PID Gain Optimization for an Internal Combustion Engine at Idle Speed

Final Project for Optimization Theory & Practice

Rajdeep Arora & Yi Cheng Yeh
(Group 7)
December 7, 2017

CONTENTS

List of Figures

List of Tables

1. General Overview

The most common method for regulating the output of a controllable system is a PID controller. With this method, the output error, the integration of the error, and the derivative of the error is added together and fed back as a control value through Feedback Loop. The weight each component carries into the control value is defined by the gain of each measurement (proportional, integral, and derivative). It is usually up to the designer to manually tune the gains to achieve an ideal output response. This project implements an optimization routine in place of manual tuning to select the ideal proportional, integral, and derivative gains (Kp, Ki, and Kd, respectively) for idle speed control of an internal combustion engine using the Simulink model shown in Figure 1.

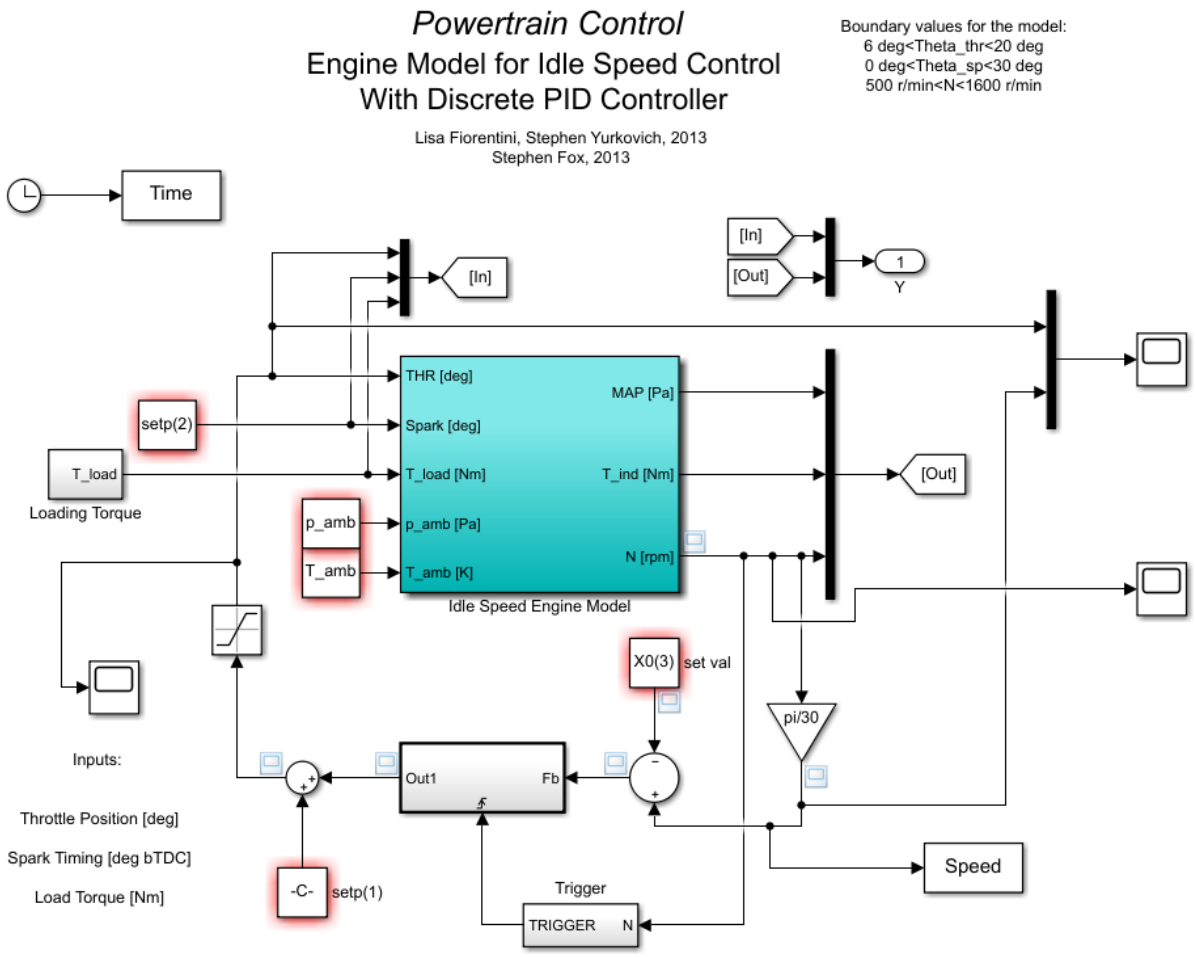


Figure 1: Simulink Diagram

2. Manual Tuning

Without a PID controller (or any controller for that matter) the engine will stall, as shown in the open-loop response in Figure 2. By manually tuning the PID gains using common heuristics, the controller can regulate the engine speed, as depicted in Figure 3. The hand-tuned gains are not always optimal though. Optimal gains are determined by the minimization a cost function that relates to the total error observed in the simulation. The cost function used in this project is the summed squared error, absolute error and time variant of errors to focus on the reduction of excursion range and settling time. There are numerous optimization algorithms available to attempt the optimization, but the highly non-linear nature of the internal combustion engine model incentivizes the use of local search algorithm called as FMINCON and global search algorithms. The global search algorithm employed in this project is the Particle Swarm Optimization (PSO) algorithm. Sections 2 and 3 introduce the PSO algorithm. The results of the optimization are discussed in Section 4, and Section 5 summarizes the project's results.

