

LAPORAN TUGAS AKHIR

KENDALI DIGITAL



Penalaan PID Berbasis *Genetic Algorithm*

Disusun Oleh : Muhamad Novida A. N. (18/427496/PA/18456)
Muhammad Fadhlan (18/427500/PA/18460)
Muhammad Ammar M. (18/427498/PA/18458)

PROGRAM STUDI ELEKTRONIKA DAN INSTRUMENTASI
FAKULTAS MATEMATIKA DAN ILMU PENGETAHUAN ALAM
UNIVERSITAS GADJAH MADA
2021

I. PENDAHULUAN

Sistem kendali merupakan kumpulan metode untuk mengendalikan suatu proses agar sesuai dengan apa yang diharapkan. Dalam aplikasinya, sistem kendali berperan untuk mengatur keluaran (*output*) dalam suatu sistem berdasarkan suatu masukan (*input*) melalui elemen kontrol. Sistem kendali yang optimal diperlukan untuk menjalankan suatu proses didalam *plant* yang didapatkan melalui penalaan pada suatu sistem. Penalaan suatu sistem salah satunya dapat dilakukan dengan menggunakan metode kendali PID (*Proportional, Integral, and Derivative*)

Metode kendali PID adalah metode yang dapat digunakan untuk melakukan penalaan pada suatu sistem, untuk mendapatkan respon sistem yang lebih baik, untuk melakukan penalaan pada suatu sistem dengan metode kendali PID dapat menggunakan metode manual dengan perhitungan matematis, dan juga dapat menggunakan metode kecerdasan buatan, pada tugas ini kami akan menggunakan metode kecerdasan berupa Algoritma Genetika (*Genetic Algorithm*)

Algoritma Genetika merupakan algoritma yang bekerja secara acak yang dikembangkan untuk meniru mekanisme seleksi alam dan genetika alam. Algoritma ini bekerja dengan aturan *survival of the fittest* dengan menggunakan pertukaran informasi yang acak namun terstruktur.

II. TINJAUAN PUSTAKA

Kendali PID adalah salah satu metode yang dapat digunakan untuk control suatu sistem dengan menggunakan 3 buah konstanta kendali, konstanta-konstanta tersebut adalah konstanta *proportional*, konstanta *integral* dan konstanta *derivative* [1].

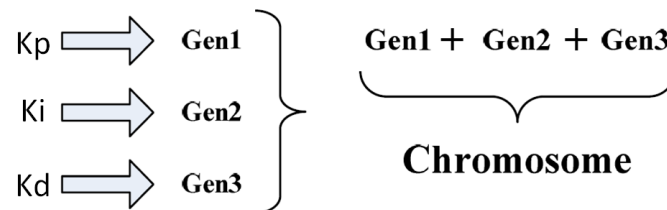
Diskritisasi ZOH adalah suatu metode dalam proses diskritisasi untuk men-holding sinyal dari satu waktu sampling ke waktu sampling selanjutnya [1].

Genetic Algorithm adalah algoritma pencarian acak yang telah dikembangkan dalam upaya untuk meniru mekanisme seleksi alam dan genetika alam [2], *Genetic Algorithm* beroperasi pada struktur string, seperti struktur biologis, yang berkembang dalam waktu sesuai dengan aturan *survival of the fittest* dengan menggunakan pertukaran informasi yang acak namun terstruktur.

Parameter pada *Genetic Algorithm* disebut sebagai parameter kontrol. Parameter kontrol dapat berupa ukuran populasi, probabilitas crossover, mutasi, gap generasi, strategi seleksi dan

skala fungsi[3]. *Genetic Algorithm* bekerja untuk menemukan kontrol yang optimal dalam pengontrolan suatu sistem.

Penalaan PID yang memanfaatkan *Genetic Algorithm* menggunakan pendekatan pada koefisien PID-nya. Koefisien PID dikodekan dalam satu kromosom. Struktur kromosom dibuat untuk mewakili koefisien pada penalaan PID yang dapat dilihat pada gambar dibawah ini.

