comando identicamente nulle, entrambi i problemi a valori iniziali (5.39)-(5.41) hanno una soluzione identicamente nulla. Dei sistemi (5.39) interessano le soluzioni non banali che originano da condizioni iniziali non nulle a comandi bloccati (quindi ingressi \mathbf{u} identicamente nulli). Questa è una *risposta libera* del sistema velivolo a comandi bloccati provocata da una perturbazione delle condizioni iniziali rispetto alla condizione nominale. Un'importante conseguenza del disaccoppiamento delle equazioni linearizzate è che per piccole perturbazioni una risposta latero-direzionale $x_{LD}(t)$ origina esclusivamente disturbi delle variabili asimmetriche, e in maniera analoga una risposta longitudinale $x_{LON}(t)$ origina esclusivamente disturbi delle variabili simmetriche.

5.1.5 Modi di risposta dei velivoli

Le $n_x = 12$ equazioni linearizzate del moto di un velivolo sono caratterizzate da un polinomio caratteristico del tipo:

$$\begin{aligned} \Delta(s) &= s^{12} + a_{11}s^{11} + \cdots + a_1s + a_0 \\ &= (s - \lambda_1)(s - \lambda_2) \cdots (s - \lambda_{12}) \end{aligned} \quad (5.42)$$

con radici λ_k ($k = 1, \dots, n_x$). Dalla separazione delle equazioni possiamo separare gli autovalori in due gruppi distinti.

$$\begin{aligned}\Delta_{LON}(s) &= (s - \lambda_1)(s - \lambda_2) \cdots (s - \lambda_6) \\ \Delta_{LD}(s) &= (s - \lambda_7)(s - \lambda_8) \cdots (s - \lambda_{12})\end{aligned}\quad (5.43)$$

Quindi:

$$\Delta(s) = \Delta_{LON}(s)\Delta_{LD}(s) \quad (5.44)$$

Possiamo riformulare le (5.43) come segue:

$$\begin{aligned}\Delta_{LON}(s) &= (s - \lambda_{RANGE})(s - \lambda_{HEIGHT})(s - \lambda_{PH})(s - \lambda_{PH}^*)(s - \lambda_{SP})(s - \lambda_{SP}^*) \\ \Delta_{LD}(s) &= (s - \lambda_{CROSSRANGE})(s - \lambda_{HEADING}) \\ &\quad \times (s - \lambda_{SPIRAL})(s - \lambda_{ROLL})(s - \lambda_{DR})(s - \lambda_{DR}^*)\end{aligned}\quad (5.45)$$

Le radici reali sono legate al modo di *range* e *height*. Le radici complesse sono relative a due modi:

- *modo di lungo periodo* o di *fugoide* (phugoid, PH), possiede una frequenza bassa e uno smorzamento relativamente piccolo
- *modo di corto periodo* (short period, SP), caratterizzata da una frequenza elevata e forte smorzamento

I tipici modi di risposta oscillanti della dinamica longitudinale possono essere espressi in termini della loro frequenza naturale ω_n e del loro rapporto di smorzamento ζ . Esplicitandole otteniamo per il fugoide la:

$$\begin{aligned}(s - \lambda_{PH})(s - \lambda_{PH}^*) &= [s - (\sigma_{PH} + j\omega_{PH})][s - (\sigma_{PH} - j\omega_{PH})] \\ &= s^2 - 2\zeta_{PH}\omega_{n,PH}s + \omega_{n,PH}^2\end{aligned}\quad (5.46)$$

mentre per il corto periodo la:

$$\begin{aligned}(s - \lambda_{SP})(s - \lambda_{SP}^*) &= [s - (\sigma_{SP} + j\omega_{SP})][s - (\sigma_{SP} - j\omega_{SP})] \\ &= s^2 - 2\zeta_{SP}\omega_{n,SP}s + \omega_{n,SP}^2\end{aligned}\quad (5.47)$$

Si osserva infine che in genere si ha:

$$\lambda_{Range} \approx 0, \quad \lambda_{Height} \approx 0 \quad (5.48)$$

Dunque i modi di *range* e *height* sono di scarsa rilevanza.

Si osserva che la dinamica longitudinale del velivolo è essenzialmente caratterizzata da due modi di risposta oscillanti determinati dall'interazione dei disturbi di velocità, angolo d'attacco e angolo di beccheggio.

5.1.6 Dinamica Longitudinale

In questo elaborato si darà risalto principalmente alle equazioni della dinamica longitudinale. È possibile estrapolare (a partire dalle precedenti equazioni), questa formula:

$$\dot{x}_{Lon} = A_{Lon} x_{Lon} + B_{Lon} u_{Lon} \quad (5.49)$$

In cui:

$$x_{Lon} = \begin{Bmatrix} u \\ w \\ q \\ \theta \end{Bmatrix} \quad u_{Lon} = \begin{Bmatrix} \delta_T \\ \delta_e \end{Bmatrix} \quad (5.50)$$

$$A_{Lon} = \left[\begin{array}{cccc} \hat{X}_u & \hat{X}_w & 0 & -g C_{\Theta_0} \\ \frac{\hat{Z}_u}{1-\hat{Z}_{\dot{\psi}}} & \frac{\hat{Z}_w}{1-\hat{Z}_{\dot{\psi}}} & \frac{\hat{Z}_q + U_0}{1-\hat{Z}_{\dot{\psi}}} & \frac{-g S_{\Theta_0}}{1-\hat{Z}_{\dot{\psi}}} \\ \hat{M}_u + k\hat{Z}_u & \hat{M}_w + k\hat{Z}_w & \hat{M}_q + k(\hat{Z}_u + U_0) & -g \hat{k} S_{\Theta_0} \\ 0 & 0 & 1 & 0 \end{array} \right] \quad (5.51)$$

$$B_{Lon} = \left[\begin{array}{cc} \hat{X}_{\delta_T} & \hat{X}_{\delta_e} \\ \frac{\hat{Z}_{\delta_T}}{1-\hat{Z}_{\dot{\psi}}} & \frac{\hat{Z}_{\delta_e}}{1-\hat{Z}_{\dot{\psi}}} \\ \hat{M}_{\delta_T} + k\hat{Z}_{\delta_T} & \hat{M}_{\delta_e} + k\hat{Z}_{\delta_e} \\ 0 & 0 \end{array} \right] \quad (5.52)$$

con:

$$\hat{X}_\xi = \frac{X_\xi}{m} \quad \hat{Z}_\xi = \frac{Z_\xi}{m} \quad \hat{\mathcal{M}}_\xi = \frac{\mathcal{M}_\xi}{I_{yy}} \quad (5.53)$$

per $\xi \in \{u, w, \dot{w}, q, \delta_T, \delta_e\}$ e:

$$k = \frac{\mathcal{M}_{\dot{w}}}{m - Z_{\dot{w}}} \quad \hat{k} = \frac{\hat{\mathcal{M}}_{\dot{w}}}{m - \hat{Z}_{\dot{w}}} \quad (5.54)$$

In cui m è la massa del velivolo, mentre I_{yy} è il momento di inerzia rispetto all'asse di beccheggio. Le matrici (5.51) e (5.52) sono fondamentali nello studio della dinamica longitudinale. In esse, oltre alle grandezze relative alla condizione iniziale e alle caratteristiche geomtriche e inerziali del velivolo, compaiono combinazioni delle derivate di stabilità.

Derivata	Formula di calcolo	Unità
\hat{X}_u	$\begin{cases} -\frac{\bar{q}_0 S}{m U_0} (2C_D _0 + M_0 \frac{\partial C_D}{\partial M} _0) \\ -\frac{\bar{q}_0 S}{m U_0} (3C_D _0 + C_L _0 \tan \Gamma_0 + M_0 \frac{\partial C_D}{\partial M} _0) \end{cases}$	per spinta costante per potenza costante
\hat{X}_w	$\frac{\bar{q}_0 S}{m U_0} (C_L _0 - C_{D_\alpha} _0)$	s ⁻¹
\hat{X}_w	≈ 0	-
\hat{X}_q	≈ 0	ms ⁻¹
\hat{X}_{δ_e}	≈ 0	ms ⁻²
\hat{X}_{δ_T}	$\begin{cases} \frac{\bar{q}_0 S}{m} (C_{T_{Fx}} + k_V/U_0^2) & \text{per spinta costante} \\ \frac{\bar{q}_0 S}{m} (C_{T_{Fx}} + k_V/U_0^3) & \text{per potenza costante} \end{cases}$	ms ⁻²
\hat{Z}_u	$-\frac{\bar{q}_0 S}{m U_0} \left[2C_L _0 + \frac{M_0^2}{1-M_0^2} C_L _0 \right]$	s ⁻¹
\hat{Z}_w	$-\frac{\bar{q}_0 S}{m U_0} (C_D _0 + C_{L_\alpha} _0)$	s ⁻¹
\hat{Z}_w	$= -\frac{1}{2} \frac{\rho_0 S \bar{c}}{2m} C_{L_\alpha} _0 = -\frac{1}{2\mu_0} C_{L_\alpha} _0 \approx 0$	-
\hat{Z}_q	$= -\frac{U_0}{2\mu_0} C_{L_q} _0 \approx 0$	ms ⁻¹
\hat{Z}_{δ_e}	$= -\frac{\bar{q}_0 S}{m} C_{L_{\delta_e}} _0 = -\frac{\bar{q}_0 S}{m} \eta_H \frac{S_H}{S} C_{L_{\alpha,H}} \tau_e$	ms ⁻²
$\hat{\mathcal{M}}_u$	$\frac{\bar{q}_0 S \bar{c}}{I_{yy} U_0} C_{M_u} _0 = \frac{\bar{q}_0 S \bar{c}}{I_{yy} U_0} M_0 C_{\mathcal{M}_M} _0$	m ⁻¹ s ⁻¹
$\hat{\mathcal{M}}_w$	$\frac{\bar{q}_0 S \bar{c}}{I_{yy} U_0} C_{M_\alpha} _0$	m ⁻¹ s ⁻¹
$\hat{\mathcal{M}}_w$	$= \frac{\rho_0 S \bar{c}^2}{4I_{yy}} C_{\mathcal{M}_\alpha} _0 = -2 \frac{\rho_0 S \bar{c}^2}{4I_{yy}} K \eta_H \frac{l_H}{\bar{c}} \frac{d\epsilon}{d\alpha} C_{L_{\alpha,H}}$	m ⁻¹
$\hat{\mathcal{M}}_q$	$= \frac{\rho_0 U_0 S \bar{c}^2}{4I_{yy}} C_{\mathcal{M}_q} _0 = -2 \frac{\rho_0 U_0 S \bar{c}^2}{4I_{yy}} K \eta_H \bar{v}_H \frac{l_H}{\bar{c}} C_{L_{\alpha,H}}$	s ⁻¹
$\hat{\mathcal{M}}_{\delta_e}$	$= \frac{\bar{q}_0 S \bar{c}}{I_{yy}} C_{\mathcal{M}_{\delta_e}} _0 = -\frac{\bar{q}_0 S \bar{c}}{I_{yy}} \eta_H \bar{v}_H C_{L_{\alpha,H}} \tau_e$	s ⁻²