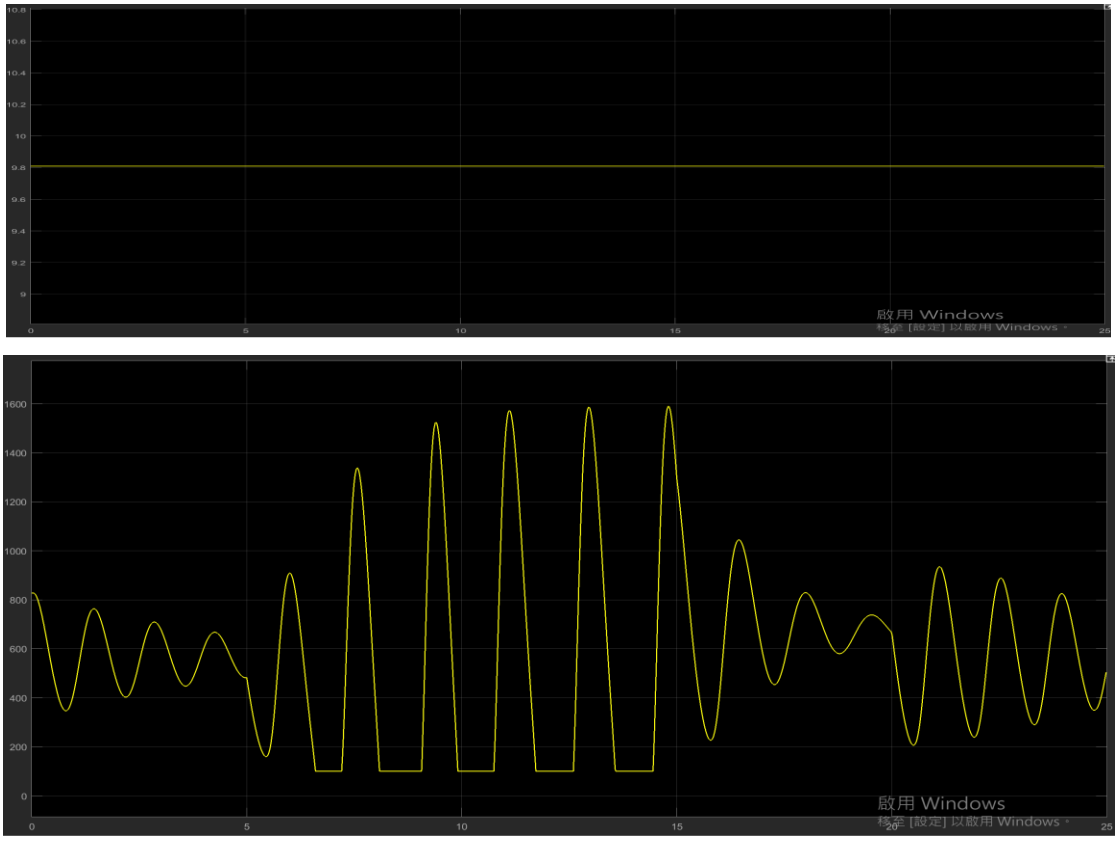


Figure 2: Open Loop response

Figure 3: Closed Loop response with KP = 0.225, KI = 0.01, KD = 4.25

		Open Loop	Closed Loop		
PID Gains			KP = 0.1, KI = 0.005, KD = 1	KP = 0.2, KI = 0.005, KD = 2	KP = 0.225, KI = 0.01, KD = 4.25
Engine Speed	Max	1590	964.5	936.9	922.2
	Min	100	636	677.8	707.4
	Mean	766.8	798.8	799.2	800.6
Throttle Angle	Max	9.8	14.7	16.72	20
	Min	9.8	6.835	6	6
	Mean	9.8	7.9	11.73	11.71

Table 1: Engine Speed and Throttle Angle values for Manual Tuned PID Control

3. FMINCON Optimization

FMINCON finds a constrained minimum of a scalar function of several variables starting at an initial estimate. This is generally referred to as constrained nonlinear optimization or nonlinear programming. x = fmincon(fun,x0,A,b) starts at x0 and finds a minimum x to the function described in fun subject to the linear inequalities A*x <= b. x0 can be a scalar, vector, or matrix.

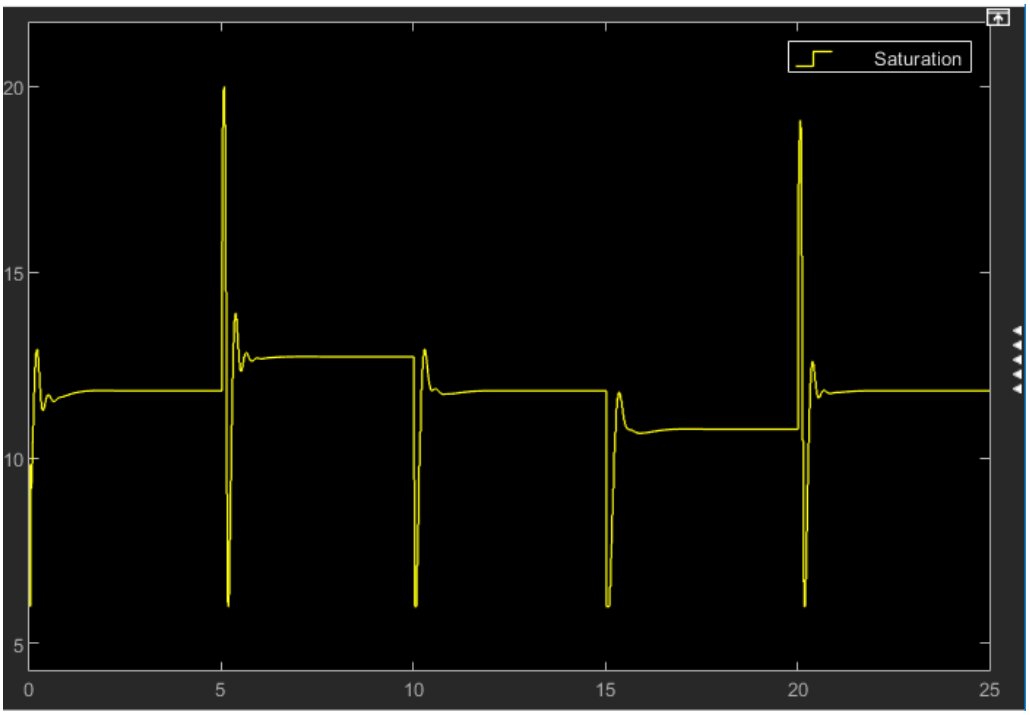
x = fmincon(fun,x0,A,b,Aeq,beq) minimizes fun subject to the linear equalities Aeq*x = beq as well as A*x <= b. Set A=[] and b=[] if no inequalities exist. x = fmincon(fun,x0,A,b,Aeq,beq,lb,ub) defines a set of lower and upper bounds on the design variables, x, so that the solution is always in the range lb <= x <= ub.

x = fmincon(fun,x0,A,b,Aeq,beq,lb,ub,nonlcon) subjects the minimization to the nonlinear inequalities c(x) or equalities ceq(x) defined in nonlcon. fmincon optimizes such that c(x) <= 0 and ceq(x) = 0.

$$\min_x f(x) \text{ such that } \begin{cases} c(x) \le 0 \\ ceq(x) = 0 \\ A \cdot x \le b \\ Aeq \cdot x = beq \\ lb \le x \le ub, \end{cases}$$

Figure 4: FMINCON Optimization

We have selected excursion range, throttle range, settling time,iterations and cost function values as the metrics to identify the best PID gains. As all the three objective functions namely absolute error, norm square absolute error and time variant norm error provides different dimension to metrics; decision has been taken in account to a specific objective function. However, as time variant norm error adds time dimension to cost value, it is the best measure for identifying the performance of the optimization algorithm.