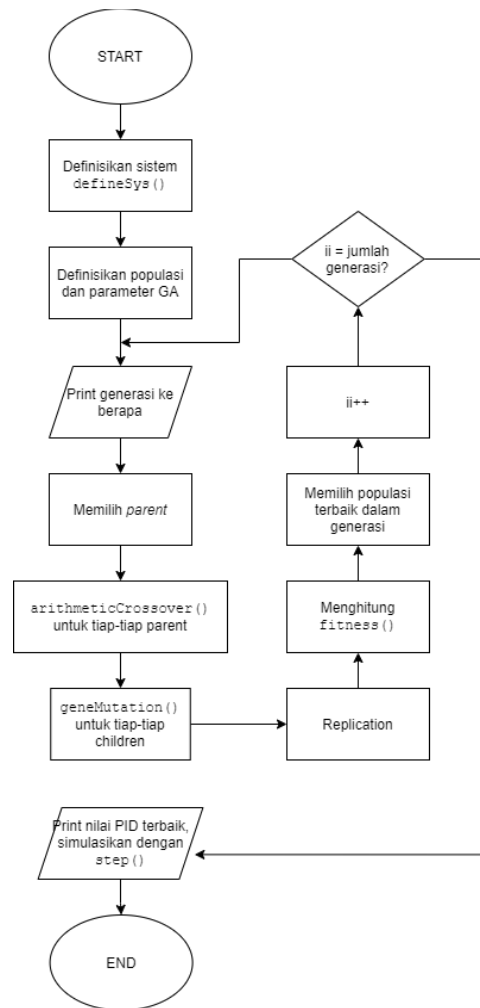


Gambar 2.1 Struktur kromosom untuk PID

Untuk melakukan penalaan menggunakan metode PID dengan menggunakan *Genetic Algorithm*, langkah pertama adalah menginisialisasi populasi pada PID populasi yang dibentuk adalah objek yang mengandung kp , ki dan kd [4], kemudian mengevaluasi nilai *fitness*, kemudian memilih 2 *parents* dengan nilai *fitness* yang terbaik [4], selanjutnya melakukan mutasi pada objek yang memiliki nilai *fitness* yang kurang baik, dengan tujuan untuk mendapatkan nilai *fitness* yang lebih baik dan memasukan nya ke populasi kembali, kemudian hal ini akan dilakukan secara berulang hingga mencapai nilai variabel *generation*, pada program yang kami gunakan variabel *generation* yang digunakan sebesar 200.

III. HASIL DAN PEMBAHASAN

Dalam eksperimen ini telah dikembangkan suatu program yang dapat melakukan penalaan PID menggunakan *genetic algorithm* (GA) berdasarkan spesifikasi *step response* yang diberikan. Keseluruhan *listing code* dari program yang disusun telah tersedia di **Lampiran**. Program penalaan PID dengan GA ini terdiri dari lima file MATLAB. Program utama dijalankan pada file `main.m` yang dijelaskan pada diagram alur pada **gambar 3.1**.



Gambar 3.1 Diagram Alur `main.m`

Pada alur kerja utama, terdapat langkah-langkah metode GA yang telah diimplementasi ulang menggunakan *code from scratch*. GA yang digunakan pada aplikasi ini ditulis ulang sehingga dapat diketahui langkah-langkahnya. Secara umum, langkah-langkah metode GA dilakukan sesuai dengan iterasi jumlah generasi maksimal. Langkah-langkah tersebut antara lain pemilihan *parent*, *crossover*, mutasi, replikasi, dan perhitungan *fitness*. Langkah-langkah GA dijelaskan sebagai berikut:

1. Pemilihan *parent*

Proses pemilihan *parent* dilakukan secara acak. Salah satu individu dalam setiap populasi dipilih dengan menggunakan perintah `geornd(0.1)+1` agar didapatkan individu yang acak. Pemilihan *parent* kemudian disimpan pada variabel `y1` dan `y2`. Proses ini dilakukan dengan

menyesuaikan jumlah *crossover* yang telah didefinisikan sebelumnya dalam variabel `crossoverTimes`.

2. *Crossover*

Proses *crossover* menghasilkan *children* atau anak dari persilangan dua *parent* yang telah dipilih pada proses sebelumnya. Dalam implementasi ini, proses *crossover* dilakukan dengan memanfaatkan fungsi `arithmeticCrossover()` yang mengambil parameter dua *parent*. Fungsi `arithmeticCrossover()` dituliskan sebagai berikut:

```
function [children] = arithmeticCrossover(parent1, parent2)
    alpha = rand;
    child1 = alpha*parent1 + (1-alpha)*parent2;
    child2 = (1-alpha)*parent1 + alpha*parent2;
    children = [child1; child2];
end
```

Dapat kita perhatikan bahwa proses persilangan atau *crossover* memanfaatkan nilai acak yang disimpan pada variabel **alpha**. Dengan tanpa parameter, fungsi `rand` menghasilkan nilai acak dari interval 0 hingga 1, sehingga proses persilangan tidak dilakukan dengan cara pertukaran kromosom melainkan dengan cara penambahan atau adisi dari kedua *parent*. Hal ini lebih efektif dalam aplikasi penalaan PID karena nilai K_p , K_i , dan K_d merupakan nilai yang tidak bisa ditukar satu sama lain.