Tabella 5.1: Derivate di stabilità dimensionali del moto longitudinale

Approssimazione di corto periodo

Le approssimazione di corto periodo consistono nell'eliminare dal vettore di stato le variabili u e θ che nel breve periodo non manifestano significative oscillazioni, a differenza di w e q . Riscrivendo in queste ipotesi le equazioni della dinamica e ricavando nuovamente il polinomio caratteristico, si perviene alle:

$$\zeta_{SP} = -\frac{1}{2\omega_{n,SP}} \left(\frac{\hat{Z}_\alpha}{U_0} + \hat{\mathcal{M}}_q + \hat{\mathcal{M}}_\alpha \right) \quad (5.55)$$

$$\omega_{n,SP} = \sqrt{\frac{\hat{Z}_\alpha}{U_0} \hat{\mathcal{M}}_q - \hat{\mathcal{M}}_\alpha}$$

Queste ultime sono note come formule di *short period approximation*. Tenendo conto degli ordini di grandezze delle derivate aerodinamiche dei velivoli di configurazione tradizionale, le ultime equazioni possono ulteriormente semplificate grazie a:

$$2\zeta\omega_{n,SP} \approx -\hat{\mathcal{M}}_q \quad (5.56)$$

$$\omega_{n,SP}^2 \approx -\hat{\mathcal{M}}_w U_0 = -\hat{\mathcal{M}}_\alpha$$

Che conducono alle:

$$\begin{aligned}\zeta_{SP} &= \frac{-\hat{\mathcal{M}}_q}{2\sqrt{-\hat{\mathcal{M}}_w U_0}} = \frac{-\hat{\mathcal{M}}_q}{2\sqrt{-\hat{\mathcal{M}}_\alpha}} \\ \omega_{n,SP} &= \sqrt{-\hat{\mathcal{M}}_w U_0} = \sqrt{-\hat{\mathcal{M}}_\alpha}\end{aligned}\quad (5.57)$$

Queste ultime sono note come le *short period coarse approximation*

Approssimazione di lungo periodo

Le approssimazioni di lungo periodo eliminano dal vettore di stato le variabili w e q . In forma matriciale si ottiene:

$$\begin{Bmatrix} u \\ q \end{Bmatrix} = \begin{bmatrix} \frac{\hat{\mathcal{M}}_u U_0 - \hat{\mathcal{M}}_q \hat{\mathcal{Z}}_u}{-\hat{\mathcal{M}}_w U_0 + \hat{\mathcal{M}}_q \hat{\mathcal{Z}}_w} \\ \frac{\hat{\mathcal{M}}_u \hat{\mathcal{Z}}_u - \hat{\mathcal{M}}_u \hat{\mathcal{Z}}_w}{-\hat{\mathcal{M}}_w U_0 + \hat{\mathcal{M}}_q \hat{\mathcal{Z}}_w} \end{bmatrix} u \quad (5.58)$$

Che diventano la:

$$\begin{Bmatrix} \dot{u} \\ \dot{\theta} \end{Bmatrix} = \begin{bmatrix} \hat{X}_u + \hat{X}_w \frac{\hat{\mathcal{M}}_u U_0 - \hat{\mathcal{M}}_q \hat{\mathcal{Z}}_u}{-\hat{\mathcal{M}}_w U_0 + \hat{\mathcal{M}}_q \hat{\mathcal{Z}}_w} & -g \\ \frac{\hat{\mathcal{M}}_u \hat{\mathcal{Z}}_u - \hat{\mathcal{M}}_u \hat{\mathcal{Z}}_w}{-\hat{\mathcal{M}}_w U_0 + \hat{\mathcal{M}}_q \hat{\mathcal{Z}}_w} & 0 \end{bmatrix} \begin{Bmatrix} u \\ \theta \end{Bmatrix} \quad (5.59)$$

La (5.59) viene detta *phugoid full approximation*, il suo vettore di stato è:

$$x_{Lon, P_H} = \begin{Bmatrix} u \\ \theta \end{Bmatrix} \quad (5.60)$$

Il modello si semplifica ulteriormente osservando che tipicamente si ha:

$$|\hat{\mathcal{M}}_u \hat{\mathcal{Z}}_w| \ll |\hat{\mathcal{M}}_w \hat{\mathcal{Z}}_u|, \quad |\hat{\mathcal{M}}_w U_0| \gg |\hat{\mathcal{M}}_q \hat{\mathcal{Z}}_w|, \quad |\hat{\mathcal{M}}_u X_w / \hat{\mathcal{M}}_w| \ll |\hat{X}_u| \quad (5.61)$$

ottenendo la seguente *phugoid coarse approximation*

$$\begin{Bmatrix} \dot{u} \\ \dot{\theta} \end{Bmatrix} = \begin{bmatrix} \hat{X}_u & -g \\ \frac{-\hat{\mathcal{Z}}_u}{U_0} & 0 \end{bmatrix} \begin{Bmatrix} u \\ \theta \end{Bmatrix} \quad (5.62)$$

L'equazione caratteristica è:

$$s^2 + 2\zeta_{P_H} \omega_{n,P_H} s + \omega_{n,P_H}^2 \quad (5.63)$$

con

$$\begin{aligned}2\zeta \omega_{n,P_H} &\approx -\hat{X}_u \\ \omega_{n,P_H}^2 &\approx -\frac{g \hat{Z}_u}{U_0}\end{aligned}\quad (5.64)$$

Per voli a bassi numeri di Mach è possibile ulteriormente approssimare per ottenere le seguenti:

$$\begin{aligned}\zeta_{P_H} &= \frac{-\hat{X}_u}{2\zeta \omega_{n,P_H}} = \frac{\frac{q_0 S}{m U_0} 2 C_D |_0}{2\sqrt{2} \frac{g}{U_0}} = \frac{1}{\sqrt{2}} \frac{q_0 S C_D |_0}{q_0 S C_L |_0} = \frac{1}{\sqrt{2}} \frac{C_D |_0}{C_L |_0} \\ \omega_{n,P_H} &= \sqrt{\frac{g \hat{Z}_u}{U_0}} = \sqrt{2} \frac{g}{U_0}\end{aligned}\quad (5.65)$$

5.2 Esercizio 1 (16.1-2)

L'obiettivo di questo esercizio consiste nel ricavare la matrice \mathbf{A}_{Lon} , calcolarne gli autovalori e autovettori, per poi diagrammare le risposte modali e i fasori degli autovettori. Si verifica l'esistenza dei modi di corto periodo e di fugoide e successivamente si calcolano le caratteristiche di ciascun modo in questi termini:

- coefficiente di smorzamento ζ ;
- pulsazione naturale ω_n ;
- periodo T;
- tempo di dimezzamento $t_{1/2}$;
- numero di cicli fino al tempo di dimezzamento $N_{1/2}$.

In questo esercizio si farà riferimento al velivolo Boeing 747. Alcuni dati che rimarranno costanti durante i vari esercizi sono riportati nella Tabella 5.2.

superficie alare	S	510,97 m ²
corda media aerodinamica	c	8.32 m
margine statico	SM = (X_N-X_cg)/c	0.22