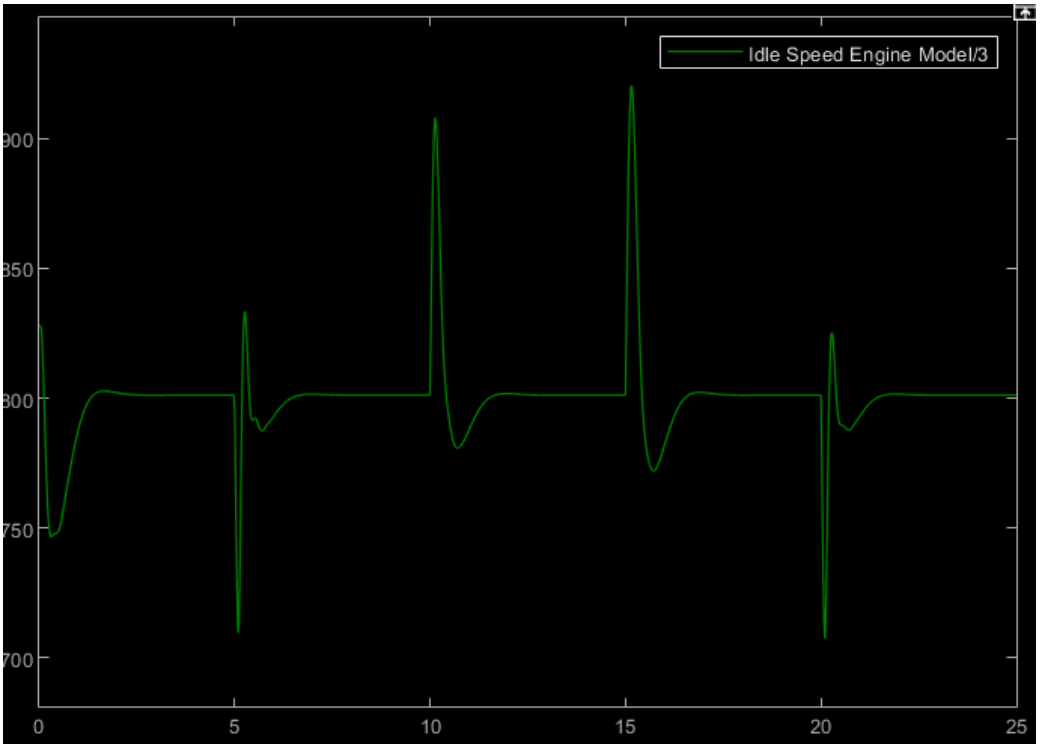


Figure 5: Closed Loop FMINCON Optimization T*Norm Square Objective Function Response with Initial KP = 0.225, KI = 0.01, KD = 4.25

Absolute Error Objective Function			
		Closed Loop	
Initial PID Gains		KP = 0.1, KI = 0.005, KD = 1	KP = 0.225, KI = 0.01, KD = 4.25
Engine Speed	Max	1017	921
	Min	556	706
	Mean	796	800
Throttle Angle	Max	13.37	20
	Min	7.37	6
	Mean	11.7	14
Final PID Gains		KP = 0.01, KI = 0.001, KD = 0.89	KP = 0.27, KI = 0.01, KD = 4.09
Metrics		1.17E+07	2.70E+06

Table 2: Engine Speed, Throttle Angle, Function Value and Final PID gains for Absolute Error Objective Function

Norm Square Objective Function			
		Closed Loop	
Initial PID Gains		KP = 0.01, KI = 0.001, KD = 1	KP = 0.225, KI = 0.01, KD = 4.25
Engine Speed	Max	1014	920
	Min	561	709
	Mean	794	800
Throttle Angle	Max	13.37	20
	Min	7.37	6
	Mean	11.7	14
Final PID Gains		KP = 0.01, KI = 0.001, KD = 1	KP = 0.27, KI = 0.007, KD = 4.3738
Metrics		1.24E+07	2.85E+06

Table 3: Engine Speed, Throttle Angle, Function Value and Final PID gains for Norm Square Objective Function

Time Variant Norm Square Objective Function			
		Closed Loop	
Initial PID Gains		KP = 0.01, KI = 0.001, KD = 1	KP = 0.225, KI = 0.01, KD = 4.25
Engine Speed	Max	1014	925
	Min	561	707
	Mean	791	800
Throttle Angle	Max	13.37	20
	Min	7.37	6
	Mean	11.6	11.7
Final PID Gains		KP = 0.01, KI = 0.001, KD = 1	KP = 0.2058 KI = 0.007, KD = 4.24
Metrics		1.24E+07	2.83E+06

Table 4: Engine Speed, Throttle Angle, Function Value and Final PID gains for Time Variant Norm Square Objective Function

4. Particle Swarm Optimization – Introduction

Particle Swarm Optimization (PSO) is a global search algorithm that uses numerous “particles" and their respective costs to find the optimal policy. In this project's case, the “particles" are points in 3-D space corresponding to the Kp, Ki, and Kd values. The swarm of particles interact with each other to drive the swarm to the best location in the defined space, thus resulting in the optimal policy.
As PSO algorithm iterates, each particle keeps track of its own best-so-far position as well as the swarm's best-so-far position, also called the “global-best" position. The particle updates by adding a velocity vector to the particle. This velocity vector has an element of randomness to it, to allow the particle to explore new values. The closer the particle is to its best-so-far position and/or the global-best position, however, the less randomness is introduced to the velocity vector; the velocity will more closely resemble the velocity that brought the particle to this position. How much the particles learn from their own history versus the swarm's history is governed by hyperparameters c1 and c2, known as the “cognitive" and “social" components, respectively. If c1 is larger than c2, the movement of each particle is controlled more by its best-so-far position. Comparatively, if c2 is larger than c1, the movement of each particle is controlled more by the global-best position.
Because of the randomness of particle movement, it is common for the global-best to remain unchanged for several iterations before a new global-best position to be discovered. Due to this behavior, the stopping criteria for PSO should not be a tolerance in the change in cost. PSO should stop once the cost reaches a set threshold value, or after a set number of iterations. In this project, we are not seeking a specific cost. Rather we are seeking a desired system response. Thus, the number of iterations was fixed at 10.