3. *Mutasi*

Proses mutasi dilakukan untuk memberikan variasi pada suatu populasi. Pada aplikasi yang dibuat, proses mutasi dilakukan pada *parent* untuk menghasilkan *child* tanpa melakukan persilangan. Mutasi dilakukan dengan memilih kromosom secara acak dengan menggunakan fungsi `randsample`. Setelah dipilih satu kromosom, kemudian kromosom tersebut dikalikan dengan angka acak dari interval 0 hingga 1 untuk kemudian dikurangi dengan nilai awal. Keseluruhan proses mutasi ini dilakukan dengan memanfaatkan fungsi `geneMutation()`.

4. Replikasi

Replikasi dilakukan untuk mengambil satu individu secara acak dari generasi sekarang untuk diturunkan ke generasi selanjutnya. Replication dilakukan untuk individu yang tidak mengalami mutasi ataupun hasil persilangan *parent*.

5. Perhitungan *fitness function*

Fungsi *fitness* atau *fitness function* menggambarkan kecocokan suatu individu dengan lingkungannya. Pada kasus ini, *fitness* dari sebuah individu didefinisikan sebagai nilai kesalahan atau selisih dari nilai parameter *step respon* yang diinginkan. Fungsi *fitness* didefinisikan pada fungsi `fitness()` yang isinya sebagai berikut:

```
function err = fitness(x, plant, Ts, desired)

% Fungsi ini mendefinisikan sistem yang terkontrol dengan PID
% Parameter: x -> [Kp, Ki, Kd]
%           plant -> sistem Diskrit
%           Ts -> waktu sampling
%           desired -> keinginan spesifikasi

c = pid(x(1), x(2), x(3), 0, Ts);

system = feedback(series(c, plant), 1);
sim = stepinfo(system);

riseTimeError = sim.RiseTime - desired.RiseTime;
overShootError = sim.Overshoot - desired.Overshoot;
settlingTimeError = sim.SettlingTime - desired.SettlingTime;

err = riseTimeError + overShootError + settlingTimeError;
end
```

Dapat diperhatikan apabila untuk menghitung nilai *fitness*, perlu dilakukan simulasi dengan menggunakan `stepinfo()` untuk mendapatkan parameter *rise time*, *overshoot*, dan *settling time* dari sistem dengan suatu kromosom PID. Setelah itu, nilai error dari tiap-tiap parameter diagregasi dengan menambahkan keseluruhan error menjadi satu variabel `err`. Cara seperti ini memiliki satu kelemahan yaitu tidak adanya *weighting* atau pembebanan pada

parameter tertentu, sehingga dimungkinkan proses GA hanya melakukan optimasi pada salah satu parameter saja.

Dalam eksperimen yang dilakukan, digunakan suatu sistem yang direpresentasi dalam model ruang keadaan (*state space*). Sistem tersebut didefinisikan sebagai berikut:

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \\ \dot{x}_3 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ -18 & -19 & -12 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} u$$
$$y = \begin{bmatrix} 30 & 0 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}$$

Kemudian, sistem ini ingin dikendalikan secara digital menggunakan PID. Kendali PID harus diatur sedemikian sehingga memenuhi spesifikasi berikut saat dikenai input *step*:

- *Rise time* kurang dari 0.2 detik
- *Settling time* kurang dari 2 detik
- *Overshoot* kurang dari 11%