Tabella 5.2: Caratteristiche geometriche Boeing 747

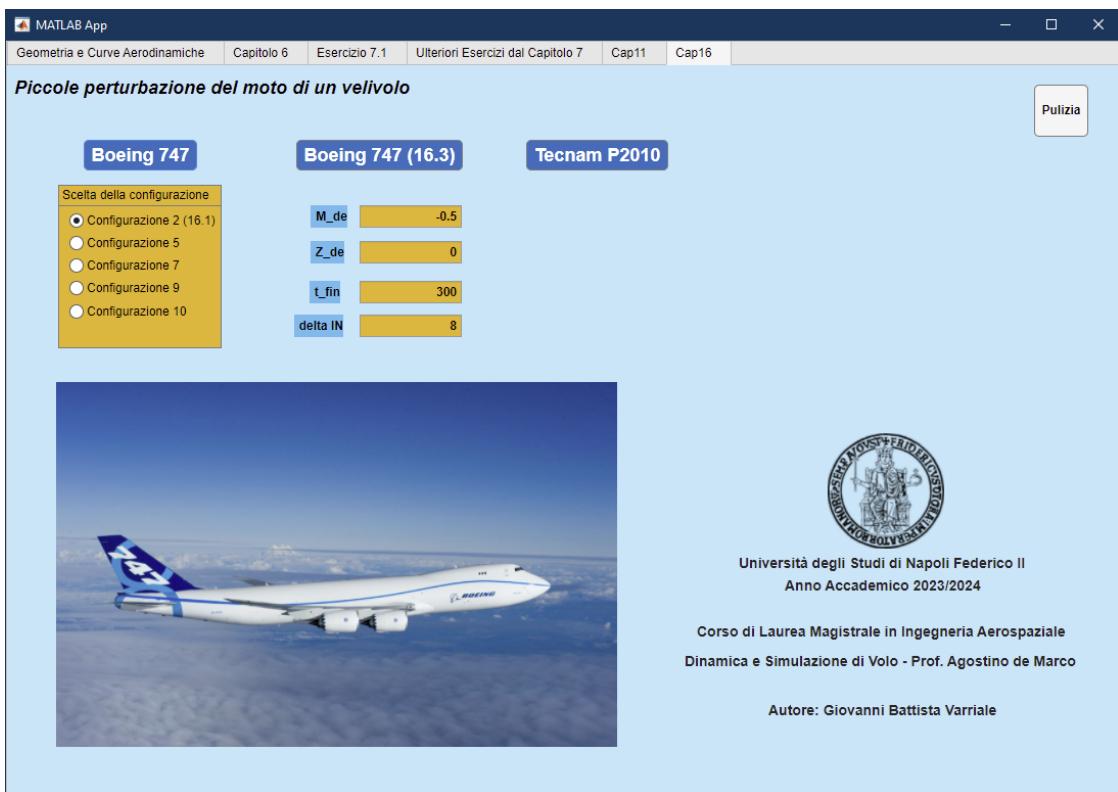


Figura 5.1: Vista del capitolo 16 in APP

5.2.1 Input

In Tabella 5.3 vengono mostrate le varie configurazioni proposte per il velivolo Boeing 747. E' possibile scegliere all'interno dell'APP quale delle varie condizioni si vuole analizzare attraverso un *radio menu*, per poi clickare sull'apposito pulsante per avviare il listato. Per farlo è stato necessario anche rendere la suddetta Tabella un file .xlsx leggibile da MATLAB. Di seguito il codice:

```

1 function [] = func162(l)
2 % -----
3 % Esercizi Capitolo 16
4 % -----
5 % Esercizio N°1-2
6 %
7 diary('ex16_1-2.txt')
8 diary on
9 COND=table2array(readtable("BoeingDataCorrect.xlsx"));
10 Cond=str2double(COND(:,1));
11 %----- DEFINIZIONI GRANDEZZE DA INSERIRE NELLE MATRICI %
12 %----- %
13 %----- %
14
15 Weight = convforce(Cond(4),'lbf','N');% Peso [N]
16 g_0 = 9.81; % accelerazione di gravità [m/s^2]
17 mass = Weight/g_0; % massa [kg]
18 h_0 = Cond(1); % Altezza iniziale [m]
19 [T_0,a_0,p_0,rho_0] = atmosisa(h_0); % funzione ISA
20 Mach_0 = Cond(2); % Mach iniziale
21 U_0 = Mach_0*a_0; % Velocità iniziale [m/s]
22
23 % Momento di inerzia
24 SLUGFT2toKGM2 = convmass(1,'slug','kg')... % Si passa da [slug*ft^2] a
25 *(convlength(1,'ft','m')^2); % [kg*m^2]
26 Iyy = Cond(5)*10^-6*SLUGFT2toKGM2; % Momento di inerzia
27 % longitudinale [kg*m^2]
28
29 % Grandezze geometriche
30 S = 5500*(convlength(1,'ft','m'))^2; % Superficie alare [m^2]
31 cbar = convlength(27.3,'ft','m'); % mac [m]
32 qbar_0 = 0.5*rho_0*(U_0^2); % pressione dinamica iniziale [N/m^2]
33 mu_0 = mass/(0.5*rho_0*S*cbar);
34 Gamma_0 = 0.0; % angolo di salita iniziale [rad]
35
36 % Coefficiente aerodinamici forze
37 C_L = Cond(6);
38 C_D = Cond(7);
39 C_L_Mach = Cond(15);
40 C_L_alpha = Cond(8); % [1/rad]
41 C_L_alpha_dot = Cond(11); % [1/rad]
42 C_L_q = Cond(13); % [1/rad]
43 C_D_alpha = Cond(9); % [1/rad]
44 C_D_alpha_dot = 0.0; % [1/rad]
45 C_D_q = 0.0; % [1/rad]
46 C_D_Mach = Cond(16);
47
48 % Coefficiente aerodinamici momenti
49 SM_0 = 0.22; % Margine statico iniziale
50 C_m_alpha_0 = Cond(10); % [1/rad]
51 SM = 0.22; % Margine statico
52 C_m_alpha = C_m_alpha_0*(SM/SM_0); % [1/rad]
53 C_m_alpha_dot = Cond(12); % [1/rad]
54 C_m_q = Cond(14); % [1/rad]
55 C_m_Mach = Cond(17);
56 C_L_de = Cond(18); % [1/rad]
57 C_m_de = Cond(19); % [1/rad]
58
59 % Derivate di stabilità
60 X_u = -(qbar_0*S/(mass*U_0))... % [1/s]
61 *(2*C_D + Mach_0*C_D_Mach); % constant thrust
62 X_w = (qbar_0*S/(mass*U_0))... % [1/s]
63 *(C_L - C_D_alpha); X_wdot = 0;
64 X_q = 0;
65 X_de = 0;
66 X_dT = 0;
67
68 Z_u = -(qbar_0*S/(mass*U_0))*( ... % [1/s]
69 2*C_L+(Mach_0^2/(1-Mach_0^2))...
70 *C_L_Mach); % constant thrust
71 Z_w = -(qbar_0*S/(mass*U_0))... % [1/s]
72 *(C_D + C_L_alpha);
73 Z_wdot = -(1/(2*mu_0))*C_L_alpha_dot;
74 Z_q = -(U_0/(2*mu_0))*C_L_q; % [m/s]
75 Z_de = 0;
76 Z_dT = 0;
77 M_u = (qbar_0*S*cbar/(Iyy*U_0))... % [1/(ms)]
78 *Mach_0*C_m_Mach;
79 M_w = (qbar_0*S*cbar/(Iyy*U_0))... % [1/(ms)]
80 *C_m_alpha;
81 M_wdot = (rho_0*S*(cbar^2)/(4*Iyy))...% [1/m]
82 *C_m_alpha_dot;
83 M_q = (rho_0*U_0*S...
84 *(cbar^2)/(4*Iyy))*C_m_q;
85 M_de = 0;
86 M_dT = 0;
87
88 k_hat = M_wdot/(1-Z_wdot);
89
```

```

90
91 %%%%%%%%
92 %           DEFINIZIONI MATRICI %
93 %%%%%%%%
94
95 % Definizione matrice A_longitudinale. Si veda l'eq. (16.147b)
96 A_lon(1,1) = X_u;
97 A_lon(1,2) = X_w; A_lon(1,3) = 0;
98 A_lon(1,4) = -g_0*cos(Gamma_0); % Prima riga
99
100 A_lon(2,1) = Z_u/(1 - Z_wdot);
101 A_lon(2,2) = Z_w/(1 - Z_wdot);
102 A_lon(2,3) = (Z_q + U_0)/(1 - Z_wdot);
103 A_lon(2,4) = -g_0*sin(Gamma_0)/(1 - Z_wdot); % Seconda riga
104
105 A_lon(3,1) = M_u + k_hat*Z_u;
106 A_lon(3,2) = M_w + k_hat*Z_w;
107 A_lon(3,3) = M_q + k_hat*(Z_q+U_0);
108 A_lon(3,4) = -k_hat*g_0*sin(Gamma_0); % Terza riga
109
110 A_lon(4,1) = 0;
111 A_lon(4,2) = 0;
112 A_lon(4,3) = 1;
113 A_lon(4,4) = 0; % Quarta riga
114
115 % Matrice input. Si veda l'eq. (16.147c)
116 B_lon = [ ...
117 X_de/mass, X_dT/mass; ...
118 Z_de/(mass-Z_wdot), Z_dT/(mass-Z_wdot); ...
119 (M_de + Z_de*M_wdot/(mass-Z_wdot))/Iyy, ...
120 (M_dT + Z_dT*M_wdot/(mass-Z_wdot))/Iyy; ...
121 0, 0];
122
123 % Matrici output
124 C_lon = eye(3,4);
125 D_lon = zeros(3,2);
126
127 %%%%%%%%
128 %           SOLUZIONE PROBLEMA AGLI AUTOVALORI E AUTOVETTORI %
129 %%%%%%%%
130
131 % symbolic determinant --> characteristic equation
132 syms lambda
133
134 characteristic_sym_polyonial_lon = det(A_lon-lambda*eye(4));
135 char_poly_lon = sym2poly(characteristic_sym_polyonial_lon);
136
137 a1 = char_poly_lon(1);
138 a2 = char_poly_lon(2);
139 a2 = a2/a1;
140 a3 = char_poly_lon(3);
141 a3 = a3/a1; a4 = char_poly_lon(4);
142 a4 = a4/a1;
143 a5 = char_poly_lon(5);
144 a5 = a5/a1;
145 a1 = 1;
146
147 % Autovettori e autovali
148 [V,D] = eig(A_lon);% La funzione eig accetta in input una matrice
149 % quadrata e restituisce una matrice di autovettori
150 % V e autovalori D. Quest'ultima ha gli autovalori
151 % sulla diagonale.
152
153 W = inv(V); % L'inversa di V ha gli autovettori sinistri sulle righe
154
155 V_SP = V(:,1); % Autovettori di short period
156 V_SP = V_SP/V_SP(4); % normalizzo rispetto V_SP(4)
157 V_SP(1,1) = V_SP(1,1)/U_0;
158 V_SP(2,1) = V_SP(2,1)/U_0;
159 V_SP(3,1) = V_SP(3,1)/(2*U_0/cbar);
160
161 V_Ph = V(:,3); % Autovettori fugoide
162 V_Ph = V_Ph/V_Ph(4); % normalizzo rispetto V_Ph(4)
163 V_Ph(1,1) = V_Ph(1,1)/U_0;
164 V_Ph(2,1) = V_Ph(2,1)/U_0;
165 V_Ph(3,1) = V_Ph(3,1)/(2*U_0/cbar);
166
167 % Trasformazione delle variabili di stato
168 syms a11 a12 a13 a14 a21 a22 a23 a24 a31 a32 a33 a34 a41 a42 a43 a44
169 AA = [ ...
170 a11, a12, a13, a14; ...
171 a21, a22, a23, a24; ...
172 a31, a32, a33, a34; ...
173 a41, a42, a43, a44];
174
175 % Matrice di trasformazione. Si vedano le eq. (16.208)-(16.210)
176 syms U0 Cbar
177 TT = diag([1/U0,1/U0,Cbar/(2*U0),1]);
178 AA1 = TT*AA*inv(TT); % Matrice A cappello. Si veda l'eq. (16.210)

```

```

179
180 % Si valuta la matrice AA1 sostituendo i numeri ai simboli
181 A_loni = ...
182     double( ... % rende il risultato di tipo double
183     subs( ... % sostituisco
184         AA1, ... % in AA1
185         [a11, a12, a13, a14, ... % i valori aij
186         a21, a22, a23, a24, ...
187         a31, a32, a33, a34, ...
188         a41, a42, a43, a44 ...
189         U0, Cbar], ... % etc, con i seguenti numeri
190         [A_lon(1,1), A_lon(1,2), A_lon(1,3), A_lon(1,4), ...
191         A_lon(2,1), A_lon(2,2), A_lon(2,3), A_lon(2,4), ...
192         A_lon(3,1), A_lon(3,2), A_lon(3,3), A_lon(3,4), ...
193         A_lon(4,1), A_lon(4,2), A_lon(4,3), A_lon(4,4), ...
194         U_0, cbar] ...
195     ) ...
196 );
197
198 % autovettori destri (colonne) and autovalori
199 [V1,D1] = eig(A_loni);
200
201 % autovettori sinistri (righe)
202 W1 = inv(V1);
203
204 % Creo una rappresentazione dello spazio degli stati tramite la struttura
205 % dati sys
206 sys = ss( ... % ss = spazio degli stati
207     A_loni, ... % A
208     zeros(size(A_loni,1),1), ... % B = 0 perchè voglio studiare
209     % senza forzante)
210     eye(4,4), ... % C
211     zeros(4,1) ... % D
212 );
213
214 sys1 = ss( ... %caso non normalizzato rispetto theta
215     A_lon , ... % A
216     zeros(size(A_lon ,1) ,1) , ... % B
217     eye(4,4) , ... % C
218     zeros(4,1) ... % D
219 );
220
221 % Short-Period/Phugoid, autovettori sinistri normalizzati
222 V1SP = V1(:,1); % SP = short period
223 V1SP = V1SP/V1SP(4,1);
224
225 V1Ph = V1(:,3); % PH = phugoid
226 V1Ph = V1Ph/V1Ph(4,1);
227
228 % risposta temporale prendendo lo stato iniziale coincidente con gli
229 % autovettori originali si vedano i paragrafi (16.6.1) e (16.7.4)
230 x_init_1 = real(V1SP); % Uso come condizione iniziale l'autovalore di
231 % corto periodo per eccitare solo questo modo
232 [y1,t1,x1] = initial(sys,x_init_1);
233
234 x_init_3 = real(V1Ph); % Uso come condizione iniziale l'autovalore di
235 % fugoide per eccitare solo questo modo
236 [y3,t3,x3] = initial(sys,x_init_3);
237
238 x_init_alpha = [ 0, U_0, 0, 0];
239 [ya ,ta ,xa] = initial(sys1, x_init_alpha);
240 % La funzione initial calcola la risposta del sistema con date condizioni
241 % iniziali assegnate
242
243 %%%%%%%%%%%%%% CARATTERISTICHE MODALI %%%%%%
244 % Di seguito vengono calcolate le caratteristiche di ciascun modo, ossia
245 % il coefficiente di smorzamento "zita", la pulsazione naturale
246
247 % "omega_n", il periodo "Period", tempo di dimezzamento "t_half" (cioè
248 % t_1/2) e numero di cicli fino al tempo di dimezzamento "N_half"
249 % Definizioni delle funzioni anonime
250
251 zeta = @(sigma,omega) ... % vedi eq. (16.203a)
252 sqrt(1/(1+(omega/sigma)^2));
253
254 omega_n = @(sigma,omega) ... % vedi eq. (16.203b)
255 -sigma/zeta(sigma,omega);
256
257 Period = @(sigma,omega) ... % vedi eq. (16.203c)
258 2*pi/(omega_n(sigma,omega)*sqrt(1-(zeta(sigma,omega))^2));
259
260 t_half = @(sigma,omega) ... % vedi eq. (16.203d)
261 log(2)/(omega_n(sigma,omega)*zeta(sigma,omega));
262
263 N_half = @(sigma,omega) ... % vedi eq. (16.203e)

