طراحی کنترل کننده PID بهینه با استفاده از الگوریتم بهینه سازی انبود ذرات[1]

در این پروژه سعی بر طراحی یک کنترل کننده PID بهینه با استفاده از الگوریتم PSO جهت کنترل سیستم رگولاتور ولتاژ اتوماتیک[2] داریم. لذا در ابتدا به بررسی الگوریتم PSO و سیستم AVR خواهیم پرداخت.

الگوریتم بهینه سازی انبوه ذرات:

الگوریتم بهینه‌سازی انبوه ذرات اولین بار توسط اِبرهارت[3] و کِنِدی[4] ارائه گشت. این الگوریتم شامل انبوهی از ذرات می‌باشد که هر کدام می‌توانند پاسخی مطلوب برای مساله تحت بهینه‌سازی باشند. الگوریتم به طور مکرر با محاسبه سرعت هر ذره، موقعیت ذره را به‌هنگام می‌کند. اگر $x_i(t)$ موقعیت ذره i ام در زمان t باشد، موقعیت ذره در هر زمان برابر خواهد بود با:

$$x_i(t+1) = x_i(t) + v_i(t+1) \quad (1)$$

برای هر ذره i سرعت شامل سه مولفه می‌باشد :

- مولفه شناختی، که از بهترین موقعیتی که ذره تا کنون تجربه کرده است نتیجه می‌شود.
- مولفه اجتماعی، که از موقعیت بهترین ذره‌ای که ذره i از آن آگاه است نتیجه می‌شود.
- مولفه اینرسی، که از سرعت قبلی ذره نتیجه می‌شود.

هر ذره بهترین موقعیت خود را با محاسبه مولفه شناختی حفظ می‌کند. مولفه اجتماعی بر اساس موقعیت بهترین ذره که در همسایگی ذرات وجود دارد محاسبه می‌شود. همسایگی به ساختاری اجتماعی گفته می-شود که در آن ذرات با یکدیگر در ارتباط باشند. ساختار الگوریتم به گونه‌ای است که در آن تمام ذرات با

[1] Particle Swarm Optimization (PSO)
[2] Automatic Voltage Control System (AVR)
[3] Eberhart
[4] Kennedy

یکدیگر کاملا ارتباط دارند. در هر چرخه زمانی تمام ذرات بوسیله مولفه اجتماعی که از موقعیت بهترین ذره در کل ذرات نتیجه شده است، بهنگام می‌شوند. این مولفه را $\hat{y}(t)$ می نامیم. سرعت را به‌صورت زیر محاسبه می‌کنیم :

$$v_{i,j}(t+1) = \omega v_{i,j}(t) + c_1 r_{1,j}\left[y_{i,j}(t) - x_{i,j}(t)\right] + c_2 r_{2,j}\left[\hat{y}_j(t) - x_{i,j}(t)\right] \quad (۲)$$

که در آن $v_{i,j}(t)$ سرعت ذره i در بُعد j در زمان t می‌باشد. ثابت‌های ω، c_1 و c_2 به ترتیب وزن اینرسی، ضریب آموزش شخصی و ضریب آموزش سراسری هستند که جهت تنظیم مولفه‌های شناختی و اجتماعی مورد استفاده قرار می‌گیرند. بردارهای $r_{1,j}(t) \sim U(0,1)$ و $r_{2,j}(t) \sim U(0,1)$ عوامل تصادفی الگوریتم هستند. با فرض می‌نیمم‌سازی، بهترین موقعیت ذره i ام که با $\hat{y}_i$ نشان داده می‌شود، به‌صورت زیر بدست می‌آید:

$$y_i(t+1) = \begin{cases} y_i(t) & if\ f\left(x_i(t+1)\right) \geq f(y_i(t)) \\ x_i(t+1) & if\ f(\left(x_i(t+1)\right) \leq f(y_i(t)) \end{cases} \quad (۳)$$

که در آن f تابع هدف می‌باشد. شبه کد الگوریتم بهینه‌سازی انبوه ذرات به‌صورت زیر می‌باشد.

۱. تعیین و مقداردهی اولیه موقعیت و سرعت ذرات

۲. بدست آوردن تابع هزینه برای هر ذره و تعیین بهترین ذره (با کوچکترین مقدار تابع هزینه)

۳. تا زمانی که شرایط توقف برقرار نشده است:

۴. به روز رسانی سرعت و موقعیت ذرات با توجه به موقعیت بهترین ذره

۵. بدست آوردن تابع هزینه تمام ذرات و تعیین بهترین ذره

۶. بروز رسانی بهینه سراسری

۷. پایان الگوریتم

عملکرد الگوریتم بهینه‌سازی انبوه ذرات به طور کامل به وزن اینرسی، ضریب آموزش شخصی و ضریب آموزش سراسری وابسته است. این الگوریتم از دقت و سرعت بالایی در یافتن بهینه‌ترین پاسخ برخوردار است و لذا برای مسائل بهینه‌سازی در فضای چند بعدی بسیار خوب عمل می‌کند.

