Control System Design Applied to Idle Stabilization of a Spark Ignition Engine

Jeremiah Mahler
jmahler@mail.csuchico.edu

California State University Chico
December 21, 2013

Abstract

The task of maintaining a stable idle for an internal combustion engine with spark ignition is non-trivial. Any time an accessory is turned on/off the torque applied to the engine changes. And changes in torque will change the engine rpm if the control inputs are constant. This paper investigates how control system designs can be applied to the problem of idle stabilization. Methods include: pole placement, direct design, and Control Law designs. Because the engine model is inherently discrete all of the methods discussed are also discrete.

Contents

1 Engine Model **3**

2 Direct Design **8**

3 Control Law, Intuitive Design **11**

4 Control Law, Full Feedback, No Prediction **13**

5 Conclusion **15**

A Steady State to Delta Transform Derivation **16**
A.1 Matlab Source 19

B Delta to Steady State Transform Derivation **21**
B.1 Matlab Source 23

C Control Response **24**

D Engine Model Matlab Source **26**

E General Matlab Functions **29**

F Direct Design Matlab Source **31**

G Control Law Matlab Source **34**

H Colophon **37**

1 Engine Model

The engine model used here is based work by Butts and Sivashankar[1] which was derived from the work by Powell and Cook.[2] The configuration is a modern 4.6L V-8 gas engine. To simplify this analysis the linearized model was used as shown in Figure 1.

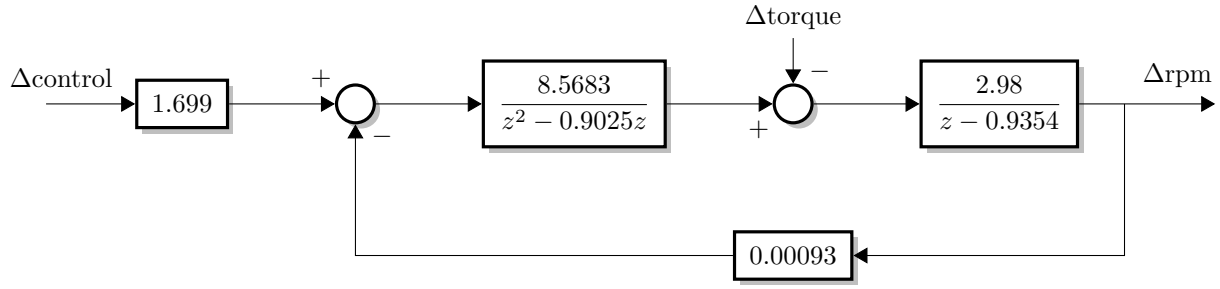

Figure 1: Linear engine model of a modern 4.6L V-8.

This model takes two inputs: a torque, and a idle control signal. When the torque is greater than zero it will oppose the rotation of the engine causing it to slow down. The idle control signal is some fraction of unity. This fraction corresponds to a pulse width modulated idle control valve which is at a minimum near zero and at a maximum near unity.

Because all the inputs and outputs are defined as deltas (Δ) this model cannot be used directly with typical control systems which expect steady state values. It is possible to convert these deltas to steady state equivalents. Figure 2 shows the transfer function to convert steady state values to delta values [3]. Figure 3 shows the transfer function to delta values to steady state values [4].

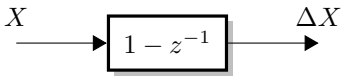

Figure 2: The Z transform used to accumulate the input and convert a steady state input to delta output.

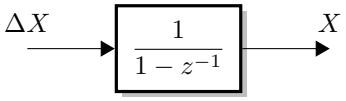

Figure 3: The Z transform used to convert delta input to a steady state output.

Typical control systems have an associated time step. And the choice of this time step is crucial in determining performance with regard to the Nyquist frequency. However this model does not suffer from this issue because it is inherently discrete. A single ignition event of the engine corresponds to a single step of the model.

[1] K. Butts, N. Sivashankar, and J. Sun. "Feedforward and feedback design for engine idle speed control using l1 optimization". In: *American Control Conference, Proceedings of the 1995*. Vol. 4. 1995, 2587–2590 vol.4. DOI: 10.1109/ACC.1995.532315.

