Supplementary Simulation Files

Hawkmoth: Extremum-Seeking vs. Optimal PID

Ahmed A. Elgohary and Sameh A. Eisa

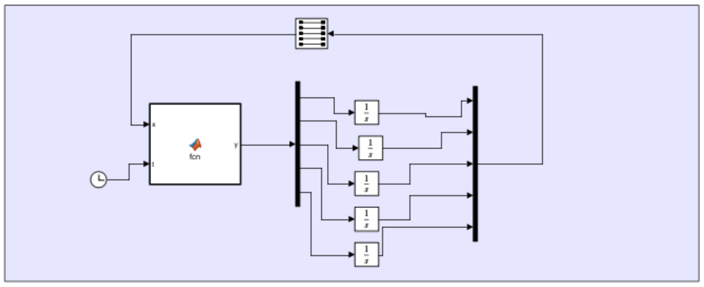
7/19/2025

This PDF embeds both MATLAB/Simulink function blocks used to generate the hawkmoth results. The same template—changing only the numerical parameters—was used for every other insect.

1. Simulation Models ESC and PID

Let's start with the simulation file, so MATLAB and Simulink was used to generate the simulation results for ES, PID, and optimal PID controllers. For ES, we used Simulink model as shown in the following figure. There are 4 states for the model and one additional state represent the ES $\hat{u}$ control update.

Hawkmoth Flapping ESC

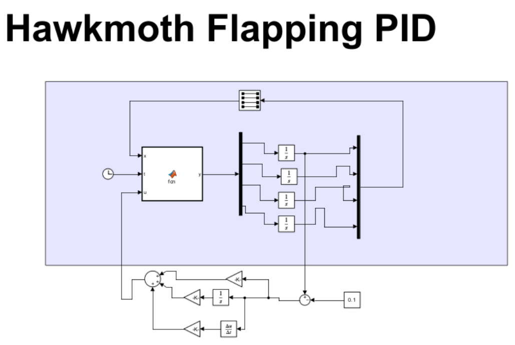


The following code was implemented Inside the Simulink MATLAB function,

```
function y= fcn(x,t)
kd1 =0.0353739; %system parameter kd1
g =9.81; %system parameter g
kL =0.000621676; %system parameter kL
%a4 =1.37801*10^( -7); %system parameter I
f = 26.3; %system frequency in Hz
```

```
%T = 1/f; %system period
%a5 =2*pi*f; %system parameter omega
kd2 =0.33915; %system parameter kd2
kd3 =16.5766; %system parameter kd3
IF=1.37801*10^( -7);
omega=2*pi*f;
J=(x(1)-0)^2;
c2=1;
a=180;
k=10000;
y= [x(3);
    x(4);
    -kd1*abs(x(4))*x(3)+g-kL*x(4)^2;
    -kd3*x(3)*x(4)-kd2*abs(x(4))*x(4)+ c2*x(5) + a*omega*cos(omega*t);
     k*J*a*omega*cos(omega*t)
    ];
```

We used the same model also for the PID controller but (a) was used as a control input for the PID. The following figure represent the flapping PID model. For Simulink MATLAB function, the same Function mentioned above was used but we set the amplitude a value as the control input for the PID.