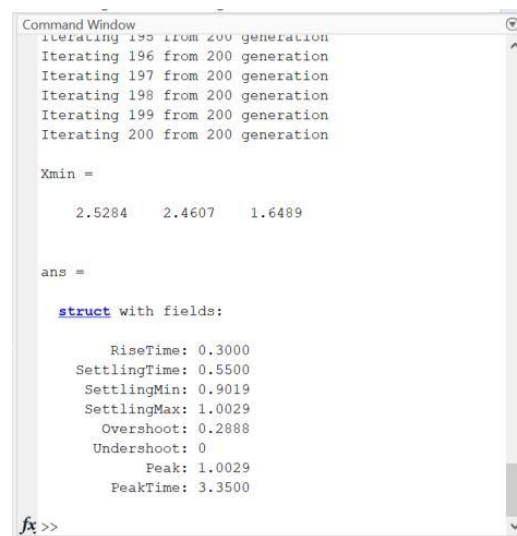
Untuk itu, kita perlu mengubah `defineSys.m` agar sesuai dengan sistem dan spesifikasi yang kita inginkan. Perintah pada `defineSys.m` diatur menjadi sebagai berikut:

```
function [plant_d, desired, Ts] = defineSys()
    A = [0 1 0; 0 0 1; -18 -19 -12];
    B = [0; 0; 1];
    C = [30 0 0];
    D = [];
    plant = ss(A, B, C, D);

    Ts = 0.05;
    plant_d = c2d(plant, Ts, 'ZOH');

    desired = struct('RiseTime', 0.19, ...
                    'SettlingTime', 1.9, ...
                    'Overshoot', 10.5);
end
```

Langkah selanjutnya adalah menjalankan `main.m`. Program ini diatur untuk berjalan sebanyak 200 generasi. Program ini juga menyimpan individu terbaik di setiap generasinya. Setelah GA selesai melakukan iterasi, akan dicetak individu terbaik sebagai parameter PID untuk kendali sistem. Berikut adalah hasil output pada *console window* MATLAB setelah program selesai berjalan, serta grafik respon sistem dan nilai *fitness* tiap generasi.



```
Command Window
Iterating 195 from 200 generation
Iterating 196 from 200 generation
Iterating 197 from 200 generation
Iterating 198 from 200 generation
Iterating 199 from 200 generation
Iterating 200 from 200 generation

Xmin =

    2.5284    2.4607    1.6489

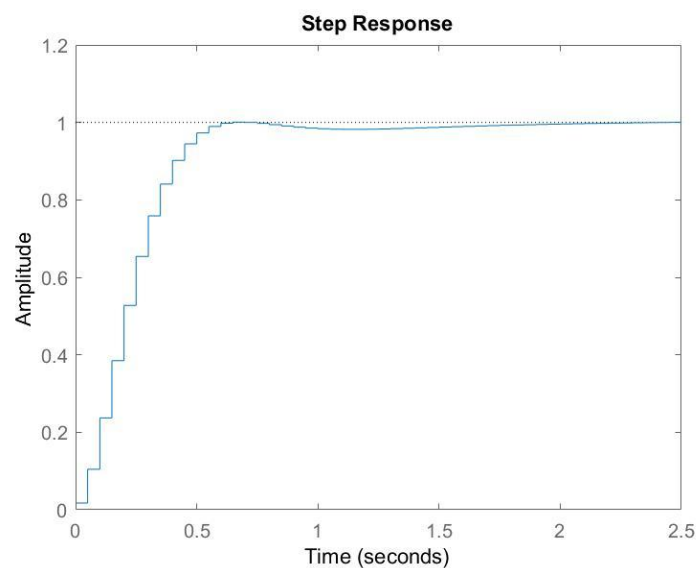
ans =

struct with fields:

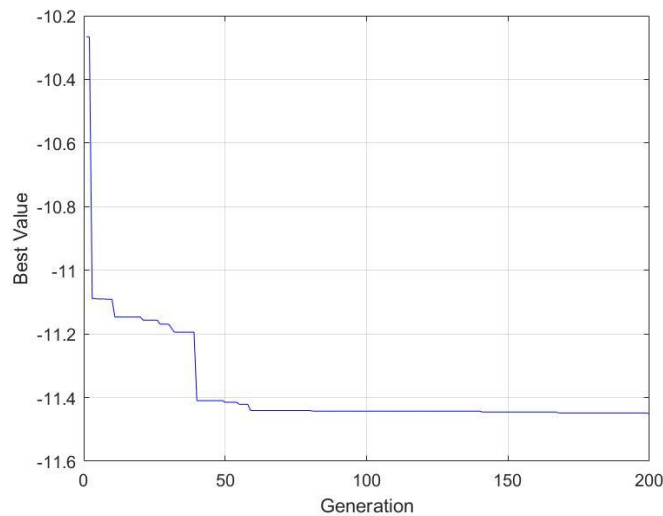
    RiseTime: 0.3000
    SettlingTime: 0.5500
    SettlingMin: 0.9019
    SettlingMax: 1.0029
    Overshoot: 0.2888
    Undershoot: 0
    Peak: 1.0029
    PeakTime: 3.3500

fx >>
```