[2] B.K. Powell and J. A. Cook. "Nonlinear Low Frequency Phenomenological Engine Modeling and Analysis". In: *American Control Conference, 1987*. 1987, pp. 332–340.

[3] The derivation of the steady state to delta conversion is given in Appendix A

[4] The derivation of the delta to steady state conversion is given in Appendix B.

In order to construct a controller the model (Figure 1) needs to be simplified in to a transfer function. However the prescence of two inputs, control and torque, complicates matters. To resolve this issue the torque can be set zero. Then it can be simplified by recognizing that it matches the well known form shown in Figure 4 which has the transfer function in Equation 1.

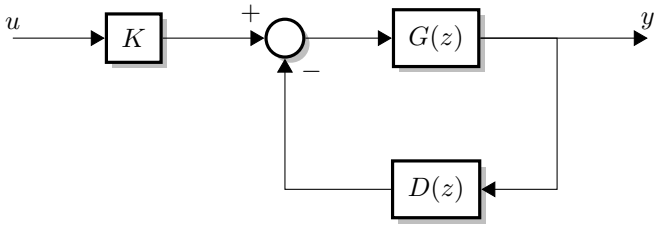

Figure 4: Direct Design system with K as a scaling input, G is the plant and D is the controller.

$$\frac{y}{u} = K\frac{B\alpha}{A\alpha + B\beta} \tag{1}$$

To simply first gather the parts from the engine model (Figure 1).

$$\begin{aligned}
\frac{B}{A} &= \frac{8.5683}{(z^2 - 0.9025z)}\frac{2.98}{(z - 0.9354)} \\
K &= 1.699 \\
\frac{\beta}{\alpha} &= 0.00093
\end{aligned}$$

Then substitute them in to Equation 1 and simplify. The result is Equation 2.

$$\begin{aligned}
\frac{y}{u} &= \frac{(1.699)(8.5683)(2.98)(1)}{(z^2 - 0.9025z)(z - 0.9354)(1) + (8.5683)(2.98)(0.00093)} \\
\boxed{\frac{y}{u} &= \frac{43.38}{z^3 - 1.838z^2 + 0.8442z + 0.02375}}
\end{aligned} \tag{2}$$

It is still necessary to address the delta inputs and outputs. This can be resolved by placing the conversion transforms (from Figure 2 and 3) on either end of the engine model as shown in Figure 5. A beneficial side effect becomes apparent in this form. The transform that converts from steady state to a delta cancels with the transform that converts from a delta to steady state [5]. Therefore the final transform is still Equation 2. An example of the output response with no control is shown in Figure 6. The Matlab source code for this plot and others are in Appendix D.

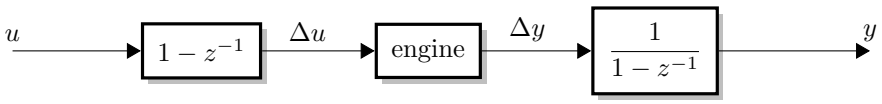

Figure 5: Engine model with transforms for converting from steady state to delta and vice versa.

[5] In the more general case, when torque is not zero, the transforms wouldn't cancel.

