Project of an UAV of infinite autonomy

APPENDIX

AUTHOR
Sánchez Bravo, Àlex

DIRECTOR
Pérez Llera, Luis Manuel

ESEIAAT
Polytechnic University of Catalonia

Delivery date
10/06/2019

Contents

A Conceptual Design Code

This section of the Appendix presents the code employed to choose an optimum configuration for the UAV in the Conceptual Design stage.

A.1 Input Data

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%THIS IS THE INITIAL DATA SCRIPT%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%% AERODYNAMICS

%RG15
%Cl=0.638;%Re=200k, alpha=5,73
%Cd=0.0115;%Re=200k, alpha=5,73

%S7012
%Cl = 0.857;
%Cd = 0.0131;
%SD7003
Cl = 0.645;
Cd = 0.013;

v=13;%m/s
rho=0.1815;%kg/m^3
mi=12.83e-6;%Pa*s
e = 0.89;%Span planform efficiency factor (not oswald's)
%% EFFICIENCY
eta_p=0.7;%Propulsive efficiency
eta_sc=0.3;%Solar Cell efficiency: it theory, it considers the loss of efficiency ...
    due to packaging
eta_bat_in = 0.837; %loss of energy entering the batteries
eta_bat_out = 0.837; %loss of energy going out of the batteries
eta_motor = 0.97; %Efficiency of the electric motor
%% POWER CONSUMPTION
P_avionics = 2.5 + 6.5 + 2.3;%W ->Controller, Datalink, 4Servos
%% WEIGHT -> All the weights should be in Newtons
W_avionics= (0.18 + 0.18 +0.14)*9.81;%N ->Controller, Datalink, 4 Servos
EdW_batteries = 1.8e6/9.81;%J/N (Sion Batteries)
%WdP_propulsion = 0.032;%N/W ->taken from the "HELIPLAT" document.
WdP_propulsion = 0.08; %(Noth, 2008)
%Power control of the batteries??
WdS_sc = 0.35*9.81;%N/m^2   ->  weight of the solar cells (MicroLink devices)
%% DESIGN FACTORS
n = 2; %limit load factor ->Altman, p.38 states that n=2 is a reasonable number.
```

A.2 Main

```
%P : "Power"
%W : "Weight"
%E : "Energy"
%EdP: Energy/Power ->Energy divided Power
clear;close all;clc
run('input_data');
%% First Iteration
AR_vec = linspace(7,11,5);
Sw_vec = linspace(5,25,30);
%AR_vec = linspace(9.5/1.05,9.5/1.05,1);
%Sw_vec = linspace(9.975,9.975,1);
L_vec = zeros(length(AR_vec),length(Sw_vec)); %AR rows, Sw columns
W_vec = zeros(length(AR_vec),length(Sw_vec)); %AR rows, Sw columns
for i=1:length(AR_vec)
for j=1:length(Sw_vec)
    AR = AR_vec(i);
    Sw = Sw_vec(j);
%% Initial Calculations
CL = Cl; L = 1/2*rho*v^2*Sw*CL;
CDi = CL^2/(pi*AR*e);CD = Cd + CDi;
D = 1/2*rho*v^2*Sw*CD;


W_airframe = 0.022*Sw^1.55*AR^1.3;

S_sc = Sw*1;
W_sc = WdS_sc*S_sc; % "1": Sw = S_sc
%% Power consumption
P_propulsion = D*v/(eta_p*eta_motor);
%% Estimate the Weight (not iterative)

W_propulsion = WdP_propulsion*P_propulsion;%N
W_constant = W_sc + W_airframe + W_avionics + W_propulsion; %f(AR,Sw,v)
[E_rad,ts] = IntegrateRadiation();
[W_batteries, SparePower] = WeightBatteries(E_rad, P_propulsion, S_sc, eta_sc,...
    eta_bat_out, EdW_batteries, P_avionics);

W = W_batteries + W_constant;

L_vec(i,j) = L;
W_vec(i,j) = W;
W_airframe_vec(i,j) = W_airframe;
W_sc_vec(i,j) = W_sc;
W_avionics_vec(i,j) = W_avionics;
W_propulsion_vec(i,j) = W_propulsion;
W_batteries_vec(i,j) = W_batteries;
end
end
c_max = sqrt(max(Sw_vec)/min(AR_vec));
c_min = sqrt(min(Sw_vec)/max(AR_vec));
Re_min = rho*v*c_min/mi;
Re_max = rho*v*c_max/mi;