2. Flapping PID gains Calculations

For PID gains calculations, we followed the same steps mentioned in section III at the paper. So here we will provide the MATLAB code direct for calculation explaining the general steps that we followed.

```
% PID calculated gains for flapping model
clear all
clc
%z=x1 , phi=x2, z_dot=x3, phi_dot=x4, u_hat=x5
% Define symbolic variables
syms x1 x2 x3 x4 kd1 g kL kd2 kd3 IF U omega real
kL =0.000621676; % system parameter kL
IF =1.37801e-7; % system parameter I
omega =2*pi*26.3; % system parameter omega
kd2 =0.33915; % system parameter kd2
```

```
kd1 =0.0353739; % system parameter kd1
epsilon = 50;
abs_phi_dot=x4 * (2/pi)*atan(epsilon*x4); Here we do smoothing
X = [x1; x2; x3; x4];
Z = [x3;
    x4;
    -kd1*abs_phi_dot*x3+g-kL*x4^2;
    -kd3*x3*x4-kd2*abs_phi_dot*x4;
    ];
Y = [
    0;
    0;
    0;
    1/IF;
   ];
%calculate [Y,Z]
L=generateLieBracket(Y, Z, X);
L1=generateLieBracket(Y, L, X);   %[Y,[Y,Z]]
  % the averaged system x_bar_dot
x_dot=Z+ (U^2/(4*omega^2))*L1;
A = jacobian(x_dot, [x1, x2, x3, x4]); % System matrix
B = jacobian(x_dot, U) ;% Input matrix
X_eq = [0, 0, 0, 0];
U_trim=0.004242;
A_num = (subs(A, [x1, x2, x3, x4, U], [X_eq,U_trim]));
B_num = (subs(B, [x1, x2, x3, x4, U], [0,0,0,0,U_trim]));
%Define System Dynamics Her we get the dynamics of the averaged system in the state space form
%Note that, this A and B matrices will change from insect to another insect based on the insect parameters mentioned
above at the beginning.
A = [0, 0, 1, 0;
     0, 0, 0, 1;
     0, 0, -1.6750e+04, 0;
     0, 0, 0, -4.8178e+05];
B = [0; 0; -4.3602e+03; 0];
eignvalues=eig(A);
% Compute Controllability Matrix
Co = ctrb(A, B);
% Check Rank of Controllability Matrix
rank_Co = rank(Co);
% Create State-Space System
sys = ss(A, B, [1 0 0 0 ; 0 1 0 0], 0); % Output is x1 (altitude)
G = tf(sys) % Convert to Transfer Function
% Get PID gains
% Define Symbolic Variables for PID Gains
syms Kp Ki Kd s real
% Define Open-Loop Transfer Function G(s) (Modify This)
G = -4360 / (s^2 + 16750*s);  % Example Open-Loop System
%% Define PID Controller C(s)
```

```
C = Kp + Ki/s;
%% Compute Closed-Loop Transfer Function H(s)
H = simplify((C * G) / (1 + C * G)) % Simplify expression
% Extract the denominator of the closed-loop system
[num, den] = numden(H); % Get numerator and denominator
den = collect(den, s);  % Collect terms in s
% Define Standard Second-Order Form  where settling time T_s= 4/ (zeta*w_n)
T_s= 5 ; % 5 seconds
zeta = 0.707; % Damping ratio  where settling
wn = 4/(T_s*zeta) ;  % Natural frequency
standard_den = s^2 + 2*wn*zeta*s + wn^2; % Standard denominator
% Solve for Kp and Ki by Matching Coefficients
coeffs_H = coeffs(den, s); % Extract coefficients from H(s) denominator
coeffs_standard = coeffs(standard_den, s); % Extract coefficients from standard form
% Solve equations for Kp and Ki
eq1 = coeffs_H(2) == coeffs_standard(2); % Match s coefficient
eq2 = coeffs_H(1) == coeffs_standard(1); % Match constant coefficient
sol = solve([eq1, eq2], [Kp, Ki]) % Solve for Kp and Ki for unit step input, and Here we get the PID gains
```

```
function lieBracket = generateLieBracket(b1, b2, X)
    % Inputs:
    %   - b1, b2: Vector fields (symbolic expressions) representing the dynamics.
    %   - X: Symbolic vector representing the state variables (e.g., [x1; x2; x3]).
    % Calculate the Lie derivatives of b1 and b2 with respect to each state variable
    diff_b1 = jacobian(b1, X);
    diff_b2 = jacobian(b2, X);
    % Compute the Lie bracket: b1b2 = diff_b2 * b1 - diff_b1 * b2
    lieBracket = simplify(diff_b2 * b1 - diff_b1 * b2);
end
```

3. GA calculations

In this step, we used the implemented MATLAB GA function, where we get the z response from the Simulink file to the workspace of MATLAB then we implemented the following code.

```
clc; clear; tic;
lb = [5e-5 4e-5 1e-5];
ub = [5e-1 4e-1 1e-1];

nvars = 3;
opts  = optimoptions('ga','Display','iter');   % or 'off'

[Gains,fval] = ga(@Airplanesim,nvars,[],[],[],[],lb,ub,[],opts);
Err = Airplanesim(Gains);
fprintf('Best gains      : %.4g  %.4g  %.4g\n',Gains);
fprintf('Objective value : %.4g\n',fval);
fprintf('t = %.2f min\n',toc/60);
disp('done');
```

The following MATLAB function was used to calculate the error for the GA function. Note that we get the z vector from Simulink.

```
function Err = Airplanesim(x)
     % Unpack gains
    k_p  = x(1);
    k_i =  x(2);
    k_d =  x(3);
   try
    options = simset('SrcWorkspace','current');
    sim('PID.slx',10,options)
    Err_z=sum((abs(0.1-ans.z)).^2);
    % Err_phi=sum((abs(ans.phi_dot)).^2);
    Err=sqrt(Err_z^2)

    catch
       Err=inf;
    end
end
```