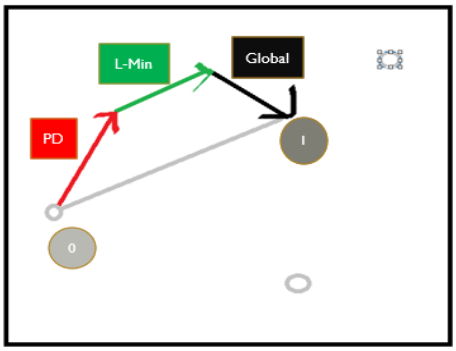


Figure 6: Particle Swarm Optimization Randomness, Social and Cognitive factors in velocities

5. Particle Swarm Optimization Algorithm

To begin, the initial PID values are used to construct a set of n particles. A random number from an equal distribution of [-1,1] is added to each particle's components to create the initial position of the swarm. The first particle is left unchanged to account for the possibility that the initial PID values are the optimal values. For each particle, an initial velocity vector, v(i), is generated randomly from an equal distribution of [-1,1].
The cognitive and social components c1 and c2 are defined.
For each particle, the simulation is run and the cost is calculated. The ith cost is assigned to C(i) for each particle, the ith particle's best-so-far cost , and the ith position (vector of PID gains) is assigned to Pi, the particle's best-so-far position. The global-best cost is assigned to Cg and the corresponding position is assigned to G. The iteration then begins.
Step 1. Vectors r and s are randomly generated with values coming from an equal distribution of [0,1].
Step 2. The ith particle's velocity, v(i), is updated with by Equation 1.

$$v_{k+1}^{(i)} = \omega v_k^{(i)} + c_1 r(P_i - X_k^{(i)}) + c_2 s(G - X_k^{(i)})$$

Step 3. The ith particle's position is then updated by Equation 2.

$$X_{k+1}^{(i)} = X_k^{(i)} + v_{k+1}^{(i)}$$

Step 4. With new particle positions, the simulation is run for each particle and the corresponding costs are recorded.
Step 5. If the ith particle's new cost is less than, Pi, C(i) and Pi are updated accordingly.
Step 6. The minimum of the swarm's costs is compared to the Cg. If the new minimum is better, Cg and G are updated accordingly.
Step 7. If the stopping criteria has been reached, stop. Otherwise, reiterate.

6. Hyperparameter Tuning

There is a trade-off for the search of Global Minima with change in social and cognitive parameters for POS algorithm.

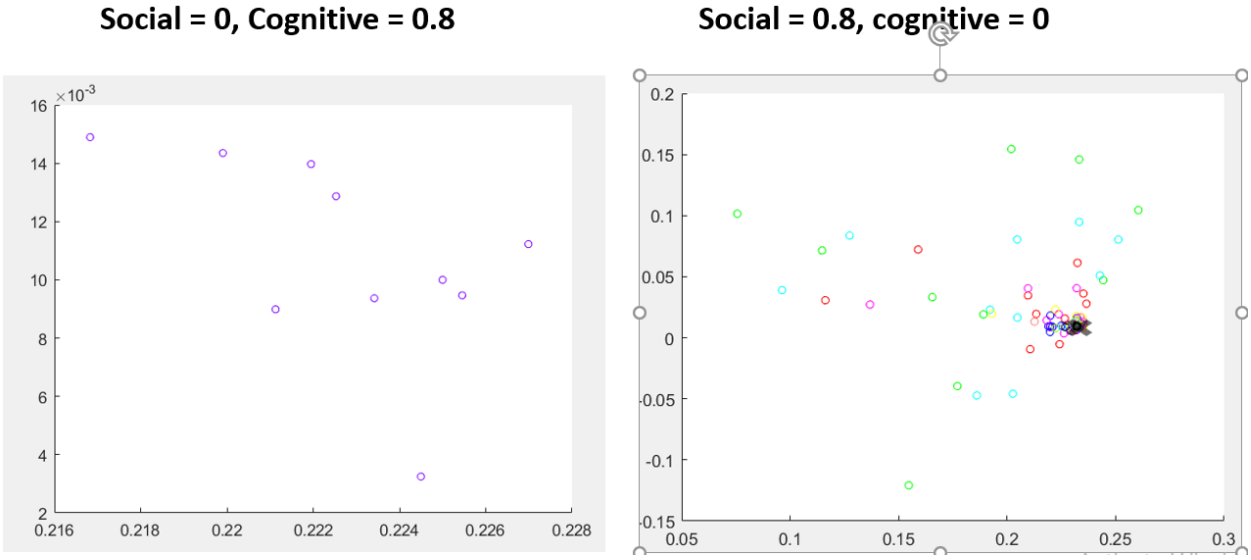


Figure 7: POS Algorithm convergence after 10 iterations with Social and Cognitive parameters impact

For low social parameters, the convergence is negligible as the particle stays at the minima and doesn't have an impact of global minima information. However, as the social parameter grows up and cognitive drown to 0; the algorithm converges in 3 iterations suggesting less exploration. It means the algorithm converges to the first minima it finds and doesn't explore for global minima in the second case.

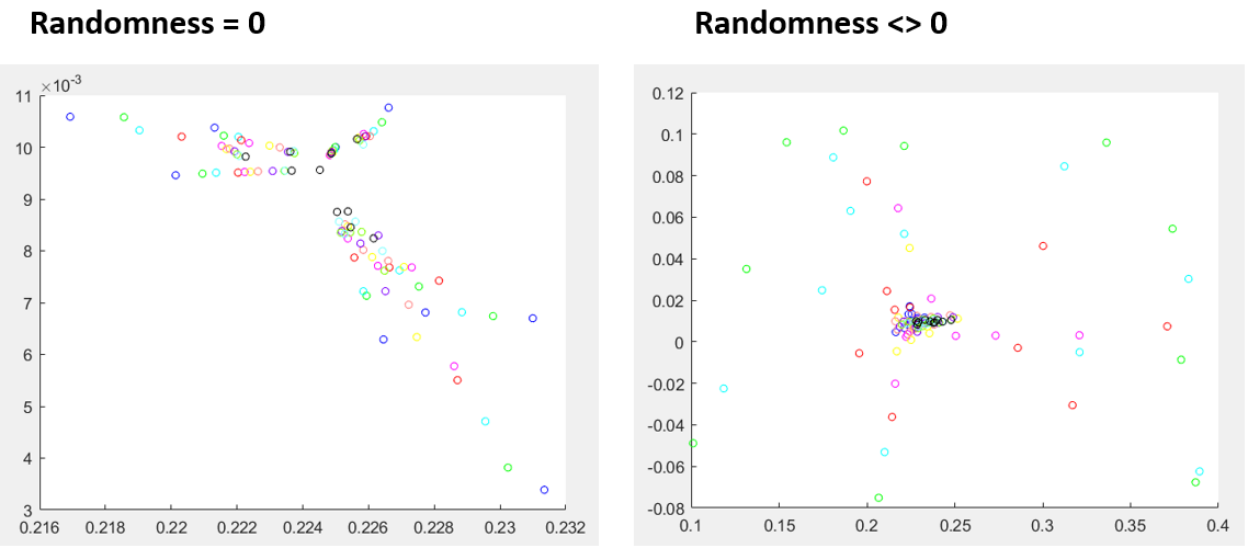


Figure 8: POS Algorithm convergence after 10 iterations with random parameters impact