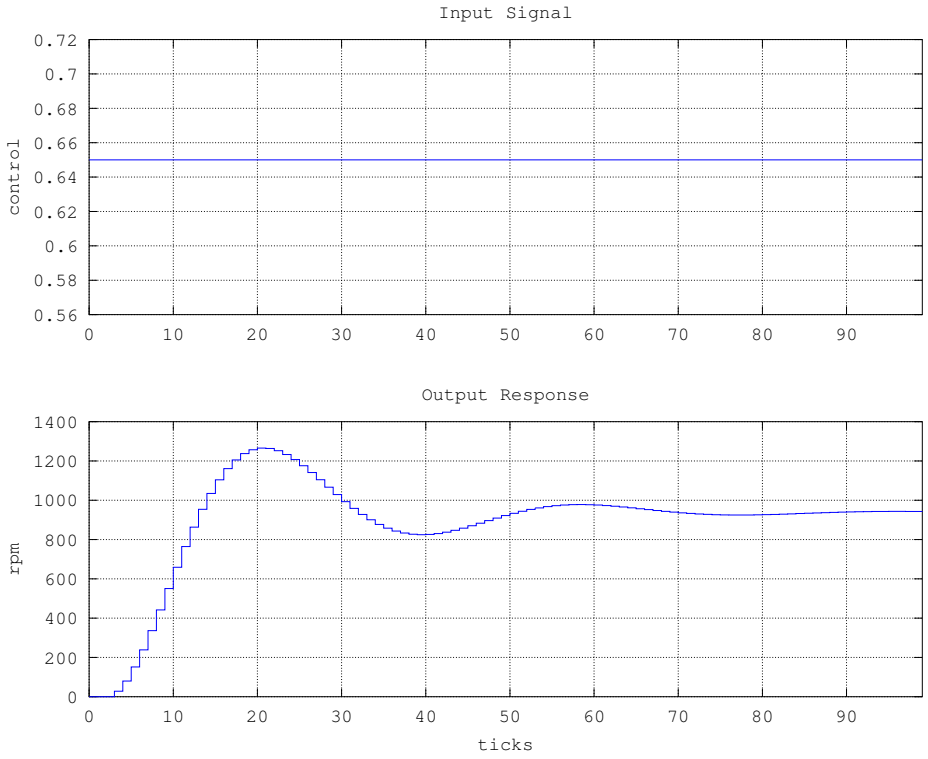

Figure 6: Output response of engine model with no control. The control can range from zero to one. Here the control is constant at 0.65 which resulted in a stable rpm near 900.

It is not apparent from Figure 6 exactly how long this process takes because the x axis represents ignition events instead of time. However the rpm can be converted to time per ignition event by examining the units as shown in Equation 3.

$$\frac{1}{x\frac{\text{rev}}{\text{min}} \cdot n\frac{\text{ign}}{\text{rev}} \cdot \frac{\text{min}}{60\text{sec}}} = y\frac{\text{sec}}{\text{ign}} \tag{3}$$

$$\begin{aligned}
\frac{\text{rev}}{\text{min}} &: \text{revolutions per minute (rpm)} \\
\frac{\text{ign}}{\text{rev}} &: \text{number ignition points per revolution (4 for 8 cylinder engine)} \\
\frac{\text{sec}}{\text{ign}} &: \text{seconds per ignition point}
\end{aligned}$$

Applying Equation 3 results in the response shown in Figure 7. It can be seen that the rise time is over 0.5 second and that the overshoot is close to 40%. The stabilization time is also several seconds. Clearly its performance could be improved with the application of a controller.