PostProcess(L_vec,W_vec,AR_vec,Sw_vec, W_airframe_vec, W_sc_vec, ...
   W_avionics_vec, W_propulsion_vec, W_batteries_vec);%AR files, Sw columnes
save('data');
```

A.3 Incident Energy from the Sun

```
function [E_rad,ts] = IntegrateRadiation()
%E_rad is the energy per square meter from the radiation
div = 5;%divisions each hour
P_rad = readtable('RadiationJune.xlsx');
    hours=size(P_rad,1);%amount of hours (24 hopefully)
    pos = div*hours;%number of positions of E_rad
E_rad = zeros(pos,2);% time ; radiation

cont = 0;
for i=1:hours
    if i≠hours
    fst = P_rad.radiation(i);%radiation this hour
    scd = P_rad.radiation(i+1);%radiation next hour
    in = (scd - fst)/div;%radiation increase each step
    else
       fst = P_rad.radiation(i);
       scd = P_rad.radiation(1);
       in = (scd - fst)/div;

    end

    for j=1:div
        cont = cont +1;
       E_rad(cont,2) = 1800/div*(2*fst + in*(2*j -1));%J/m^2
       E_rad(cont,1) = P_rad.hour(i) + (j-1)/div; %hours
       E_rad(cont,1) = E_rad(cont,1)*3600; %seconds
    end
end
ts = 3600/div; %timestep (seconds)
plot(E_rad(:,1),E_rad(:,2),'color',[0 0 0])
end
```

A.4 Weight of the Batteries

```
function [W_batteries, SparePower] = WeightBatteries(E_rad, P_propulsion, S_sc, ...
    eta_sc,...
    eta_bat_out, EdW_batteries, P_avionics)


%CALCULATE W_batteries
[t1,t2,E_night,E_day] = EnergySwitch(P_avionics, P_propulsion, S_sc, eta_sc, E_rad);
    tn = 24*3600 - (t2 - t1);%Night Duration
    E_batteries_night = ((P_avionics + P_propulsion)*tn ...
        -E_night*S_sc*eta_sc)*1/eta_bat_out;
    W_batteries = E_batteries_night/EdW_batteries;
    if E_day - (P_avionics + P_propulsion)*(t2 - t1) < E_batteries_night
        warning('The batteries have not filled up!!')
    end
    SparePower = E_day-E_batteries_night-(P_avionics + P_propulsion)*(t2 - t1);

end
```

A.5 Calculate start and end of the day

```
function [t1,t2,E_night,E_day] = EnergySwitch(P_avionics, P_propulsion, S_sc, ...
    eta_sc, E_rad)
close all;
%P_avionics = 11;P_propulsion = 69; Sw = 10; eta_sc = 0.3;
%[E_rad,ts] = IntegrateRadiation();
ts = E_rad(2,1)-E_rad(1,1);%timestep
b1=false;%we start the loop when there isn't enough energy from solar cells -> ...
    false==night
E_night = 0; E_day = 0;
t1=0;t2=0;
for i=1:length(E_rad)

b2 = S_sc*eta_sc*E_rad(i,2) - ts*(P_avionics + P_propulsion)>0; %b2=true if energy ...
    balance >0

%Obtaining t1 and t2;
%From t1 -> t2 the batteries are filling up.
    if b2≠b1 && b1==0
    t1 = E_rad(i,1);
    pos1 = i;
    elseif b2≠b1 && b1==1
    t2 = E_rad(i,1);
    pos2 = i;
    end
b1 = b2;