```

```

267 t_half(sigma_omega)/Period(sigma_omega);
268 % Short Period
269 lambda_SP = D(1,1);
270 sigma_SP = real(lambda_SP);
271 omega_SP = imag(lambda_SP);
272 zeta_SP = zeta(sigma_SP,omega_SP);
273 omega_n_SP = omega_n(sigma_SP,omega_SP);
274 T_SP = Period(sigma_SP,omega_SP);
275 t_half_SP = t_half(sigma_SP,omega_SP);
276 N_half_SP = N_half(sigma_SP,omega_SP);
277 % Phugoid
278 lambda_Ph = D(3,3);
279 sigma_Ph = real(lambda_Ph);
280 omega_Ph = imag(lambda_Ph);
281 zeta_Ph = sqrt(1/(1+(omega_Ph/sigma_Ph)^2));
282 omega_n_Ph = -sigma_Ph/zeta_Ph;
283 T_Ph = 2*pi/(omega_n_Ph*sqrt(1-zeta_Ph^2));
284 t_half_Ph = log(2)/(omega_n_Ph*zeta_Ph);
285 N_half_Ph = t_half_Ph/T_Ph;
286
287
288 % Mostro a video
289 disp ('Caratteristiche modo di corto periodo');
290 disp(['sigma_SP = ', num2str(sigma_SP), ' [N*s/kg*m]']);
291 disp(['omega_SP = ', num2str(omega_SP), ' [1/s]']);
292 disp(['zeta_SP = ', num2str(zeta_SP)]);
293 disp(['omega_n_SP = ', num2str(omega_n_SP), ' [1/s]']);
294 disp(['T_SP = ', num2str(T_SP), ' [s]']);
295 disp(['t_half_SP = ', num2str(t_half_SP), ' [s]']);
296 disp(['N_half_SP = ', num2str(N_half_SP)]);
297
298 disp (' ');
299 disp ('Caratteristiche modo di fugoide');
300 disp(['sigma_Ph = ', num2str(sigma_Ph), ' [N*s/kg*m]']);
301 disp(['omega_Ph = ', num2str(omega_Ph), ' [1/s]']);
302 disp(['zeta_Ph = ', num2str(zeta_Ph)]);
303 disp(['omega_n_Ph = ', num2str(omega_n_Ph), ' [1/s]']);
304 disp(['T_Ph = ', num2str(T_Ph), ' [s]']);
305 disp(['t_half_Ph = ', num2str(t_half_Ph), ' [s]']);
306 disp(['N_half_Ph = ', num2str(N_half_Ph), ' [s]']);
307
308 %% plot
309 % complex axis
310 fig1=figure("NumberTitle","off","Name","Short Period")
311
312 compass(real(V_SP(2)),imag(V_SP(2)),'-g'); hold on;
313 compass(real(V_SP(1)),imag(V_SP(1)),'-r'); hold on;
314 compass(real(V_SP(3)),imag(V_SP(3)),'-k'); hold on;
315 compass(real(V_SP(4)),imag(V_SP(4)),'-b'); hold on;
316
317
318 fig2=figure("NumberTitle","off","Name","Phugoid")
319
320 compass(real(V_Ph(1)),imag(V_Ph(1)),'-r'); hold on;
321 compass(real(V_Ph(2)),imag(V_Ph(2)),'-g'); hold on;
322 compass(real(V_Ph(3)),imag(V_Ph(3)),'-k'); hold on;
323 compass(real(V_Ph(4)),imag(V_Ph(4)),'-b'); hold on;
324
325 % plot time
326 fig3=figure("NumberTitle","off","Name","Risposta Libera - Corto Periodo")
327 plot...
328 t1,y1(:,1),'-',...
329 t1,y1(:,2),'-',...
330 t1,(cbar/(2*U_0))*y1(:,3),'-.',...
331 t1,y1(:,4),':',...
332 );
333 axis([0 10 -1 1.0]); grid on;
334 ylabel("Perturbation States u,w,q");
335 xlabel('time(s)');
336 legend('u/U_0','w/U_0','q c/(2 U_0)', '\theta')
337
338
339 fig4=figure("NumberTitle","off","Name","Risposta Libera - Fugoide")
340 plot...
341 t3,y3(:,1),'-',...
342 t3,y3(:,2),'-',...
343 t3,(cbar/(2*U_0))*y3(:,3),'-.',...
344 t3,y3(:,4),':',...
345 );
346 axis([0 600 -max(abs(y3(:,1))) max(abs(y3(:,1)))]); grid on;
347 ylabel("Perturbation States u,w,q");
348 xlabel('time(s)');
349 legend('u/U_0','w/U_0','q c/(2 U_0)', '\theta')
350 diary off
351 saveas(fig1,"Figura_16_1_1.pdf");
352 saveas(fig2,"Figura_16_1_2.pdf");
353 saveas(fig3,"Figura_16_1_3.pdf");
354 saveas(fig4,"Figura_16_1_4.pdf");
355
```

Condizione	2	5	7	9	10
h (ft)	0	20000	20000	40000	40000
M	0.25	0.50	0.80	0.80	0.90
α (deg)	5.70	6.80	0.00	4.60	2.40
W (lbf)	564000	636640	636640	636640	636640
I_{yy} (slug $ft^2 \times 10^6$)	32.30	33.10	33.10	33.10	33.1
C_L	1.10	0.68	0.27	0.66	0.52
C_D	0.10	0.04	0.02	0.04	0.04
C_{L_α}	5.70	4.67	4.24	4.92	5.57
C_{D_α}	0.66	0.37	0.08	0.43	0.53
C_{M_α}	-1.26	-1.15	-0.63	-1.03	-1.61
$C_{L_{\dot{\alpha}}}$	6.70	6.53	5.99	5.91	5.53
$C_{M_{\dot{\alpha}}}$	-3.20	-3.35	-5.40	-6.41	-8.82
C_{L_q}	5.40	5.13	5.01	6.00	6.94
C_{M_q}	-20.80	-20.70	-20.50	-24.00	-25.10
C_{L_M}	0	-0.09	0.11	0.21	-0.28
C_{D_M}	0	0	0.01	0.03	0.24
C_{M_M}	0	0.12	-0.12	0.17	-0.11
$C_{L_{\delta_e}}$	0.338	0.356	0.270	0.367	0.300
$C_{M_{\delta_e}}$	-1.34	-1.43	-1.06	-1.45	-1.20

Tabella 5.3: Caratteristiche inerziali e derivate aerodinamiche longitudinali di un Boeing 747 in diverse condizioni di volo nominali. Gli angoli di attacco sono misurati rispetto alla retta di riferimento della fusoliera e le derivate angolari in rad^{-1}

5.2.2 Risultati

Vengono ora riportati i risultati per le configurazioni assegnate.

Corto periodo	Conf 2	Conf 5	Conf 7	Conf 9	Conf 10
$\sigma_{SP} [N \text{ s}/(kg \text{ m})]$	-0.55137	-0.45672	-0.72557	-0.36804	-0.4651
$\omega_{SP} [1/s]$	0.68779	0.91191	1.0477	0.87997	1.242
ζ_{SP}	0.62548	0.44781	0.56932	0.38585	0.3507
$\omega_{n,SP} [1/s]$	0.88151	1.0199	1.2745	0.95383	1.3262
$T_{SP} [s]$	9.1353	6.8901	5.9969	7.1402	5.059
$t_{1/2,SP} [s]$	1.2571	1.5177	0.95531	1.8834	1.4903
$N_{1/2,SP}$	0.13761	0.22027	0.1593	0.26377	0.29459

Fugoide	Conf 2	Conf 5	Conf 7	Conf 9	Conf 10
$\sigma_{Ph} [N \text{ s}/(kg \text{ m})]$	-0.001576	-0.0020574	-0.0034714	-0.0031867	-0.045636
$\omega_{Ph} [1/s]$	0.1335	0.086598	0.018921	0.074122	0
ζ_{Ph}	0.011805	0.023751	0.18046	0.042953	1
$\omega_{n,Ph} [1/s]$	0.13351	0.086623	-0.019237	0.074191	0.045636
$T_{Ph} [s]$	47.0663	72.5556	332.0782	84.7678	∞
$t_{1/2,Ph} [s]$	439.8118	336.9092	199.6721	217.5094	15.1885
$N_{1/2,Ph}$	9.3445	4.6435	0.60128	2.5659	0