For low randomness parameter in the POS algorithm, there is a significant impact in the performance of the algorithm. For low random coefficient, the convergence is slow and to the first minima the algorithm finds however if the randomness grows up, the exploration component goes high, but it never converges to a definite value.

Absolute Error Objective Function			
		Closed Loop	
Initial PID Gains		KP = 0.1, KI = 0.005, KD = 1	KP = 0.225, KI = 0.01, KD = 4.25
Engine Speed	Max	1001	921
	Min	586	707
	Mean	797	800
Throttle Angle	Max	13.67	20
	Min	7.05	6
	Mean	11.7	14
Final PID Gains		KP = 0.229, KI = 0.002, KD = 1.001	KP = 0.024 KI = 0.0096, KD = 4.2525
Metrics		1.00E+07	1.00E+06

Table 5: Engine Speed, Throttle Angle, Function Value and Final PID gains for

Absolute Error Objective Function

Norm Square Error Objective Function			
		Closed Loop	
Initial PID Gains		KP = 0.1, KI = 0.005, KD = 1	KP = 0.225, KI = 0.01, KD = 4.25
Engine Speed	Max	987	920
	Min	614	707
	Mean	795	800
Throttle Angle	Max	14	20
	Min	6.9	6
	Mean	11.7	11.7
Final PID Gains		KP = 0.06 KI = 0.0015, KD = 1	KP = 0.223 KI = 0.0077, KD = 4.2594
Metrics		1.00E+09	1.00E+07

Table 6: Engine Speed, Throttle Angle, Function Value and Final PID gains for Norm Square Objective Function

Time Variant Norm Square Error Objective Function			
		Closed Loop	
Initial PID Gains		KP = 0.1, KI = 0.005, KD = 1	KP = 0.225, KI = 0.01, KD = 4.25
Engine Speed	Max	1012	920
	Min	565	707
	Mean	795	800
Throttle Angle	Max	13.53	20
	Min	7.1	6
	Mean	11.7	11.7
Final PID Gains		KP = 0.0096 KI = 0.0014, KD = 1	KP = 0.2244 KI = 0.0060, KD = 4.2431
Metrics		1.00E+10	1.00E+08

Table 7: Engine Speed, Throttle Angle, Function Value and Final PID gains for Time Variant Norm Square Objective Function

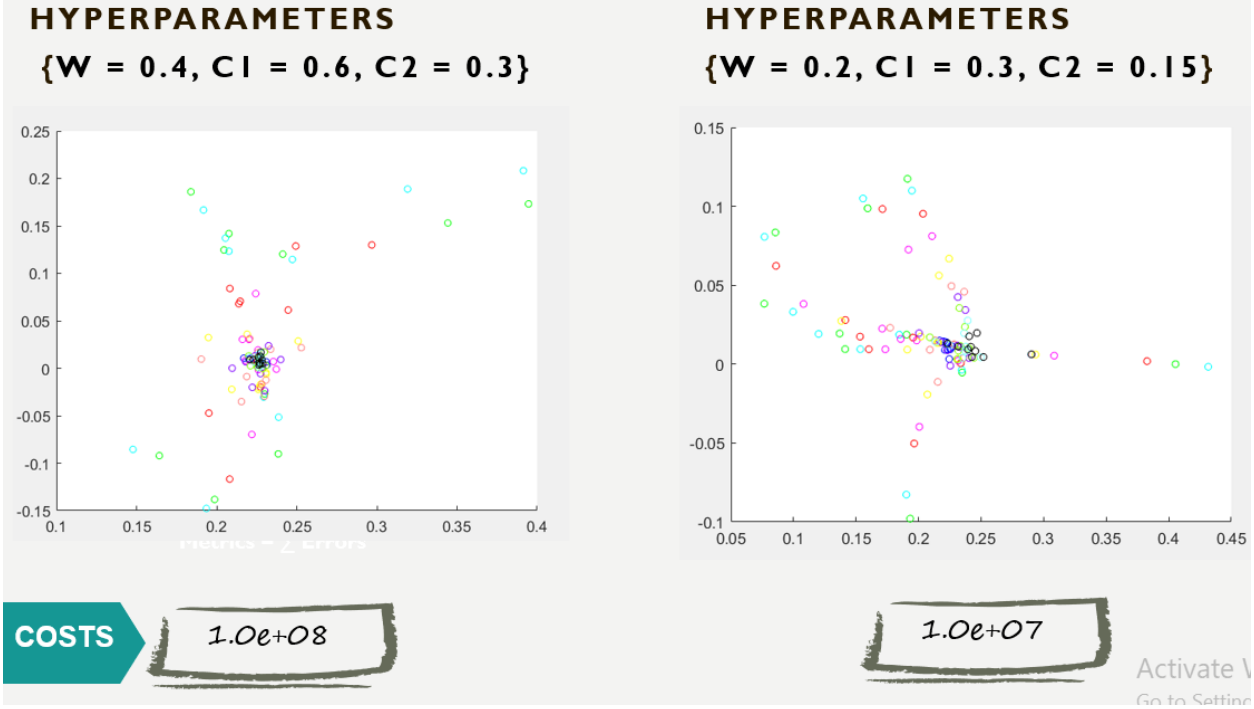


Figure 9: Impact of Hyperparameter Tuning on the POS Algorithm

There is a need to identify the best set of hyper parameters to find the best performance of the algorithm. We can use non-linear optimization for hyper parameter tuning as well however this part is not being done for this project submission.
Note: Stopping criterion for the POS algorithm is 10 Iterations in order to study the performance of the algorithm and understand system response.