%Total energy obtained during night and during day
    if b2 == 0
        E_night = E_night + E_rad(i,2);
    elseif b2 ==1
        E_day = E_day + E_rad(i,2);
    end


end

if t1==0 ||t2==0
    warning('no hi ha prou potencia propulsiva! t1 o t2 no shan assignat');

end
%postprocess
%{
figure(1)
plot(E_rad(pos1:pos2,1),E_rad(pos1:pos2,2),'color',[0.5 0.5 ...
    0.5],'LineWidth',1,'DisplayName','Batteries filling up')
hold on
plot(E_rad(1:pos1,1),E_rad(1:pos1,2),E_rad(pos2:24*5,1),E_rad(pos2:24*5,2),'color',[0 ...
    0 0],'LineWidth',2,'DisplayName','Batteries providing energy')
hold on
%plot(E_rad([1:pos1,pos2:120],1),E_rad([1:pos1,pos2:120],2),'color',[0 0 ...
    0],'LineWidth',2,'DisplayName','Batteries providing energy')
legend
%}


end
```

A.6 Postprocess

```
function PostProcess(L_vec,W_vec,AR_vec,Sw_vec, W_airframe_vec, W_sc_vec, ...
    W_avionics_vec, W_propulsion_vec, W_batteries_vec)
%load('data')->If you want to use the function as an independent code
%% Figure 1: W/L vs AR and Sw
figure(1)

for i=1:length(AR_vec)
    lgd = ['AR = ', num2str(AR_vec(i))];

    LdW = L_vec(i,:)./W_vec(i,:);
    clr = [0.1 0.1 0.1]*(i-1)*9.5/(length(AR_vec)-1);
    wdth = 2-1*(i-1)/(length(AR_vec)-1);
    plot(Sw_vec,LdW, 'DisplayName',lgd,'color',clr,'linewidth',wdth);
    hold on;
end
xlabel('Sw (m^2)')
ylabel('L/W')
title('Spare Weight of the UAV vs AR and Sw')
legend
    hor = linspace(min(Sw_vec),max(Sw_vec),10);
    ver = ones(10,1);
    plot(hor, ver,'--','DisplayName','n=1','color',[0 0 0])
    hold on;

%% Figure 2: W_airframe vs AR and Sw
figure(2)

for i=1:length(AR_vec)
      lgd = ['AR = ', num2str(AR_vec(i))];

    clr = [0.1 0.1 0.1]*(i-1)*9.5/(length(AR_vec)-1);
    wdth = 2-1*(i-1)/(length(AR_vec)-1);
    plot(Sw_vec,W_airframe_vec(i,:), 'DisplayName',lgd,'color',clr,'linewidth',wdth);
    hold on;
end
xlabel('Sw (m^2)')
ylabel('W_{airframe} (N)')
title('Weight of the Airframe vs AR and Sw')
legend

%% Figure 3: W_batteries vs AR and Sw
figure(3)

for i=1:length(AR_vec)
      lgd = ['AR = ', num2str(AR_vec(i))];

    clr = [0.1 0.1 0.1]*(i-1)*9.5/(length(AR_vec)-1);
    wdth = 2-1*(i-1)/(length(AR_vec)-1);
    plot(Sw_vec,W_batteries_vec(i,:), 'DisplayName',lgd,'color',clr,'linewidth',wdth);
    hold on;
end
xlabel('Sw (m^2)')
ylabel('W_{batteries} (N)')
title('Weight of the Batteries vs AR and Sw')
legend