Tabella 5.4: Confronto tra i dati ottenuti nelle varie configurazioni

Configurazione 2

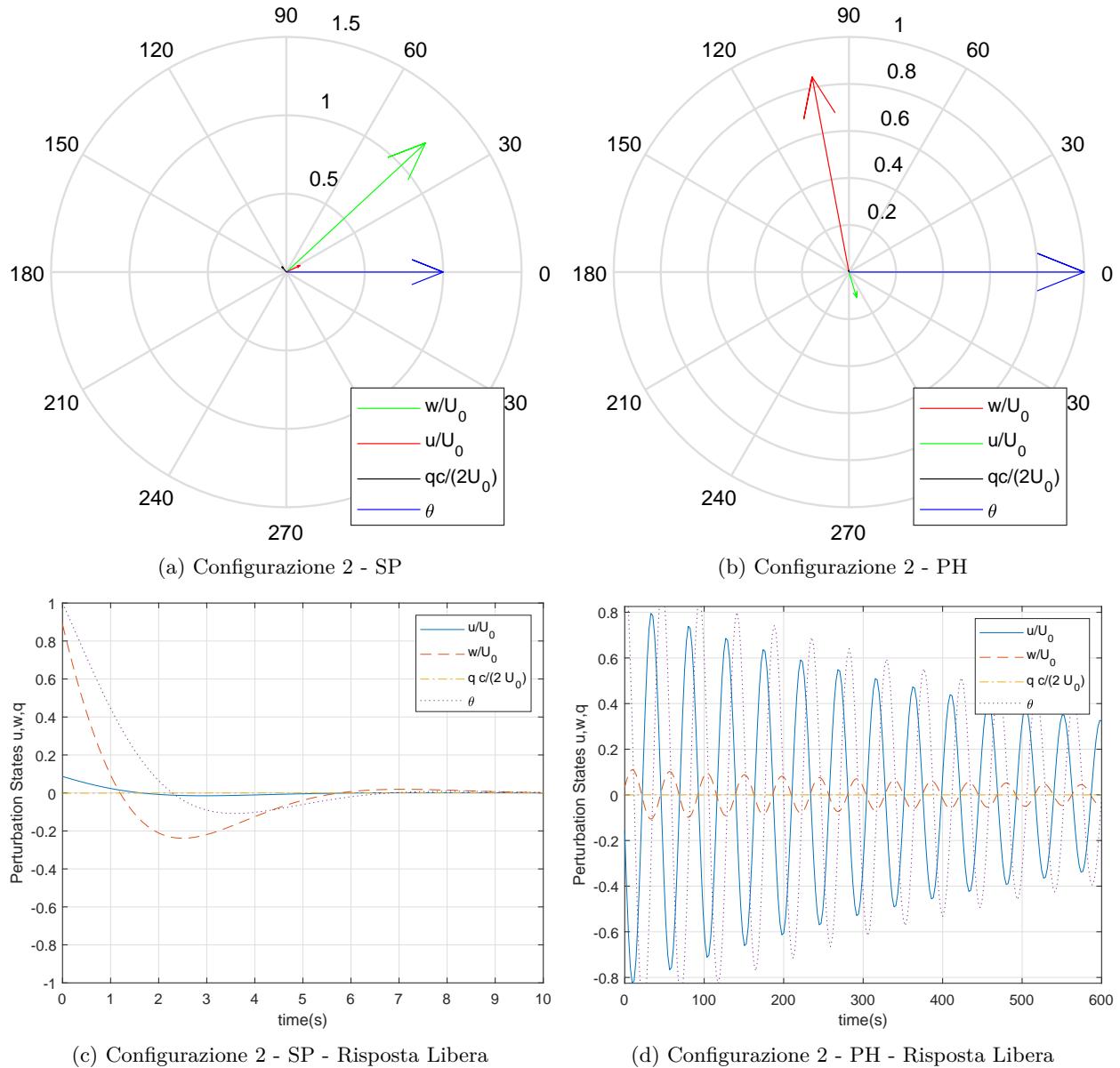
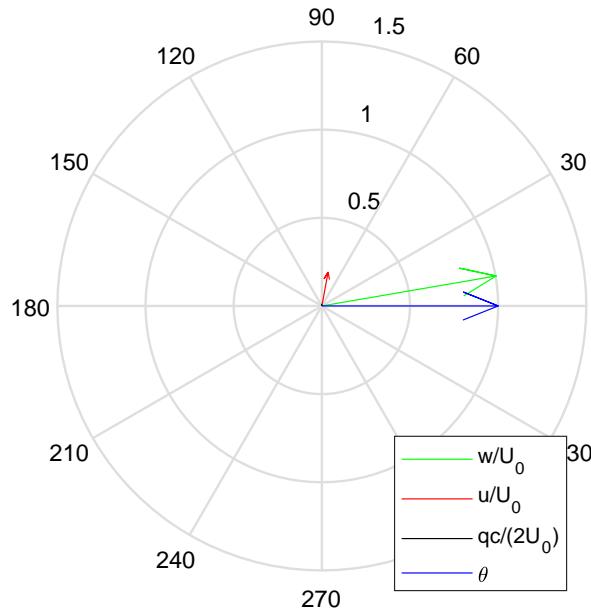
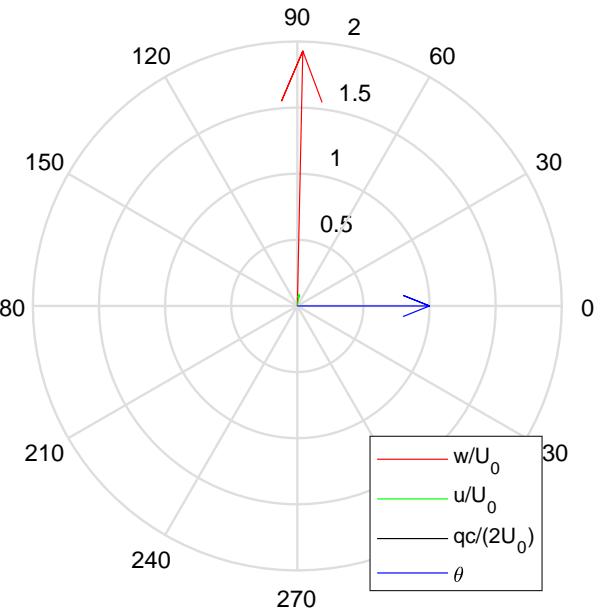