رگولاتور ولتاژ اتوماتیک (AVR) :

یک سیستم AVR شامل چهار بخش اساسی می باشد. این بخش ها که عبارتند از تقویت کننده، محرک، ژنراتور و سنسور غیرخطی بوده و برای شبیه سازی نیازمند خطی سازی می باشند. پس از خطی سازی، تابع تبدیل هر بخش شامل یک بهره و یک ثابت زمانی به شرح زیر می باشد.

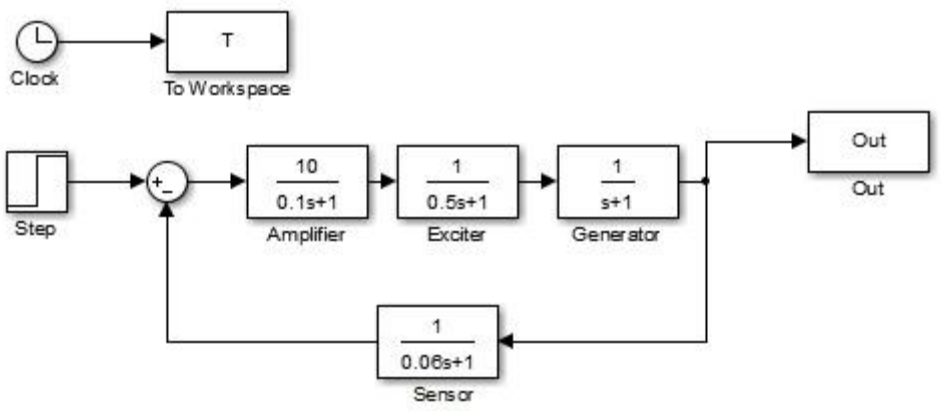
$$TF_{Amplifier} = \frac{K_A}{1+\tau_A s} \ , \ K_A = 10 \ , \qquad \tau_A = 0.1$$

$$TF_{Exciter} = \frac{K_E}{1+\tau_E s} \ , \qquad K_E = 1 \ , \qquad \tau_E = 0.5$$

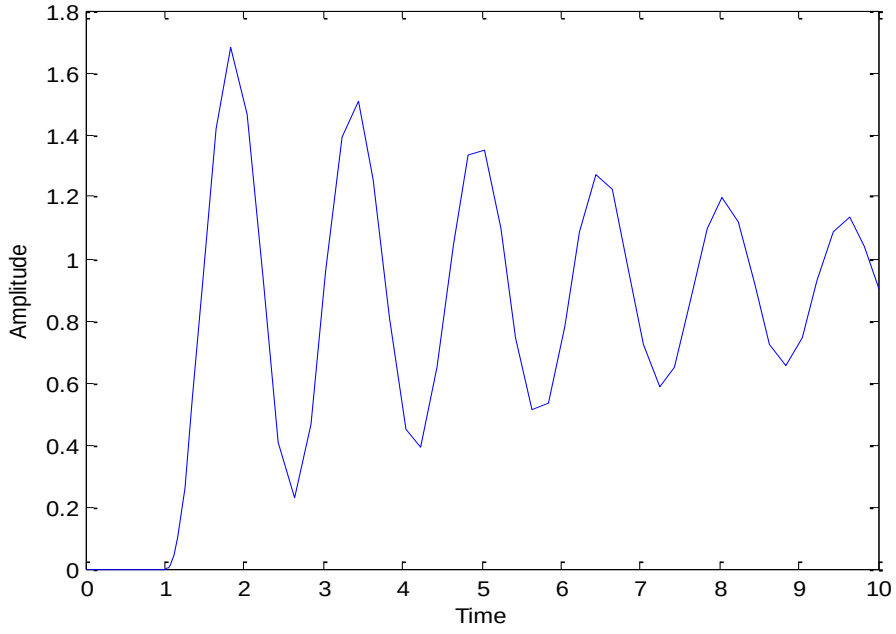
$$TF_{Generator} = \frac{K_G}{1+\tau_G s} \ , \ K_G = 1 \ , \qquad \tau_G = 1$$

$$TF_{Sensor} = \frac{K_S}{1+\tau_S s} \ , \qquad K_S = 1 \ , \qquad \tau_S = 0.06$$

ابتدا سیستم AVR را مطابق شکل زیر شبیه سازی کرده، پاسخ پله آنرا مشاهده می کنیم.