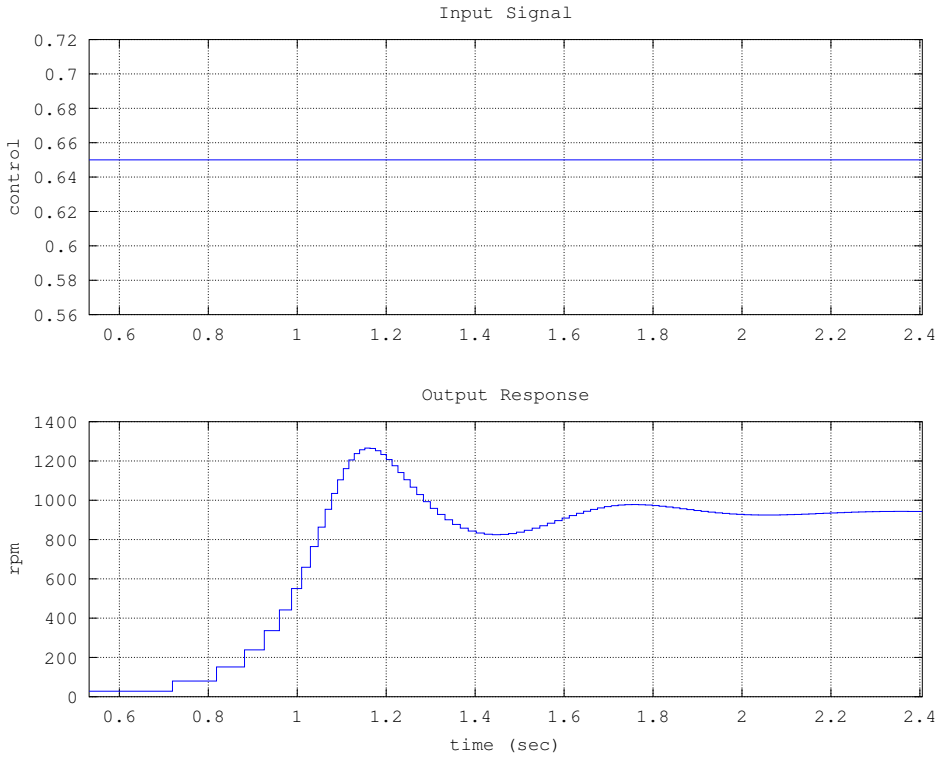

Figure 7: Engine rpm response compared to time with a constant 0.65 input control.

The response from zero rpm is not a realistic measurement of performance because zero rpm is impossibly slow. A typical idle rpm range is from 600 to 1000 rpm. Figure 8 shows the response for a step input with control inputs chosen to reach 600 and 1000 rpm [6]. This is a more realistic representation of typical operation. The rise time is near 0.2 seconds and the overshoot is 15%. And the stabilization time is almost one second. Its behavior can still be improved with a controller.

[6]The control input to stable rpm values are given in Appendix C

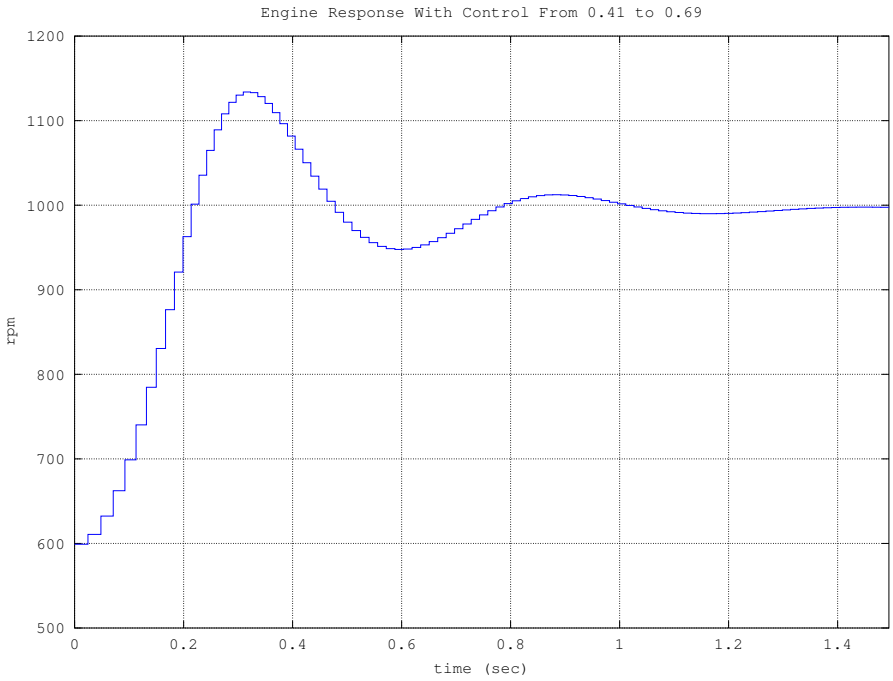

Figure 8: Response for a shifted and scaled step input with zero torque.

2 Direct Design

There are several characteristics of this Direct Design method. The controller is built entirely in the discrete domain. If the plant ($G(z)$) is continuous it is converted to discrete using a Zero Order Hold. In this case it is already discrete. It is a pole placement technique where the desired roots of the entire system are specified by the designer. The controller values for β and α are found using the Diophantine equation along with a Sylvester matrix. And the gain (K) is chosen such that the desired limit is reached. Figure 9 shows the structure of the system.

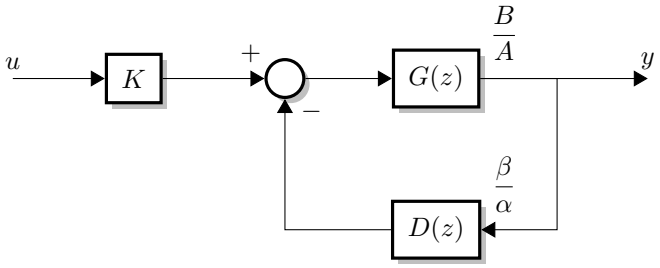

Figure 9: Direct Design system where K is the scaling input, G is the plant and D is the controller.