7. Pattern Search Optimization – Introduction

The pattern search begins at the initial point x0 that you provide. For example, x0 = [1,1]. At the first iteration, the mesh size is 1 and the GPS algorithm adds the pattern vectors to the initial point x0 to compute the following mesh points:

[2.1], [1,2],[-1,1],[1,-1]

The algorithm computes the objective function at the mesh points in the order shown above.

The algorithm polls the mesh points by computing their objective function values until it finds one whose value is smaller than the objective function value on the initial point[1,1].

If the smaller objective function value exists,the poll at iteration 1 is successful. The algorithm sets the next point in the sequence equal to
point that has the smaller objective function.

At each iteration, the mash size will multiply by 2.

If there is no smaller objective function value than the current objective function value, than mash size will multiply by 0.5.

There is a lot stop criteria for this method. You can choose whatever criteria you want to stop the process.

8. Pattern Search Optimization – Results

Absolute Error Objective Function

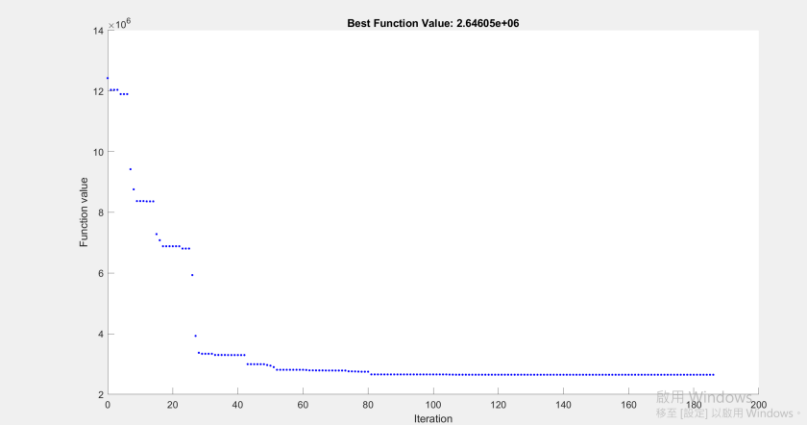


Figure 10: Function value with increasing Iteration (KP = 0.01, KI = 0.001, KD = 1)

Figure 11: Throttle Angle (KP = 0.01, KI = 0.001, KD = 1)

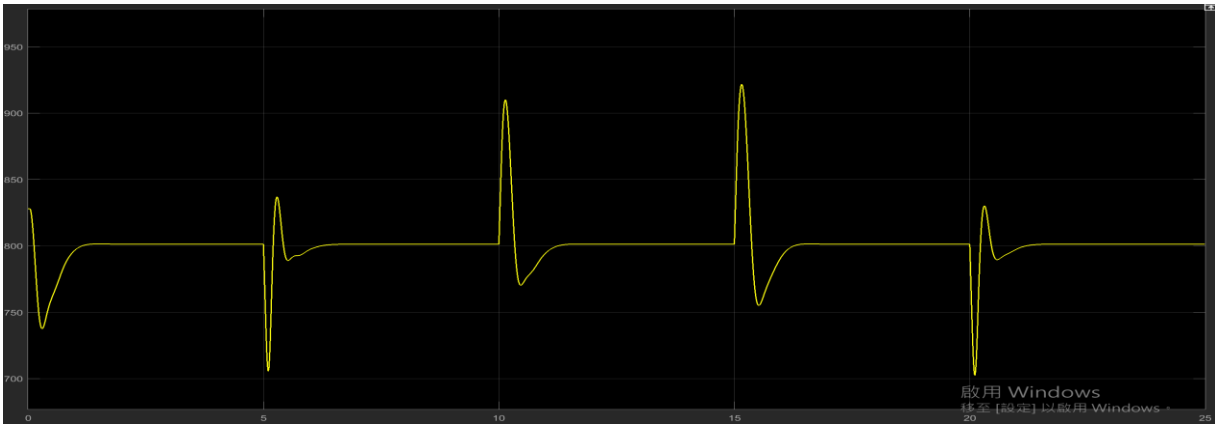


Figure 12: Engine Speed (KP = 0.01, KI = 0.001, KD = 1)

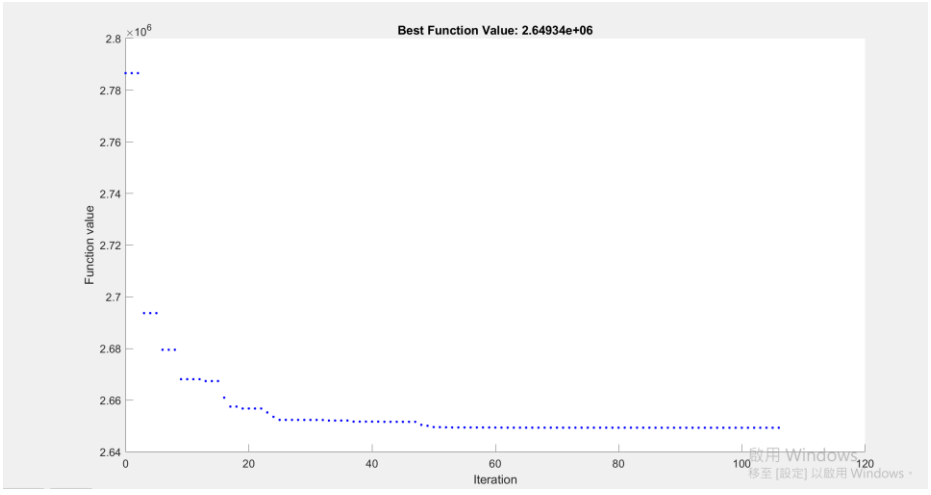


Figure 13: Function Value with Increasing Iteration (KP = 0.225, KI = 0.01, KD = 4.25)

Figure 14: Throttle Angle (KP = 0.225, KI = 0.01, KD = 4.25)

Figure 15: Engine Speed(KP = 0.225, KI = 0.01, KD = 4.25)

Absolute Error Objective Function			
		Closed Loop	
Initial PID Gains		KP = 0.01, KI = 0.001, KD = 1	KP = 0.225, KI = 0.01, KD = 4.25
Engine Speed	Max	921.8	921.9
	Min	702.3	702.6
	Mean	800.2	800.2
Throttle Angle	Max	19.36	19.5
	Min	6	6
	Mean	11.73	11.72
Final PID Gains		KP = 0.2396, KI = 0.0095, KD = 3.5972	KP = 0.2499 KI = 0.0105, KD = 3.6075
Iteration		186	106
Metrics		2.64605E+07	2.64934E+06