Gambar 3.2 Tampilan *Console Window*



Gambar 3.3 Tampilan Respon Sistem



Gambar 3.4 Nilai *Fitness* Setiap Generasi

Dapat diperhatikan bahwa hasil `stepinfo()` dari simulasi sistem menghasilkan hasil yang cukup menarik. Waktu untuk *settling time* dan *overshoot* sudah sesuai dengan spesifikasi. Namun untuk *rise time* masih belum. Hal ini dikarenakan kekurangan pada pendefinisian *fitness function* dimana selisih dari *overshoot* memenuhi nilai *fitness*. Kekurangan ini sebetulnya dapat diatasi dengan cara melakukan normalisasi interval antara *overshoot*, *settling time*, dan *rise time*.

IV. KESIMPULAN

Genetic Algorithm dapat digunakan untuk penalaan suatu sistem dengan kendali PID, mula-mula objek yang mengandung kp , ki dan kd diinisialisasikan, kemudian dari nilai yang sudah diinisialisasikan tersebut akan dipilih objek dengan dengan nilai *fitness* yang terbaik, kemudian dilakukan proses *crossover* pada objek dengan nilai *fitness* yang kurang baik, dengan tujuan untuk mendapatkan hasil yang lebih baik, dan memasukan nya lagi ke populasi, kemudian hal ini akan dilakukan secara berulang hingga mencapai nilai *variable generation*, pada program yang kami gunakan *variable generation* yang digunakan sebesar 200.

V. DAFTAR PUSTAKA

- [1] Fadali, M. S., & Visioli, A. (2013). *Digital control engineering: analysis and design*. Academic Press.
- [2] Mirjalili, S. (2019). Genetic algorithm. In *Evolutionary algorithms and neural networks* (pp. 43-55). Springer, Cham.
- [3] Yeniay, Ö.: An Overview of Genetic Algorithms. *Anadolu Üniversitesi Bilim ve Teknoloji Dergisi* 2, 37–49 (2001)
- [4] Obitko.com. 2021. *Operators of GA - Introduction to Genetic Algorithms - Tutorial with Interactive Java Applets*. [online] Available at: <https://www.obitko.com/tutorials/genetic-algorithms/operators.php> [Accessed 19 June 2021].

VI. LAMPIRAN

Listing program yang digunakan pada eksperimen ini dapat diakses pada *repository* berikut: <https://github.com/fadhln/autotune-PID-GA-scratch>

main.m

```
clc; clear; close all;

%% Define system
[plant_d, desired, Ts] = defineSys();

%% Params of GA
% Define Lower Bound (LB) and Upper Bound (UB)
LB = [0 0 0];
UB = [5 5 5];
nVars = 3;

% Max number of generation
generation = 200;
populationSize = 50;

% Initialize population
population = zeros(populationSize, nVars);
tempPopulation = zeros(populationSize, nVars);
mutRate = 0.1;

sPopulation = zeros(populationSize, nVars+1);
F = zeros(populationSize, 1);
crossoverTimes = 4;
mutationTimes = 5;

for i=1:1:populationSize
    for n=1:1:nVars
        % Assign random number for population
        population(i, n) = unifrnd(LB(1,n), UB(1,n));
    end
    F(i) = fitness(population(1,:), plant_d, Ts, desired);
end
```

```

% Sorting of the population
sPopulation(:, 1:nVars) = population(:, :);
sPopulation(:, nVars+1) = F;
sPopulation = sortrows(sPopulation, nVars+1);

%% Selection and Crossover
% Generation
for ii=1:1:generation

    txt = sprintf('Iterating %d from %d generation', ii, generation);
    disp(txt);

    k=1;

    % Elitism
    tempPopulation(k, 1:nVars) = sPopulation(1, 1:nVars);
    k=k+1;

    % Parent Selection
    for j=1:1:crossoverTimes
        y1(j) = geornd(0.1)+1;
        while y1(j) > populationSize
            y1(j) = geornd(0.1)+1;
        end
        y2(j) = geornd(0.1)+1;
        while y2(j) > populationSize
            y2(j) = geornd(0.1)+1;
        end
    end

    % Arithmetic Crossover
    for u=1:1:crossoverTimes
        parent1 = sPopulation(y1(u), 1:nVars);
        parent2 = sPopulation(y2(u), 1:nVars);

        [children] = arithmeticCrossover(parent1, parent2);
        tempPopulation(k, 1:nVars) = children(1, :);

        tempPopulation(k, 1:nVars) = max(tempPopulation(k, 1:nVars), LB);
        tempPopulation(k, 1:nVars) = min(tempPopulation(k, 1:nVars), UB);
    end
end