%% Figure 4: W_propulsion vs AR and Sw
figure(4)
```

```
59
60 for i=1:length(AR_vec)
61       lgd = ['AR = ', num2str(AR_vec(i))];
62
63     clr = [0.1 0.1 0.1]*(i-1)*9.5/(length(AR_vec)-1);
64     wdth = 2-1*(i-1)/(length(AR_vec)-1);
65     plot(Sw_vec,W_propulsion_vec(i,:), 'DisplayName',lgd,'color',clr,'linewidth',wdth);
66     hold on;
67 end
68 xlabel('Sw (m^2)')
69 ylabel('W_{propulsion} (N)')
70 title('Weight of the Propulsion vs AR and Sw')
71 legend
72
73 %% Figure 5: W_sc vs AR and Sw
74 figure(5)
75
76 for i=1:length(AR_vec)
77       lgd = ['AR = ', num2str(AR_vec(i))];
78
79     clr = [0.1 0.1 0.1]*(i-1)*9.5/(length(AR_vec)-1);
80     wdth = 2-1*(i-1)/(length(AR_vec)-1);
81     plot(Sw_vec,W_sc_vec(i,:), 'DisplayName',lgd,'color',clr,'linewidth',wdth);
82     hold on;
83 end
84 xlabel('Sw (m^2)')
85 ylabel('W_{sc} (N)')
86 title('Weight of the Solar Cells vs AR and Sw')
87 legend
88
89 %% Figure 6: W_avionics vs AR and Sw
90 figure(6)
91
92 for i=1:length(AR_vec)
93     lgd = ['AR = ', num2str(AR_vec(i))];
94     clr = [0.1 0.1 0.1]*(i-1)*9.5/(length(AR_vec)-1);
95     wdth = 2-1*(i-1)/(length(AR_vec)-1);
96     plot(Sw_vec,W_avionics_vec(i,:), 'DisplayName',lgd,'color',clr,'linewidth',wdth);
97     hold on;
98 end
99 xlabel('Sw (m^2)')
100 ylabel('W_avionics (N)')
101 title('Weight of the avionics vs AR and Sw')
102 legend
103
104 %% Figure 7: W vs AR and Sw
105 figure(7)
106
107 for i=1:length(AR_vec)
108     lgd = ['AR = ', num2str(AR_vec(i))];
109     clr = [0.1 0.1 0.1]*(i-1)*9.5/(length(AR_vec)-1);
110     wdth = 2-1*(i-1)/(length(AR_vec)-1);
111     plot(Sw_vec,W_vec(i,:), 'DisplayName',lgd,'color',clr,'linewidth',wdth);
112     hold on;
113 end
114 xlabel('Sw (m^2)')
115 ylabel('W (N)')
116 title('Total Weight as a function of AR and Sw')
117 legend
118 end
```

B Weight Refinement Code

This section of the Appendix presents the code employed to refine the weight of the UAV in the second stage of the design — the Preliminary Design. This code is also a tool to calculate the center of gravity of the aircraft, indispensable for the aerodynamic study of stability.

B.1 Input Data

```
bw = 9.5; %wing span (m)
cw = 1.05; %wing chord (m)

bht = 2.57; %horizontal tail span (m)
cht = 0.43; %horizontal tail chord (m)

bvt = 1.15; %vertical tail span (m)
cvt = 0.38; %vertical tail chord (m)

lwt = bw/2; %Distance between aerodynamic centers of the tail and the chord
dwt = cw/4 + lwt - cht/4;%Distance between the LE of the wing and the LE of the ...
    horizontal tail

M_battery = 0.154; %154g each battery
n_batteries = 20;M_batteries = n_batteries*M_battery;

M_avionics = 4.9/9.81;%(kg)

M_sc_w = bw*cw*0.7*0.35;
M_sc_t = bht*cht*0.7*0.35*0;

M_propulsion = 6.9/9.81;%kg
%% Design parameters

xcg_batteries = 6/100;
xcg_avionics = 6/100;
xcg_propulsion = -5/100;% a negative xcg for the propulsion is sensible
xcg_sc_w = 0.5*cw;
%this also accounts for the weight of the control surfaces
xcg_sc_t = 0.55*cht + dwt;
```

B.2 Main

```
clear;clc;
%This function calculates the center of gravity of the plane
[M_airframe,xcg_skin_w,M_skin_w,xcg_skin_ht,M_skin_ht,xcg_skin_vt,M_skin_vt,...
xcg_profile_w,M_profile_w,xcg_profile_ht,M_profile_ht, xcg_profile_vt, M_profile_vt,...
M_mspar,M_tespar, M_htspar, M_vtspar, M_tehtspar, M_tevtspar, M_boom] = ...
    WeightsPositions();

%% Input Data
%% Data
run('InputData_CG');