Figura 5.2: Esercizio 1 - Configurazione 2

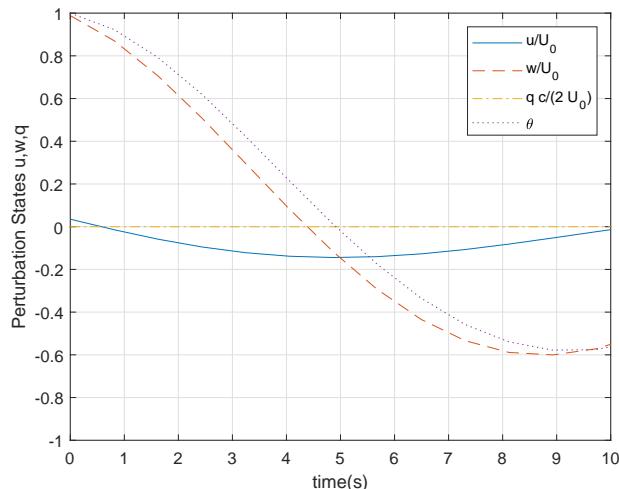
Configurazione 5



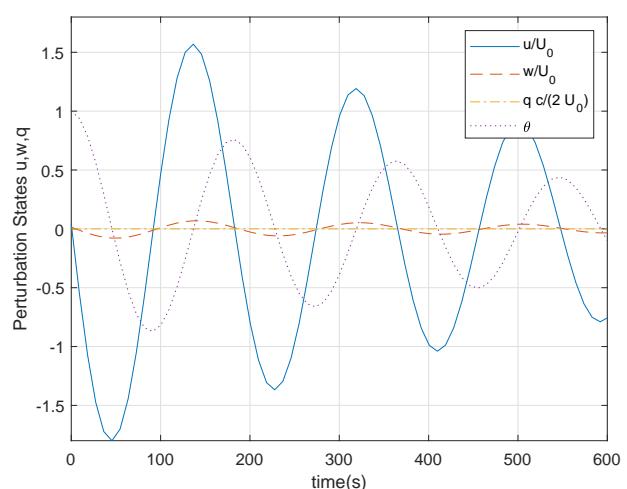
(a) Configurazione 5 - SP



(b) Configurazione 5 - PH



(c) Configurazione 5 - SP - Risposta Libera



(d) Configurazione 5 - PH - Risposta Libera

Figura 5.3: Esercizio 1 - Configurazione 5

Configurazione 7

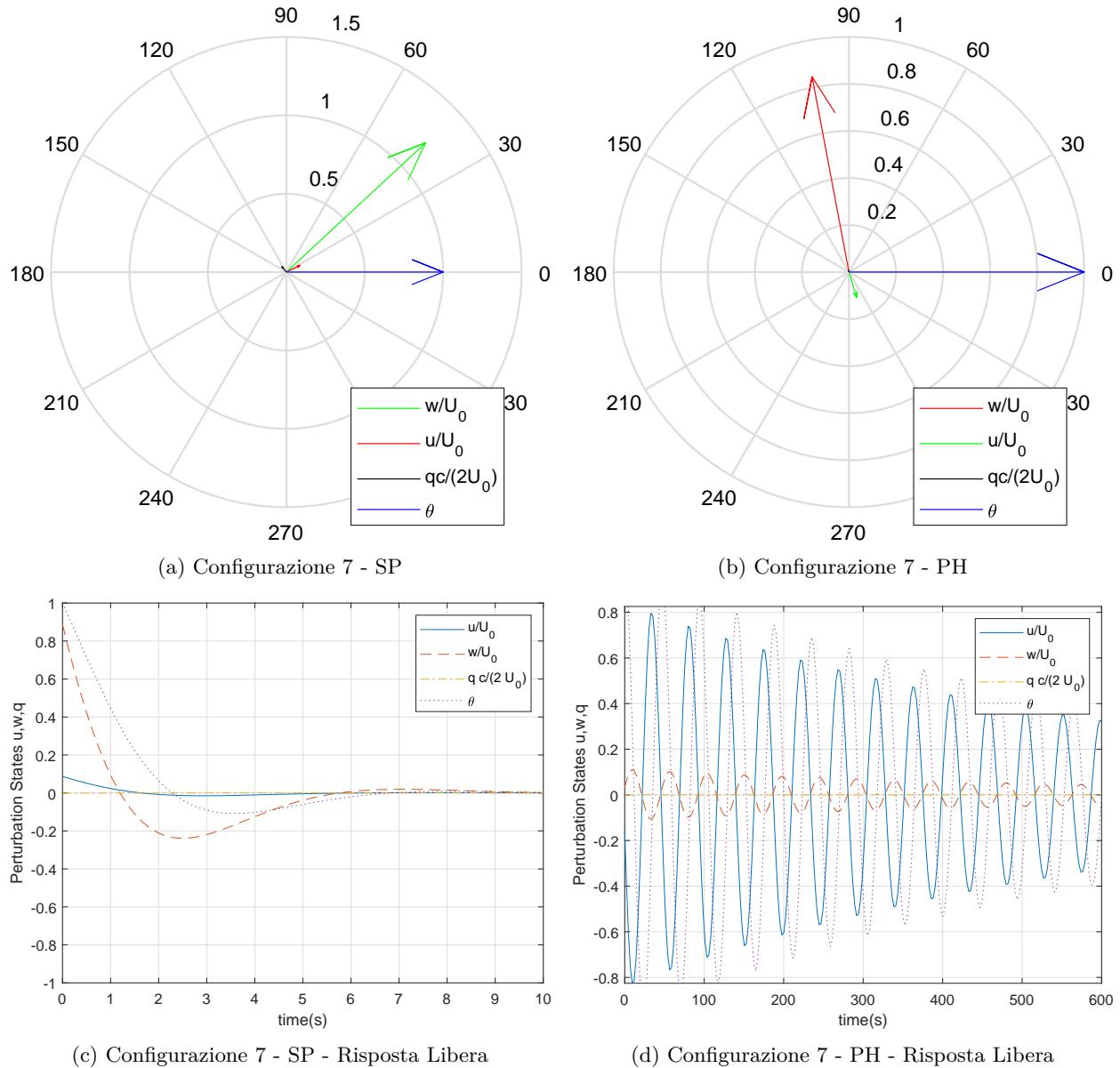
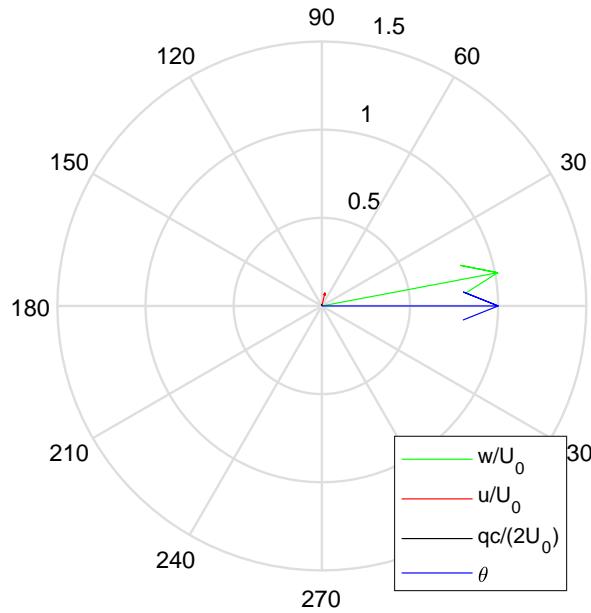
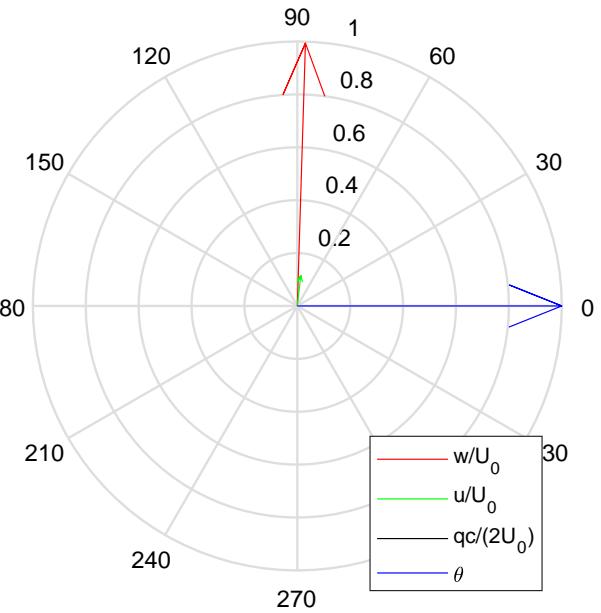


Figura 5.4: Esercizio 1 - Configurazione 7

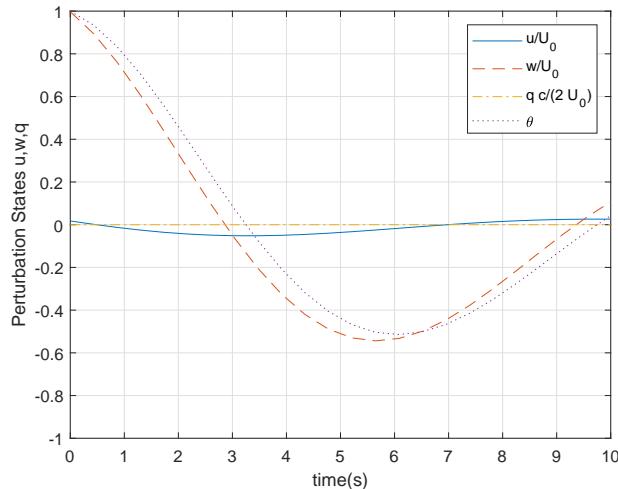
Configurazione 9



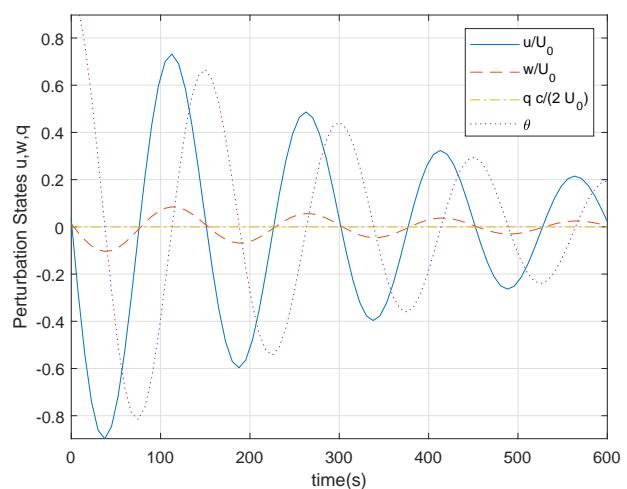
(a) Configurazione 9 - SP



(b) Configurazione 9 - PH



(c) Configurazione 9 - SP - Risposta Libera



(d) Configurazione 9 - PH - Risposta Libera

Figura 5.5: Esercizio 1 - Configurazione 9

Configurazione 10

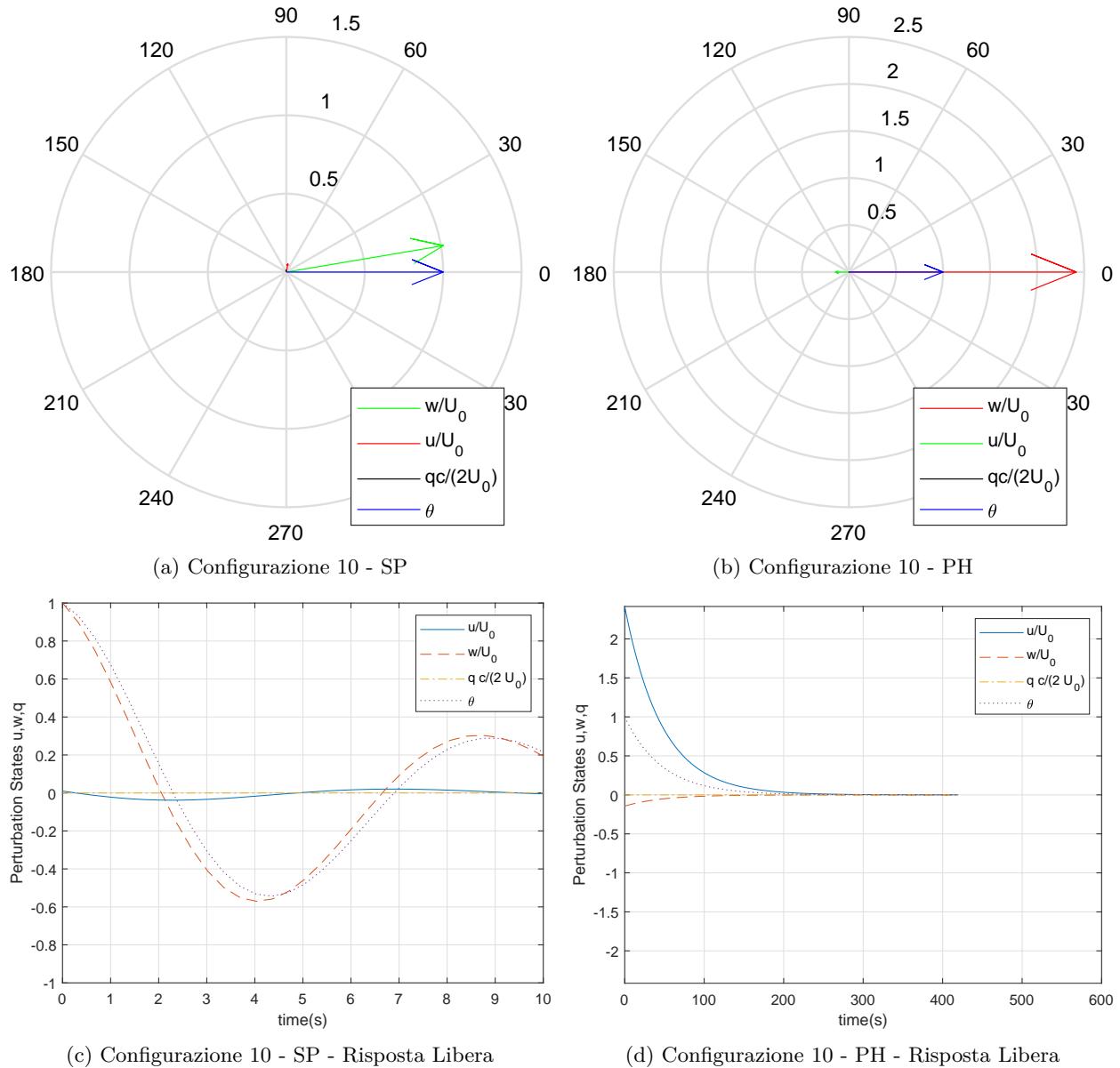


Figura 5.6: Esercizio 1 - Configurazione 10

5.3 Esercizio 2 (16.3)

Si analizza ora il caso in cui la risposta non è libera ma forzata, cioè con vettore u_{Lon} non nullo. Il velivolo considerato è ancora un Boeing 747. Si risolverà il seguente problema:

$$\begin{cases} \dot{x}_{Lon}(0) = A_{Lon}x_{Lon} + B_{Lon}u_{Lon} \\ x_{Lon}(0) = 0 \\ u_{Lon} = [\delta_e(t), 0]^T \quad \text{per } 0 \leq t \leq t_f \end{cases} \quad (5.66)$$

Occorre quindi costruire entrambe le matrici A_{Lon} e B_{Lon} e assegnare una legge dell'equilibratore $\delta_e(t)$. Viene utilizzata poi la funzione *lsim*.

In input dall'APP è possibile inserire:

- il tipo di configurazione (dal menu relativo al precedente esercizio);
- le derivate di stabilità $\hat{\mathcal{M}}_{\delta_e}$ e $\hat{\mathcal{Z}}_{\delta_e}$;
- il tempo finale di simulazione;
- l'angolo di deflessione δ_e

La legge temporale del comando è invece fissata.