Table 8: Engine Speed, Throttle Angle, Function Value and Final PID gains for Absolute Error Objective Function

Norm Square Objective Function

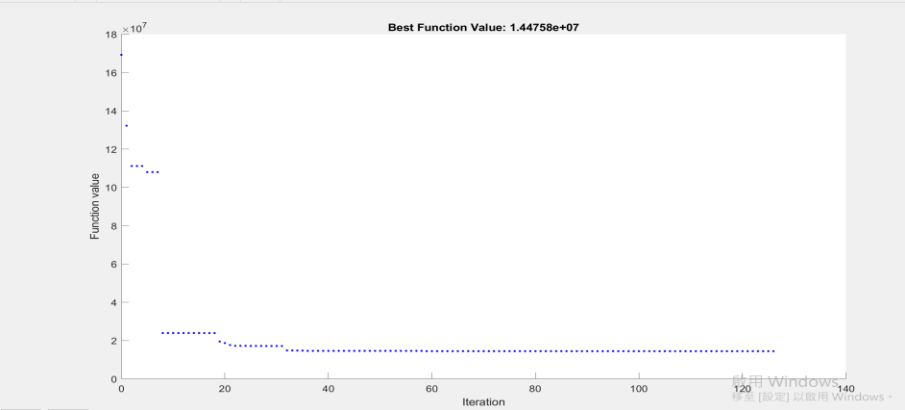


Figure 16: Function value with increasing Iteration (KP = 0.01, KI = 0.001, KD = 1)

Figure 17: Throttle Angle (KP = 0.01, KI = 0.001, KD = 1)

Figure 18: Engine Speed (KP = 0.01, KI = 0.001, KD = 1)

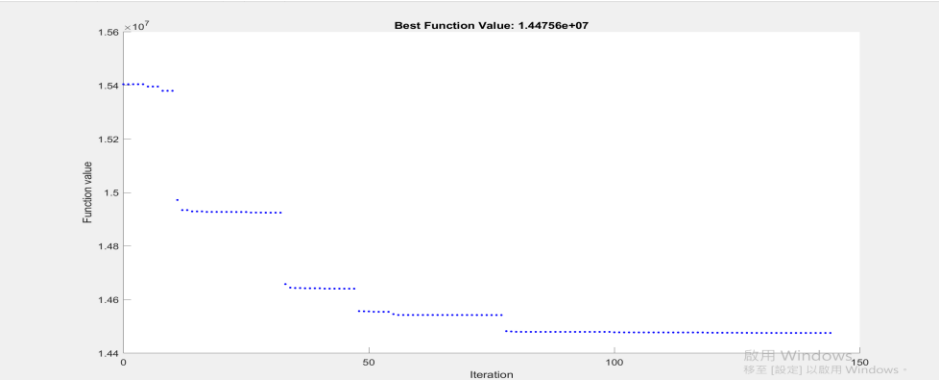


Figure 19: Function value with increasing Iteration (KP = 0.225, KI = 0.01, KD = 4.25)

Figure 20: Throttle Angle (KP = 0.225, KI = 0.01, KD = 4.25)

Figure 21: Engine Speed (KP = 0.225, KI = 0.01, KD = 4.25)

Norm Square Error Objective Function			
		Closed Loop	
Initial PID Gains		KP = 0.01, KI = 0.001, KD = 1	KP = 0.225, KI = 0.01, KD = 4.25
Engine Speed	Max	920.1	920.1
	Min	707.8	707.8
	Mean	800.3	800.3
Throttle Angle	Max	20	20
	Min	6	6
	Mean	11.7	11.7
Final PID Gains		KP = 0.2833 KI = 0.0076, KD = 4.1631	KP = 0.2827 KI = 0.0077, KD = 4.1627
Iteration		126	144
Metrics		1.44758E+07	1.44756E+07

Table 9: Engine Speed, Throttle Angle, Function Value and Final PID gains for Norm Square Objective Function

Time Variant Norm Square Objective Function

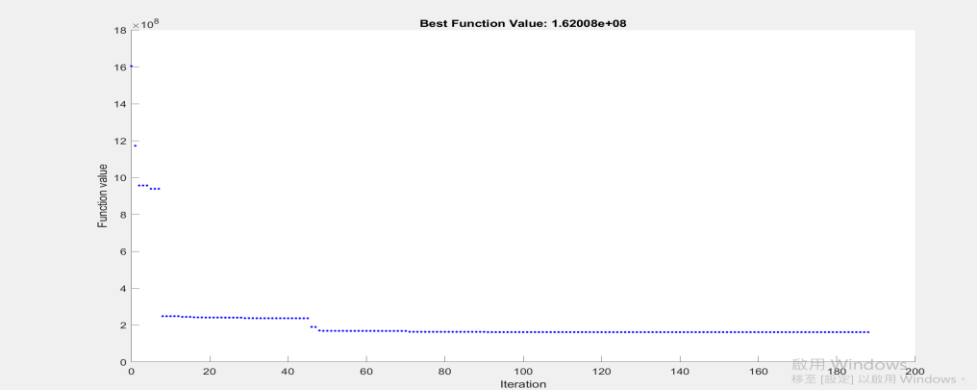


Figure 22: Function value with increasing Iteration (KP = 0.01, KI = 0.001, KD = 1)

Figure 23: Throttle Angle (KP = 0.01, KI = 0.001, KD = 1)

Figure 24: Engine Speed (KP = 0.01, KI = 0.001, KD = 1)

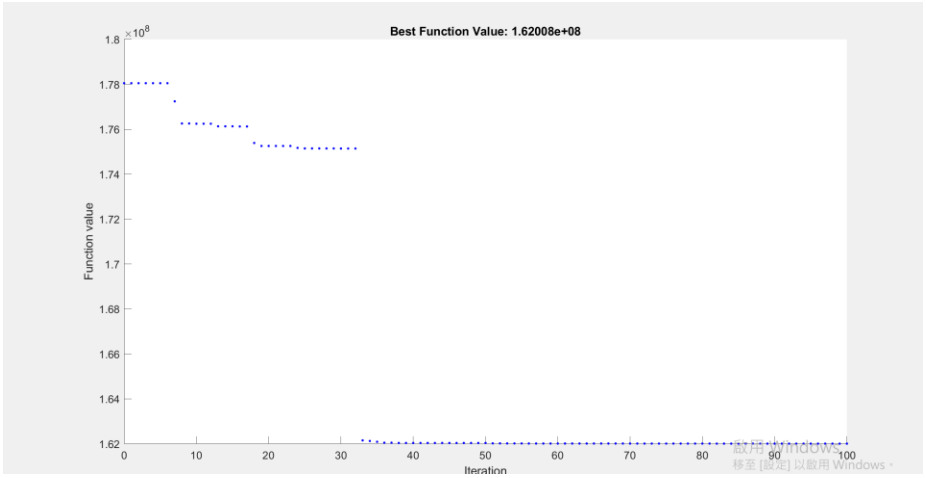


Figure 25: Function value with increasing (KP = 0.225, KI = 0.01, KD = 4.25)