%CG referenced from the LE of the wing
% NOTE -> The xcg's of the wing and the tail are referred to their respective LE
%COMPUTE THE AIRFRAME MASS:
num_xcg_airframe = xcg_skin_w*M_skin_w + (xcg_skin_ht + dwt)*M_skin_ht + ...
    (xcg_skin_vt + dwt)*M_skin_vt...
    +xcg_profile_w*M_profile_w + (xcg_profile_ht + dwt)*M_profile_ht + ...
        (xcg_profile_vt + dwt)*M_profile_vt...
    +cw/4*M_mspar + (cw - 2.5/1000)*M_tespar + (dwt + cht/4)*M_htspar + (dwt + ...
        cvt/4)*M_vtspar...
    + (dwt + cht - 1/1000)*M_tehtspar + (dwt + cvt/4 - 1/1000)*M_tevtspar + ((dwt - ...
        cw)/2 + cw)*M_boom;
xcg_airframe = num_xcg_airframe/M_airframe;


num_xCG = xcg_airframe*M_airframe + xcg_batteries*M_batteries + ...
    xcg_avionics*M_avionics...
    + xcg_sc_w*M_sc_w + xcg_sc_t*M_sc_t + xcg_propulsion*M_propulsion + 1*6/100;
M = M_airframe + M_batteries + M_avionics + M_sc_w + M_sc_t + M_propulsion + 1;
xCG = num_xCG/M

M_vt = M_skin_vt + M_vtspar + M_tevtspar + M_profile_vt;
xcg_vt = ((xcg_skin_vt+dwt)*M_skin_vt + (dwt + cvt/4)*M_vtspar + (dwt + cvt/4 - ...
    1/1000)*M_tevtspar + (xcg_profile_vt + dwt)*M_profile_vt)/M_vt

M_ht = M_skin_ht + M_htspar + M_tehtspar + M_profile_ht;
xcg_ht = ((xcg_skin_ht + dwt)*M_skin_ht + (dwt + cht/4)*M_htspar + (dwt + cht - ...
    1/1000)*M_tehtspar + (xcg_profile_ht + dwt)*M_profile_ht)/M_ht

M_w = M_skin_w + M_mspar + M_tespar + M_profile_w;
xcg_w = (xcg_skin_w*M_skin_w + cw/4*M_mspar + (cw - 2.5/1000)*M_tespar + ...
    xcg_profile_w*M_profile_w)/M_w

xcg_boom = ((dwt - cw)/2 + cw)
```

B.3 Calculate Weight and Position of the elements of the Airframe

```
function ...
    [M_airframe,xcg_skin_w,M_skin_w,xcg_skin_ht,M_skin_ht,xcg_skin_vt,M_skin_vt,...
    xcg_profile_w,M_profile_w,xcg_profile_ht,M_profile_ht, xcg_profile_vt, ...
        M_profile_vt,...
    M_mspar,M_tespar, M_htspar, M_vtspar, M_tehtspar, M_tevtspar, M_boom] = ...
        WeightsPositions()
run('InputData');
%Read Profile Coordinates (wing)
profilew = 'sd7003.dat';
[xw,yw] = ReadProfile(profilew);

%Read Profile Coordinates (tail)
profilet = 'sd8020.dat';
[xt,yt] = ReadProfile(profilet);

%% Xcg, Ycg, Mass of the skin
[xcg_skin_w, ycg_skin_w, perimeter_w] = PerimeterProfile(xw*cw, yw*cw);
M_skin_w = perimeter_w*bw*rho_skin*t_skin;
[xcg_skin_ht, ycg_skin_ht, perimeter_ht] = PerimeterProfile(xt*cht, yt*cht);
M_skin_ht = perimeter_ht*bht*rho_skin*t_skin;
[xcg_skin_vt, ycg_skin_vt, perimeter_vt] = PerimeterProfile(xt*cvt, yt*cvt);
M_skin_vt = perimeter_vt*bvt*rho_skin*t_skin;

M_skin = M_skin_w + M_skin_ht + M_skin_vt;
%% Xcg, Ycg, Mass of the profiles
[xcg_profile_w, ycg_profile_w, area_w] = AreaProfile(xw*cw, yw*cw);
[xcg_profile_vt, ycg_profile_vt, area_vt] = AreaProfile(xt*cvt, yt*cvt);
[xcg_profile_ht, ycg_profile_ht, area_ht] = AreaProfile(xt*cht, yt*cht);

M_profile_w = rho_profile*area_w*t_profile*n_profiles_w;
M_profile_vt = rho_profile*area_vt*t_profile*n_profiles_vt;
M_profile_ht = rho_profile*area_ht*t_profile*n_profiles_ht;