Figura 5.7: Esercizio 2 - Input

Il listato relativo ai calcoli è il seguente:

```

1 function [] = func163(in163)
2 %
3 %   Esercizi Capitolo 16
4 %
5 %   Esercizio N°3
6 %
7 diary('ex16_3.txt')
8 diary on
9 M_de    = in163(1);
10 Z_de   = in163(2);
11 t_fin  = in163(3);
12 delta_IN=in163(5);
13 i=in163(4);
14 COND=table2array(readtable("BoeingDataCorrect.xlsx"));
15 Cond=str2double(COND(:,i));
16 %DEFINIZIONI GRANDEZZE DA INSERIRE NELLE MATRICI%
17 %DEFINIZIONI GRANDEZZE DA INSERIRE NELLE MATRICI%
18 %
19 Weight = convforce(Cond(4),'lbf','N');% Peso [N]
20 g_0    = 9.81;                         % accelerazione di gravità [m/s^2]
21 mass   = Weight/g_0;                   % massa [kg]
22 h_0    = Cond(1);                     % Altezza iniziale [m]
23 [T_0,a_0,p_0,rho_0] = atmosisa(h_0); % funzione ISA
24 Mach_0 = Cond(2);                    % Mach iniziale
25 U_0    = Mach_0*a_0;                  % Velocità iniziale [m/s]
26 %
27 % Momento di inerzia
28 
```

```

29 SLUGFT2toKGM2 = convmass(1,'slug','kg')... % Si passa da [slug*ft^2] a
30 *(convlength(1,'ft','m')^2); % [kg*m^2]
31 Iyy = Cond(5)*10^6*SLUGFT2toKGM2; % Momento di inerzia
32 % longitudinale [kg*m^2]
33
34 % Grandezze geometriche
35 S = 5500*(convlength(1,'ft','m'))^2; % Superficie alare [m^2]
36 cbar = convlength(27.3,'ft','m'); % mac [m]
37 qbar_0 = 0.5*rho_0*(U_0^2); % pressione dinamica iniziale [N/m^2]
38 mu_0 = mass/(0.5*rho_0*S*cbar);
39 Gamma_0 = 0.0; % angolo di salita iniziale [rad]
40
41 % Coefficiente aerodinamici forze
42 C_L = Cond(6);
43 C_D = Cond(7);
44 C_L_Mach = Cond(15);
45 C_L_alpha = Cond(8); % [1/rad]
46 C_L_alpha_dot = Cond(11); % [1/rad]
47 C_L_q = Cond(13); % [1/rad]
48 C_D_alpha = Cond(9); % [1/rad]
49 C_D_alpha_dot = 0.0; % [1/rad]
50 C_D_q = 0.0; % [1/rad]
51 C_D_Mach = Cond(16);
52
53 % Coefficiente aerodinamici momenti
54 SM_0 = 0.22; % Margine statico iniziale
55 C_m_alpha_0 = Cond(10); % [1/rad]
56 SM = 0.22; % Margine statico
57 C_m_alpha = C_m_alpha_0*(SM/SM_0); % [1/rad]
58 C_m_alpha_dot = Cond(12); % [1/rad]
59 C_m_q = Cond(14); % [1/rad]
60 C_m_Mach = Cond(17);
61 C_L_de = Cond(18); % [1/rad]
62 C_m_de = Cond(19); % [1/rad]
63
64 % Derivate di stabilità
65 X_u = -(qbar_0*S/(mass*U_0))... % [1/s]
66 *(2*C_D + Mach_0*C_D_Mach); % constant thrust
67 X_w = (qbar_0*S/(mass*U_0))... % [1/s]
68 *(C_L - C_D_alpha);
69 X_wdot = 0;
70 X_q = 0;
71 X_de = 0;
72 X_dT = 0;
73
74 Z_u = -(qbar_0*S/(mass*U_0))*( ... % [1/s]
75 2*C_L+(Mach_0^2/(1-Mach_0^2))...
76 *C_L_Mach); % constant thrust
77 Z_w = -(qbar_0*S/(mass*U_0))... % [1/s]
78 *(C_D + C_L_alpha);
79 Z_wdot = -(1/(2*mu_0))*C_L_alpha_dot;
80 Z_q = -(U_0/(2*mu_0))*C_L_q; % [m/s]
81 Z_dT = 0;
82 M_u = (qbar_0*S*cbar/(Iyy*U_0))... % [1/(ms)]
83 *Mach_0*C_m_Mach;
84 M_w = (qbar_0*S*cbar/(Iyy*U_0))... % [1/(ms)]
85 *C_m_alpha;
86 M_wdot = (rho_0*S*(cbar^2)/(4*Iyy))...% [1/m]
87 *C_m_alpha_dot;
88 M_q = (rho_0*U_0*S...
89 *(cbar^2)/(4*Iyy))*C_m_q;
90
91 M_dT = 0;
92
93 M_a = M_w*U_0;
94 M_adot = M_wdot*U_0;
95
96 k_hat = M_wdot/(1-Z_wdot);
97
98 %%%%%%%%
99 % DEFINIZIONI MATRICI %
100 %%%%%%%%
101
102 % Definizione matrice A_longitudinale. Si veda l'eq. (16.147b)
103 A_lon(1,1) = X_u;
104 A_lon(1,2) = X_w; A_lon(1,3) = 0;
105 A_lon(1,4) = -g_0*cos(Gamma_0); % Prima riga
106
107 A_lon(2,1) = Z_u/(1 - Z_wdot);
108 A_lon(2,2) = Z_w/(1 - Z_wdot);
109 A_lon(2,3) = (Z_q + U_0)/(1 - Z_wdot);
110 A_lon(2,4) = -g_0*sin(Gamma_0)/(1 - Z_wdot); % Seconda riga
111
112 A_lon(3,1) = M_u + k_hat*Z_u;
113 A_lon(3,2) = M_w + k_hat*Z_w;
114 A_lon(3,3) = M_q + k_hat*(Z_q+U_0);
115 A_lon(3,4) = -k_hat*g_0*sin(Gamma_0); % Terza riga
116
117

```

```

118 A_lon(4,1) = 0;
119 A_lon(4,2) = 0;
120 A_lon(4,3) = 1;
121 A_lon(4,4) = 0; % Quarta riga
122
123 % Matrice input. Si veda l'eq. (16.147c)
124 B_lon = [ ...
125 X_de/mass, X_dT/mass; ...
126 Z_de/(mass-Z_wdot), Z_dT/(mass-Z_wdot); ...
127 (M_de + Z_de*M_wdot/(mass-Z_wdot))/Iyy, ...
128 (M_dT + Z_dT*M_wdot/(mass-Z_wdot))/Iyy; ...
129 0, 0];
130
131 % Matrici output
132 C_lon = eye(3,4);
133 D_lon = zeros(3,2);
134
135 %%%%%%%%
136 % SOLUZIONE PROBLEMA AGLI AUTOVALORI E AUTOVETTORI %
137 %%%%%%%%
138
139 % symbolic determinant --> characteristic equation
140 syms lambda
141
142 characteristic_sym_polyomial_lon = det(A_lon-lambda*eye(4));
143 char_poly_lon = sym2poly(characteristic_sym_polyomial_lon);
144
145 a1 = char_poly_lon(1);
146 a2 = char_poly_lon(2); a2 = a2/a1; a3 = char_poly_lon(3);
147 a3 = a3/a1; a4 = char_poly_lon(4);
148 a4 = a4/a1;
149 a5 = char_poly_lon(5);
150 a5 = a5/a1;
151 a1 = 1;
152
153 % Autovettori e autovalori
154 [V,D] = eig(A_lon);% La funzione eig accetta in input una matrice
155 % quadrata e restituisce una matrice di autovettori
156 % V e autovalori D. Quest'ultima ha gli autovalori
157 % sulla diagonale.
158
159 W = inv(V); % L'inversa di V ha gli autovettori sinistri sulle righe
160
161 V_SP = V(:,1); % Autovettori di short period
162 V_SP = V_SP/V_SP(4); % normalizzo rispetto V_SP(4)
163 V_SP(1,1) = V_SP(1,1)/U_0;
164 V_SP(2,1) = V_SP(2,1)/U_0;
165 V_SP(3,1) = V_SP(3,1)/(2*U_0/cbar);
166
167 V_Ph = V(:,3); % Autovettori fugoide
168 V_Ph = V_Ph/V_Ph(4); % normalizzo rispetto V_Ph(4)
169 V_Ph(1,1) = V_Ph(1,1)/U_0;
170 V_Ph(2,1) = V_Ph(2,1)/U_0;
171 V_Ph(3,1) = V_Ph(3,1)/(2*U_0/cbar);
172
173 % Creo una rappresentazione dello spazio degli stati tramite la struttura
174 % dati sys
175 sys_lon = ss( ... %caso non normalizzato rispetto theta
176 A_lon , ... % A
177 B_lon , ... % B
178 eye(4,4) , ... % C
179 zeros(4,2) ... % D
180 );
181
182 % Break Points della legge di comando 'dellequilibratore'
183 t_BP = [0, 7.99, 8, 9.0, 9.01, 19.99, 20, 21.0, 21.01, t_fin];
184 de_deg_BP = [0, 0, -delta_IN, -delta_IN, 0, 0, delta_IN, delta_IN, 0, 0];
185 % Time vector
186 t_Elevator_Doublet = [0:0.025:t_fin]';
187
188 % Deflessione comandata - vettore colonna
189 de_rad_Elevator_Doublet = ...
190 interp1(t_BP,convang(de_deg_BP,'deg','rad'),t_Elevator_Doublet);
191 de_deg_Elevator_Doublet = convang(de_rad_Elevator_Doublet,'rad','deg');
192 % Input u, matrice nx2
193 u_Elevator_Doublet = [ ...
194 de_rad_Elevator_Doublet, zeros(length(de_rad_Elevator_Doublet),1)];
195 % Condizioni iniziali
196 x0_Elevator_Doublet = [0;0;0;0]; % vettore colonna
197
198 % Simulation
199 [y_Elevator_Doublet, t_Elevator_Doublet, x_Elevator_Doublet] = ...
200 lsim(sys_lon, u_Elevator_Doublet, t_Elevator_Doublet, x0_Elevator_Doublet);
201 % La funzione lsim permette di risolvere il sistema LTI definendo anche
202 % un vettore di input
203
204
205
206