Figure 26: Throttle Angle (KP = 0.225, KI = 0.01, KD = 4.25)

Figure 27: Engine Speed (KP = 0.225, KI = 0.01, KD = 4.25)

Time Variant Norm Square Error Objective Function			
		Closed Loop	
Initial PID Gains		KP = 0.01, KI = 0.001, KD = 1	KP = 0.225, KI = 0.01, KD = 4.25
Engine Speed	Max	920.1	920.1
	Min	706.6	706.6
	Mean	799.7	799.9
Throttle Angle	Max	20	20
	Min	6	6
	Mean	11.7	11.76
Final PID Gains		KP = 0.2628 KI = 0.0061, KD = 4.0605	KP = 0.2636 KI = 0.0061, KD = 4.0579
Iteration		188	100
Metrics		1.62008E+08	1.62008E+08

Table 10: Engine Speed, Throttle Angle, Function Value and Final PID gains for Time Variant Norm Square Objective Function

Explanation -
Although using the pattern search method will take a lot of time on running the simulation, it will get the good result for us no matter what the initial gains are. From the results of pattern search, we can see that the final gains are almost the same for the same objective function on different initial gains. In the hand-tuned part, we can find out that the gains (KP=0.225, KI=0.01, Kd=4.225) is the best. So, we use this as initial gains and compare results using another initial gains on the pattern search. If the initial value is better, the number of iteration have high possibility to be less than the initial value which is not good. Besides, the function values will be less as well. Then we compare the result using different objective functions. According to the results, using the absolute error has less function value than norm-square error and time-square error. However, for the excursion range of engine speed, norm-square is than the time-square and absolute error.

9. Conclusion

The result of the PSO algorithm resulted in a 7% reduction in cost compared to the hand-tuned PID controller initial gains and FMINCON optimization only after 10 iterations. However, the optimized PID controller had a similar effect on engine perturbation response of just 920 RPM as maximum and 710 RPM as minimum engine speed at idle. These results are far better than the Pattern Search Optimization for similar gains.

Moreover, there is a requirement to optimize the hyper-parameters of the POS algorithm which further requires deploying a non-linear optimization technique which we have kept out of scope for this study.

Appendix: MATLAB Code

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Intialize PID gain
%KP = 1;
%TI = 1;
%KI = KP/TI;
%Td = 1;
%Kd = KP*Td;

simOut = sim('idle_speed_model_project_pid');

A = [];
B = [];
x0 = [0.01, 0.001, 1];
Aeq = [];
beq = [];
lb = [0, 0, 0];
ub = [5, 5, 5];
nonclon = [];
y = fmincon(@myFunc,x0, A, B, Aeq, beq, lb, ub, nonclon)


function F = myFunc(x)

    assignin('base','KP', x(1))
    assignin('base','KI', x(2))
    assignin('base','Kd', x(3))

    simout = sim('idle_speed_model_project_pid');
    %E = integral(@(t)(abs(error.signals.values)),0,numel(error.time),'ArrayValued',true);
    %E = integral(@(t)(abs(norm(error.signals.values))^2),0,numel(error.time),'ArrayValued',true);
    E = integral(@(t)power(error.signals.values,2).*(error.time),0,numel(error.time),'ArrayValued',true);
     F = sum(E);

end
```

FMINCON Optimization

```
% Initialize particle best vector

P = Ks;

% Initialize cost vector

costs = rand(1,d);

% Initialize Velocities

v = rand(3,d) - rand(3,d);

% Perform Initial cost calculations

for x = 1:d

        KP = Ks(1,x);

        KI = Ks(2,x);

        Kd = Ks(3,x);

        % Run Initial Simulation

        simOut = sim('idle_speed_model_project_pid');

        % store cost in G array

        E = integral(@(t)power(error.signals.values,2).*(error.time),0,numel(error.time),'ArrayValued',true);
        costs(x) = sum(E);
end

particle_bests = costs;

[global_best,i] = min(costs);

G = Ks(:,i);
```

```
COSTS = [global_best];

t=0;

cindex = 2;

while t < 10

    %randomize r ans s vectors

    r = rand(3,d);

    s = rand(3,d);

    G_ = rand(3,d);

    for x = 1:d

        G_(:,x) = G;

    end

    % update velocities

    v = (w*v) + ((c1*r).*(P - Ks)) + ((c2*s).*(G_ - Ks));

    % update gains

    Ks = Ks + v;

    hold on

    scatter(Ks(1,:), Ks(2,:),20,colors(cindex,:))

    cindex = cindex + 1;

    % initialize new cost array

    costs_new = rand(1,d);

    [m,i] = min(costs_new);

    if m < global_best

        global_best = m;

        G_new  = Ks(:,i);

    else

        G_new = G;

    end

    %diff = abs(m - G);

    G = G_new;

    t = t+1;

    COSTS = [COSTS global_best];

end

%plot(COSTS)


KP = G(1);

KI = G(2);

Kd = G(3);

simOut = sim('idle_speed_model_project_pid');

%}
```

Particle Swarm Optimization MATLAB Code

```
options = optimoptions('patternsearch','Display','iter','PlotFcn',@psplotbestf);
x0 = [0.225,0.01,4.25];
A = [];
b = [];
Aeq = [];
beq = [];
lb = [0,0,0];
ub = [5,5,5];
nonlcon = [];
y = patternsearch(@asd,x0,A,b,Aeq,beq,lb,ub,nonlcon,options)
function F =  asd(x)

assignin('base', 'KP',x(1));
assignin('base', 'KI',x(2));
assignin('base', 'Kd',x(3));
Simout =sim('idle_speed_model_project_pid');
%E = integral(@(t)(abs(error.signals.values)),0,numel(error.time),'ArrayValued',true);
%E = integral(@(t)((norm(abs(error.signals.values)))^2),0,numel(error.time),'ArrayValued',true);
E = integral(@(t)power(error.signals.values,2).*(error.time),0,numel(error.time),'ArrayValued',true);
F = sum(E);
end
```


Pattern Search Optimization MATLAB Code