M_profiles = M_profile_w + M_profile_vt + M_profile_ht;
%% Xcg, Ycg, Mass of the spars
area_mspar = pi/4*diameter_mspar^2 - pi/4*(diameter_mspar - t_spar)^2;
M_mspar = area_mspar*bw*rho_spar;
area_tespar = t_spar*5/1000;%the te spar will be 5x0.9 mm
M_tespar = area_tespar*bw*rho_spar;
area_tspar = pi/4*diameter_tspar^2 - pi/4*(diameter_tspar - t_spar)^2;%the two ...
    tails will have the same spar
M_htspar = area_tspar*bht*rho_spar;
M_vtspar = area_tspar*bvt*rho_spar;
area_tetspar = t_spar*2/1000; %the tail te spar will be 2x0.9mm
M_tehtspar = area_tetspar*bht*rho_spar;
M_tevtspar = area_tetspar*bvt*rho_spar;

M_spars = M_mspar + M_tespar + M_htspar + M_vtspar + M_tehtspar + M_tevtspar;

area_boom = pi/4*diameter_boom^2 - pi/4*(diameter_boom - t_spar)^2;
M_boom = area_boom*(bw/2-3/4*cw-3/4*cht)*rho_spar;

M_airframe = M_skin + M_profiles + M_spars + M_boom;

end %function
```

B.4 Read Profile Points

```
function [x,y] = ReadProfile(namefile)
profile = fopen(namefile);
k= textscan(profile, '%f');
a=k{1};
x=zeros(length(a)/2,1);
y=zeros(length(a)/2,1);
for i=1:length(a)
    par =¬ mod(i,2);
    if par == 1
        y(i/2) = a(i);
    else
    x((i+1)/2) = a(i);
    end
end
end
```

B.5 Calculate the perimeter of the airfoil

```
function [xcg, ycg, perimeter] = PerimeterProfile(x, y)
%xcg = Int_num_x/Perimeter; yxg = Int_num_y/Perimeter
perimeter = 0;
Int_num_x = 0;
Int_num_y = 0;
for i=1:length(x)
    if i==length(x)
        p_i = sqrt((x(1) - x(i))^2+(y(1) - y(i))^2);
        num_x_i = 1/2*(x(i) + x(1))*p_i;
        num_y_i = 1/2*(y(i) + y(1))*p_i;
    else
        p_i = sqrt((x(i+1) - x(i))^2 + (y(i+1) - y(i))^2);
        num_x_i = 1/2*(x(i) + x(i+1))*p_i;
        num_y_i = 1/2*(y(i) + y(i+1))*p_i;
    end
    perimeter = perimeter + p_i;
    Int_num_x = Int_num_x + num_x_i;
    Int_num_y = Int_num_y + num_y_i;
end
xcg = Int_num_x/perimeter;
ycg = Int_num_y/perimeter;
end
```

B.6 Calculate the area of the airfoil

```
%Input: (B, rho, x, y,
%Output: [M, xcg, ycg,
function [xcg, ycg, area] = AreaProfile(x, y)

%xcg = Int_num_x/Int_den;   ycg = Int_num_y/Int/den;
Int_den = 0; %Area of the profile // Denominator in xcg and ycg calculus
Int_num_x = 0; %Numerator in xcg calculus
Int_num_y = 0; %Numerator in ycg calculus
for i=1:length(x)
   if i==length(x)
      A_i = 1/2*(x(i) - x(1))*(y(i) + y(1)); %Area of the portion i: y(x)*dx
      num_x_i = 1/2*(x(i) + x(1))*A_i; %x*A_i = x*y(x)*dx
      num_y_i = 1/2*(y(i) + y(1))*A_i; %y(x)*A_i = y(x)*y(x)*dx
   else
       A_i = 1/2*(x(i) - x(i+1))*(y(i) + y(i+1));
       num_x_i = 1/2*(x(i) + x(i+1))*A_i;
       num_y_i = 1/2*(y(i) + y(i+1))*A_i;
   end
   Int_den = Int_den + A_i;
   Int_num_x = Int_num_x + num_x_i;
   Int_num_y = Int_num_y + num_y_i;
end

xcg = Int_num_x/Int_den;
ycg = Int_num_y/Int_den;
area = Int_den;

end
```