```

```

207 %% plot
208 % complex axis
210 fig1=figure("NumberTitle","off","Name","Short Period")
211
212 compass(real(V_SP(2)),imag(V_SP(2)),'-g'); hold on;
213 compass(real(V_SP(1)),imag(V_SP(1)),'-r'); hold on;
214 compass(real(V_SP(3)),imag(V_SP(3)),'-k'); hold on;
215 compass(real(V_SP(4)),imag(V_SP(4)),'-b'); hold on;
216 legend('w/U_0','u/U_0','qc/(2U_0)','\theta','Location','southeast')
217
218
219 fig2=figure("NumberTitle","off","Name","Phugoid")
220
221 compass(real(V_Ph(1)),imag(V_Ph(1)),'-r'); hold on;
222 compass(real(V_Ph(2)),imag(V_Ph(2)),'-g'); hold on;
223 compass(real(V_Ph(3)),imag(V_Ph(3)),'-k'); hold on;
224 compass(real(V_Ph(4)),imag(V_Ph(4)),'-b'); hold on;
225 legend('w/U_0','u/U_0','qc/(2U_0)','\theta','Location','southeast')
226
227 % plot time
228 fig3=figure("NumberTitle","off","Name","Risposta Libera - Fugoide")
229 tiledlayout(2,2)
230
231 nexttile
232 plot(t_Elevator_Doublet,convang(de_rad_Elevator_Doublet,'rad','deg'),'-k',LineWidth=1.3);
233 ylabel('Elevator deflection(deg)')
234 xlabel('time(s)');
235 grid on;
236
237 nexttile
238 plot(t_Elevator_Doublet,y_Elevator_Doublet(:,1),'-k',LineWidth=1.3); hold on;
239 ylabel('Velocity(ft/s)')
240 xlabel('time(s)');
241 grid on;
242
243 nexttile
244 plot(t_Elevator_Doublet,y_Elevator_Doublet(:,2),'-k',LineWidth=1.3); hold on;
245 ylabel('\alpha(deg)')
246 xlabel('time(s)');
247 grid on;
248
249 nexttile
250 plot(t_Elevator_Doublet,y_Elevator_Doublet(:,4),'-k',LineWidth=1.3); hold on;
251 ylabel('\theta(deg)')
252 xlabel('time(s)');
253 grid on;
254 diary off
255
256 saveas(fig1,"Figura_16_3_1.eps",'epsc');
257 saveas(fig2,"Figura_16_3_2.eps",'epsc');
258 saveas(fig3,"Figura_16_3_3.eps",'epsc');
259
260 end

```

5.3.1 Risultati

I risultati mostrati sono relativi ai dati iniziali in figura 5.7. Si ottengono i grafici in figura 5.8.

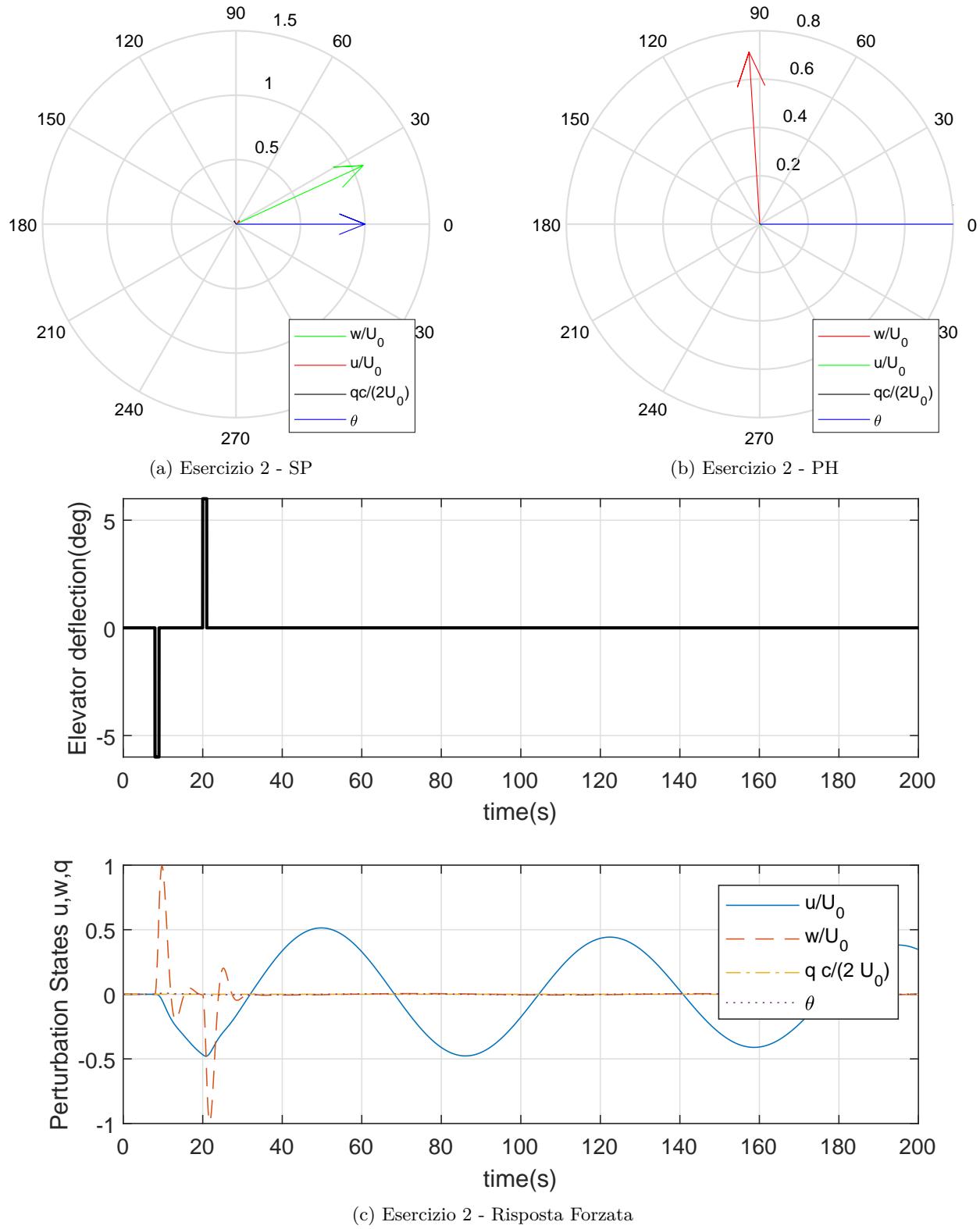


Figura 5.8: Esercizio 2 - Configurazione 2

5.4 Esercizio 3

In questo esercizio si analizza il modo di corto periodo e fugoide di un velivolo Tecnam P2010, i cui dati sono presenti in Tabella 2.2. Nella prima parte del codice c'è l'import dei dati del velivolo con la funzione `datcomimport`. Si calcolano poi le condizioni di trim, per poi procedere alla definizione della matrice A_{Lon} , alla risoluzione del problema e al calcolo della risposta libera. I dati in input sono i seguenti:

- numero di Mach iniziale (0.2);
- altitudine iniziale (1000m);
- angolo di salita iniziale (0 deg)
- velocità angolare di pitch (0 deg/s)

5.4.1 Risultati

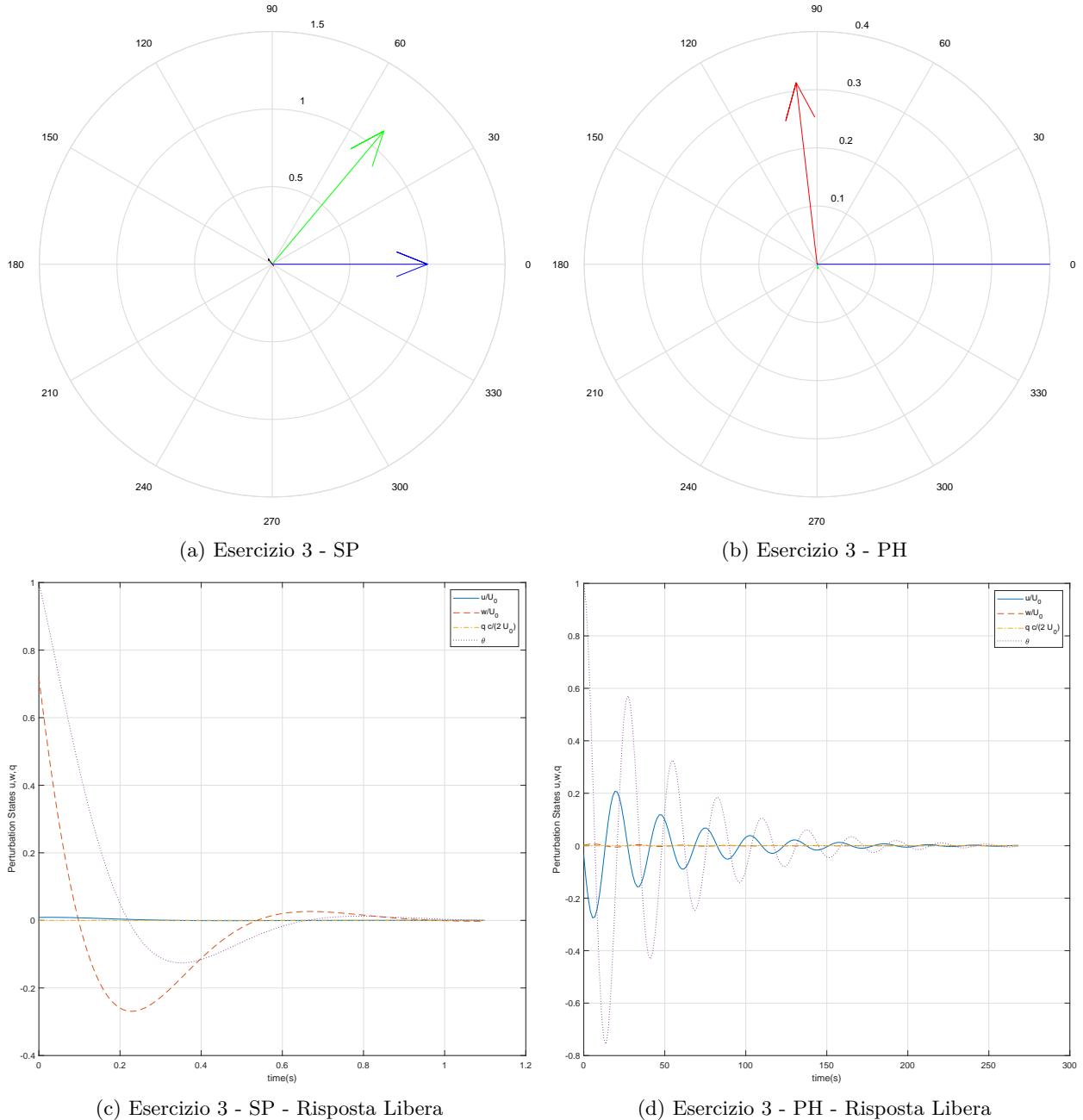


Figura 5.9: Esercizio 3

Condizioni di Trim - P2010

Le condizioni di trim iniziali ottenute sono:

alpha_0 = 3.7787°
delta_{e,0} = 2.865°
delta_{f,0} = -4.9605°
delta_{T,0} = 0.1