شکل (۱). شبیه سازی سیستم بدون کنترل کننده

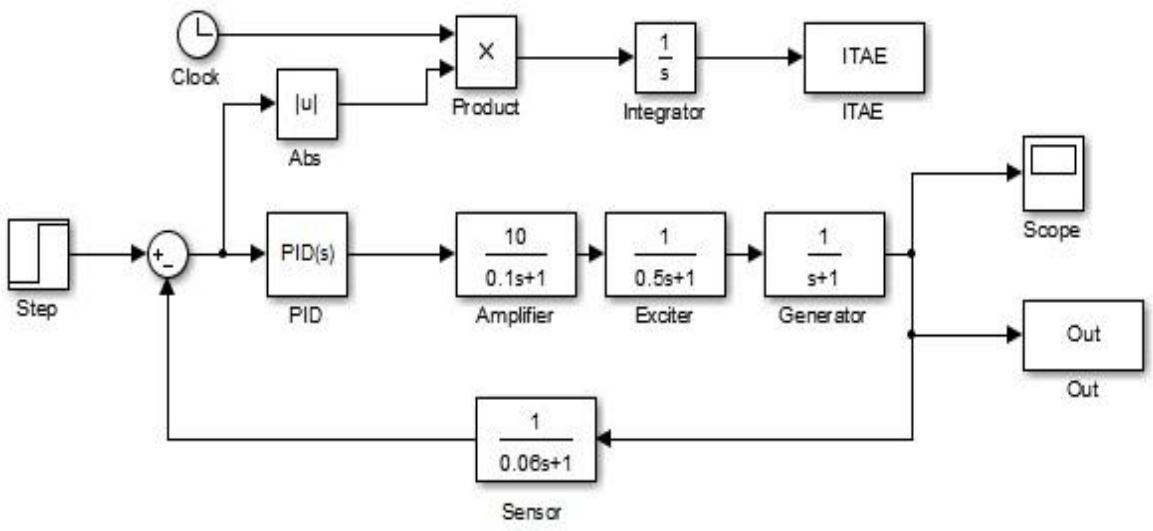


شکل (۲). پاسخ پله سیستم بدون کنترل کننده

سپس یک کنترل کننده PID را با استفاده از الگوریتم بهینه سازی PSO طراحی کرده و برای کنترل سیستم AVR بصورت شکل (۳) پیاده سازی می کنیم. معیار طراحی کنترل کننده انتگرال خطا (ITAE) می باشد که بصورت زیر تعریف می شود.

$$ITAE = \int_0^t t|e|\,dt$$

لذا خواهیم داشت:



شکل (۳). شبیه سازی سیستم با کنترل کننده

سپس از الگوریتم PSO جهت طراحی کنترل کننده استفاده می کنیم.

```
clc;
clear all;
close all;

%% Problem Definition

CostFunction=@(x) MyCost(x);        % Cost Function
nVar=3;                             % Number of Variables
VarSize=[1 nVar];                   % Size of Variables Matrix
VarMin=0;                           % Lower Bound of Variables
VarMax=1;                           % Upper Bound of Variables
VarRange=[VarMin VarMax];           % Variation Range of Variables
VelMax=(VarMax-VarMin)/10;          % Maximum Velocity
VelMin=-VelMax;                     % Minimum Velocity

%% PSO Parameters

MaxIt=5;                            % Maximum Number of Iterations
nPop=100;                           % Swarm (Population) Size
phi1=2.05;
phi2=2.05;
phi=phi1+phi2;
chi=2/(phi-2+sqrt(phi^2-4*phi));
w=chi;
c1=2;
c2=2;

%% Initialization

empty_individual.Position=[];
empty_individual.Velocity=[];
empty_individual.Cost=[];
empty_individual.Out=[];
empty_individual.Best.Position=[];
empty_individual.Best.Cost=[];
empty_individual.Best.Out=[];
pop=repmat(empty_individual,nPop,1);
```