4. Calculate Overshoot and settling for different w0

We used the MATLAB function `stepinfo` as follow

```
% clear all
clc ;
close all

z_0=0.1;

t=out.tout;
z_PID=out.PID_z;
z_ESC=out.ESC_z;

% Compute step response characteristics using stepinfo
% Set the reference value to 0 (since we shifted the response)
pid_info = stepinfo(z_PID, t, 'SettlingTimeThreshold', 0.02);
esc_info = stepinfo(z_ESC, t, 'SettlingTimeThreshold', 0.02);

% Extract overshoot and settling time
overshoot_PID = pid_info.Overshoot; % Percentage overshoot
settlingTime_PID = pid_info.SettlingTime; % Settling time (2% threshold)

overshoot_ESC = esc_info.Overshoot; % Percentage overshoot
settlingTime_ESC = esc_info.SettlingTime; % Settling time (2% threshold)

% Display results
fprintf('PID Controller:\n');
fprintf('Overshoot: %.2f%%\n', overshoot_PID);
fprintf('Settling Time: %.2f seconds\n\n', settlingTime_PID);

fprintf('ESC Controller:\n');
fprintf('Overshoot: %.2f%%\n', overshoot_ESC);
fprintf('Settling Time: %.2f seconds\n', settlingTime_ESC);
```

5. Open-loop eigenvalues

So in this part, we need to calculate the eigenvalues for the time invariant system (averaged system) for the system states w and phi_dot. So the first part is doing the same steps mentioned in part 2 till we get the averaged system x_dot_averaged then follow the following code.

```
  % the averaged system x_bar_dot
```

```
x_dot=Z+ (1/4)*L1
A = jacobian(x_dot, [x1, x2, x3, x4]) % System matrix
B = jacobian(x_dot, U) ;% Input matrix
X_eq = [0, 0, 0, 0];
g=9.81;
U_trim=(sqrt(2*g*IF^2/ (kL)));
A_num = (subs(A, [x1, x2, x3, x4, a], [X_eq, U_trim]));
B_num = (subs(B, [x1, x2, x3, x4, U], [0,0,0,0,U_trim]));
%Define System Dynamics
eignvalues=eig(A_num);
eignvalues= vpa([eignvalues(1);eignvalues(2)])
```

6. Optimal PID eigenvalues
   Here we get also the averaged system as done in step 2, then we get the A and B matrices that represent the linear time invariant system then we follow the following code

```
clear all
clc
%Hawkmoth PID stability analysis
A = [ 0     0      1        0 ;
      0     0      0        1 ;
      0     0  -1.6750e4    0 ;
      0     0      0   -4.8178e5 ];
B = [0 ; 0 ; -4.3602e3 ; 0];
C = [1 0 0 0] ;          % altitude (x1)
                  % just illustrative – keep both if you need them
D = 0;
Gp  = ss(A,B,C,D);  % 1-input, 1-output plant
% === insert your tuned gains here ===   Here we set the optimal PID gains got from GA
Kp = 0.0891245579741591;
Ki = 0.258456531545103;
Kd = 0.0240507113120525;      % leave at 0 if you designed only a PI controller
Gc  = pid(Kp,Ki,Kd);   % your tuned gains
A_eig  = feedback(Gc*Gp,1);   %
eig(CL.A)                % closed-loop eigenvalues
```

7. ESC Eigenvalues
   For ESC calculations, we follow the same steps mentioned in 2 but we will get the averaged ESC system then calculate the eigenvalues using the following code.

```
% ESC eignvalues using VOC
clear all
clc
%z=x1 , phi=x2, z_dot=x3, phi_dot=x4, u_hat=x5
% Define symbolic variables
syms x1 x2 x3 x4 x5 kd1 g kL kd2 kd3 IF U omega real
kL =0.000621676; % system parameter kL
IF =1.37801e-7; % system parameter I
```

```
omega =2*pi*26.3*1.08; % system parameter omega
kd2 =0.33915; % system parameter kd2
kd1 =0.0353739; % system parameter kd1
J=; % please select your J based on each case as z^2 and w_dot^2
a=180;
k=10000;
epsilon = 50;
abs_phi_dot=x4 * (2/pi)*atan(epsilon*x4);
X = [x1; x2; x3; x4; x5];
Z = [x3;
    x4;
    -kd1*abs_phi_dot*x3+g-kL*x4^2;
    -kd3*x3*x4-kd2*abs_phi_dot*x4+x5;
     0
    ];
Y = [
    0;
    0;
    0;
    a;
    k*J*a
   ];
%calculate [Y,Z]
L=generateLieBracket(Y, Z, X);
L1=generateLieBracket(Y, L, X) ;  %[Y,[Y,Z]]
  % the averaged system x_bar_dot
x_dot=Z+ (1/4)*L1;
A = jacobian(x_dot, [x1, x2, x3, x4, x5]); % System matrix
% B = jacobian(x_dot, U) % Input matrix
X_eq = [0, 0, 0, 0, 0.1];
A_num = (subs(A, [x1, x2, x3, x4, x5], [X_eq]));
eignvalues=(eig(A_num));
eignvalues= vpa([eignvalues(1);eignvalues(2)])
```