```

```

    k=k+1;
    tempPopulation(k, 1:nVars) = children(2,:);

    tempPopulation(k, 1:nVars) = max(tempPopulation(k, 1:nVars), LB);
    tempPopulation(k, 1:nVars) = min(tempPopulation(k, 1:nVars), UB);

    k=k+1;
end

% Mutation
for e=1:1:mutationTimes
    parent = sPopulation(unidrnd(populationSize), 1:nVars);
    [child] = geneMutation(parent, mutRate, LB, UB);

    tempPopulation(k, 1:nVars) = child;
    tempPopulation(k, 1:nVars) = max(tempPopulation(k, 1:nVars), LB);
    tempPopulation(k, 1:nVars) = min(tempPopulation(k, 1:nVars), UB);
    k = k+1;
end

% Replication
for k=k:1:populationSize
    replicatedChild = sPopulation(randi([1 populationSize]), 1:nVars);
    tempPopulation(k, 1:nVars) = replicatedChild;
    k = k+1;
end

% Calculate fitness Function
for iii=1:1:populationSize
    F(iii) = fitness(tempPopulation(iii,:), plant_d, Ts, desired);
end

% Sorting for next generation
sPopulation(:, 1:nVars) = tempPopulation;
sPopulation(:, nVars+1) = F(:,:);
sPopulation = sortrows(sPopulation, nVars+1);

bestF(ii) = sPopulation(1, nVars+1);
end

```

```

figure(1);
plot(bestF, 'b');
xlabel('Generation');
ylabel('Best Value');
grid on

%% Simulation of best Chromosome
Xmin = sPopulation(1,1:nVars)
Eval = bestF(1, generation);

c = pid(Xmin(1), Xmin(2), Xmin(3), 0, Ts);
system = feedback(series(c, plant_d),1);
figure(2)
step(system)
stepinfo(system)

```

defineSys.m

```

function [plant_d, desired, Ts] = defineSys()
    A = [0 1 0; 0 0 1; -18 -19 -12];
    B = [0; 0; 1];
    C = [30 0 0];
    D = [];
    plant = ss(A, B, C, D);

    Ts = 0.05;
    plant_d = c2d(plant, Ts, 'ZOH');

    desired = struct('RiseTime', 0.19, ...
                    'SettlingTime', 1.9, ...
                    'Overshoot', 10.5);
end

```

fitness.m

```

function err = fitness(x, plant, Ts, desired)

% Fungsi ini mendefinisikan sistem yang terkontrol dengan PID
% Parameter: x -> [Kp, Ki, Kd]

```

```

%           plant -> sistem Diskrit
%           Ts -> waktu sampling
%           desired -> keinginan spesifikasi

c = pid(x(1), x(2), x(3), 0, Ts);

system = feedback(series(c, plant), 1);
sim = stepinfo(system);

riseTimeError = sim.RiseTime - desired.RiseTime;
overShootError = sim.Overshoot - desired.Overshoot;
settlingTimeError = sim.SettlingTime - desired.SettlingTime;

err = riseTimeError + overShootError + settlingTimeError;
end

```

arithmeticCrossover.m

```

function [children] = arithmeticCrossover(parent1, parent2)
    alpha = rand;
    child1 = alpha*parent1 + (1-alpha)*parent2;
    child2 = (1-alpha)*parent1 + alpha*parent2;
    children = [child1; child2];
end

```

geneMutation.m

```

function [child] = geneMutation(parent, mutRate, LB, UB)
    nVars = numel(parent);
    nmu = ceil(mutRate * nVars);
    j = randsample(nVars, 1);
    child = parent;
    child(j) = parent(j) - rand*parent(j);
end

```