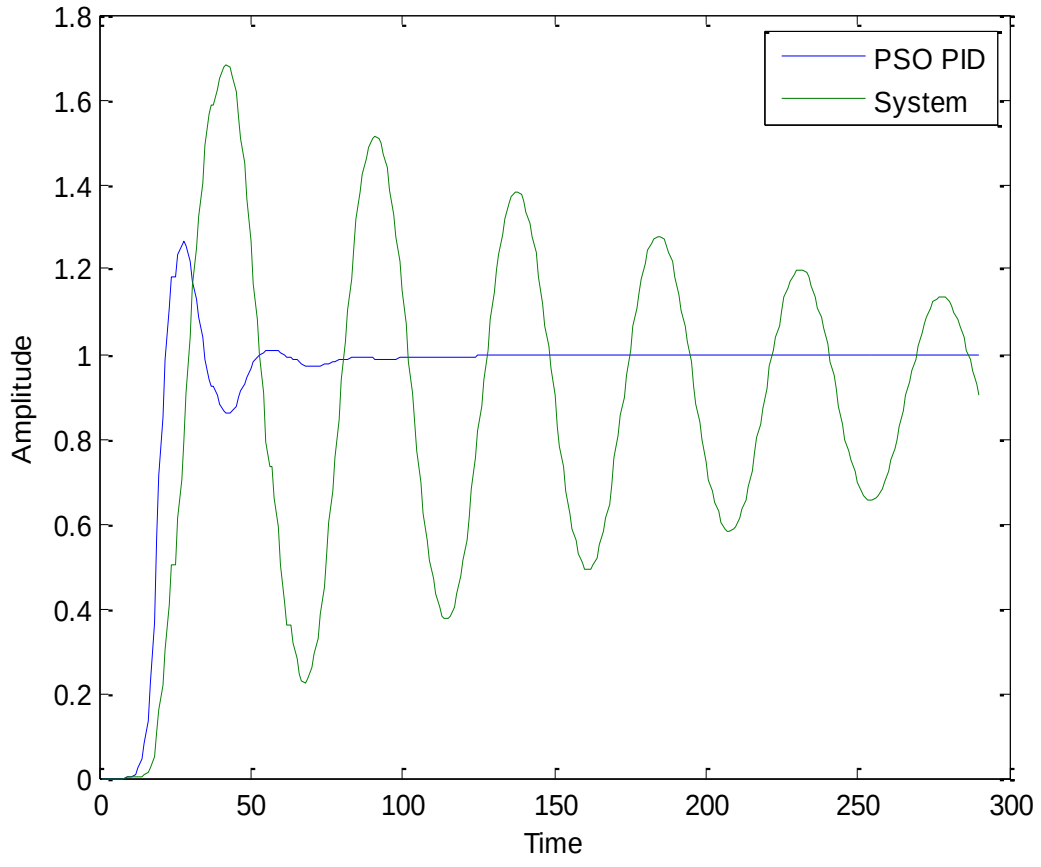
```
BestSol.Cost=inf;
for i=1:nPop
    pop(i).Position=unifrnd(VarMin,VarMax,VarSize);
    pop(i).Velocity=zeros(VarSize);
    [pop(i).Cost pop(i).Out]=CostFunction(pop(i).Position);
    pop(i).Best.Position=pop(i).Position;
    pop(i).Best.Cost=pop(i).Cost;
    pop(i).Best.Out=pop(i).Out;
    if pop(i).Best.Cost<BestSol.Cost
        BestSol=pop(i).Best;
    end
end
BestCost=zeros(MaxIt,1);

%% PSO Main Loop

for it=1:MaxIt
    for i=1:nPop
        pop(i).Velocity=w*pop(i).Velocity ...
            + c1*rand(VarSize).*(pop(i).Best.Position-pop(i).Position) ...
            + c2*rand(VarSize).*(BestSol.Position-pop(i).Position);
        pop(i).Velocity=min(max(pop(i).Velocity,VelMin),VelMax);
        pop(i).Position=pop(i).Position+pop(i).Velocity;
        flag=(pop(i).Position<VarMin | pop(i).Position>VarMax);
        pop(i).Velocity(flag)=-pop(i).Velocity(flag);
        pop(i).Position=min(max(pop(i).Position,VarMin),VarMax);
        [pop(i).Cost pop(i).Out]=CostFunction(pop(i).Position);
        if pop(i).Cost<pop(i).Best.Cost
            pop(i).Best.Position=pop(i).Position;
            pop(i).Best.Cost=pop(i).Cost;
            pop(i).Best.Out=pop(i).Out;
            if pop(i).Best.Cost<BestSol.Cost
                BestSol=pop(i).Best;
            end
        end
    end
    BestCost=BestSol.Cost;
    disp(['Iteration ' num2str(it) ': ITAE = ' num2str(BestCost)]);
end
 BestSol.Out
```

```
function [z out]=MyCost(x)
Kp=x(1);
Ki=x(2);
Kd=x(3);
set_param('Model/PID', 'P', num2str(Kp))
set_param('Model/PID', 'I', num2str(Ki))
set_param('Model/PID', 'D', num2str(Kd))
simout=sim('Model');
z=ITAE;
out.kp=Kp;
out.ki=Ki;
out.kd=Kd;
out.z=z;
end
```

در الگوریتم PSO تعداد متغیرهای طراحی برابر ۳، تعداد کل ذرات ۱۰۰ و تعداد تکرار الگوریتم ۵۰ مرتبه می باشد. پس از انجام شبیه سازی خواهیم داشت:



شکل (۴). پاسخ پله سیستم کنترل نشده و کنترل شده با کنترل کننده PID

لذا همانطور که مشاهده می شود، فراجهش سیستم کنترل شده با استفاده از کنترل کننده PID بهینه شده با الگوریتم PSO برابر با $Mp = 26.4\ \%$ می باشد.