In this case the roots were not given and no deterministic way was available to find them. So trial and error was used.

$$\text{roots} : z = 0.6 \pm 0.4i$$

Since the plant is already in discrete form it is not necessary to use a Zero Order Hold. The engine model, repeated from Equation 2, is shown below.

$$\begin{aligned} G(z) &= \frac{43.38}{z^3 - 1.838z^2 + 0.8442z + 0.02375} \\ &= \frac{B(z)}{A(z)} = \frac{b_0 z^3 + b_1 z + b2}{a_0 z^3 + a_1 z + a2} \end{aligned}$$

Next it must be arranged to place it in the form of the Diophantine Equation.

From the block diagram the equation describing the system is found.

$$\begin{aligned} \frac{y}{h} &= H \\ H &= K\frac{G}{1 + DG} \\ H &= K\frac{B\alpha}{A\alpha + B\beta} \end{aligned}$$

From the roots D is constructed.

$$\begin{aligned} D(z) &= (0.6 + 0.4i)(0.6 - 0.4i) \\ D(z) &= z^2 - 1.2z + 0.520 \end{aligned}$$

Recall the form of the Diophantine Equation where the order is n.

$$\begin{array}{cccccccc} \alpha(z) & A(z) & + & \beta(z) & B(z) & = & D \\ (n-1) & (n) & & (n-1) & (n) & & (2n-1) \end{array}$$

Since $A(z)$ is a third order polynomial $n = 3$. This requires that $D(z)$ be a fifth order polynomial $(2n-1)$. The order is increased by adding zeros.

$$\begin{aligned} D(z) &= z^5 - 1.2z^4 + 0.520z^3 \\ &= d_0z^5 + d_1z^4 + d_2z^3 + d_3z^2 + d_4z + d_5 \end{aligned}$$

This also requires that α and β are second order polynomials $(n-1)$.

$$\frac{\beta}{\alpha} = \frac{\beta_0 z^2 + \beta_1 z + \beta_2}{\alpha_0 z^2 + \alpha_1 z + \alpha_2}$$

To solve the Diophantine Equation

$$\begin{aligned} M &= E^{-1}D \\ \begin{bmatrix} \alpha \\ \beta \end{bmatrix} &= E^{-1}D \\ \begin{bmatrix} \alpha \\ \beta \end{bmatrix} &= \begin{bmatrix} \alpha_2 \\ \alpha_1 \\ \alpha_0 \\ \beta_2 \\ \beta_1 \\ \beta_0 \end{bmatrix} \end{aligned}$$

a third order Sylvester Matrix is required.

$$E = \begin{bmatrix} a_3 & 0 & 0 & b_3 & 0 & 0 \\ a_2 & a_3 & 0 & b_2 & b_3 & 0 \\ a_1 & a_2 & a_3 & b_1 & b_2 & b_3 \\ a_0 & a_1 & a_2 & b_0 & b_1 & b_2 \\ 0 & a_0 & a_1 & 0 & b_0 & b_1 \\ 0 & 0 & a_0 & 0 & 0 & b_0 \end{bmatrix}$$

Combining these parts and solving for α and β results in a solution for $D(z)$.

$$\boxed{\frac{\beta}{\alpha} = D(z) = \frac{0.02297z^2 - 0.01686z - 0.0004643}{z^2 + 0.6379z + 0.8482}}$$

The gain (K) still needs to be found and this can be accomplished using the limit.

$$\lim_{z\to 1} K\frac{B\alpha}{A\alpha + B\beta} = 1$$
$$\lim_{z\to 1} K\frac{(43.38)(z^2 + 0.6379z + 0.8482)}{(z^5 - 1.2z^4 + 0.520z^3)} = 1$$
$$K = \frac{(1 - 1.2 + 0.520)}{(43.38)(1 + 0.6379 + 0.8482)}$$
$$\boxed{K = 0.0029671}$$

This solution results in the response shown in Figure 10. It is a dramatic improvement compared to no control. The rise time, percent overshoot, and settling time are all improved. This result is particularly interesting because the chosen roots were a guess. It is likely that further improvement could be made by optimizing the roots.

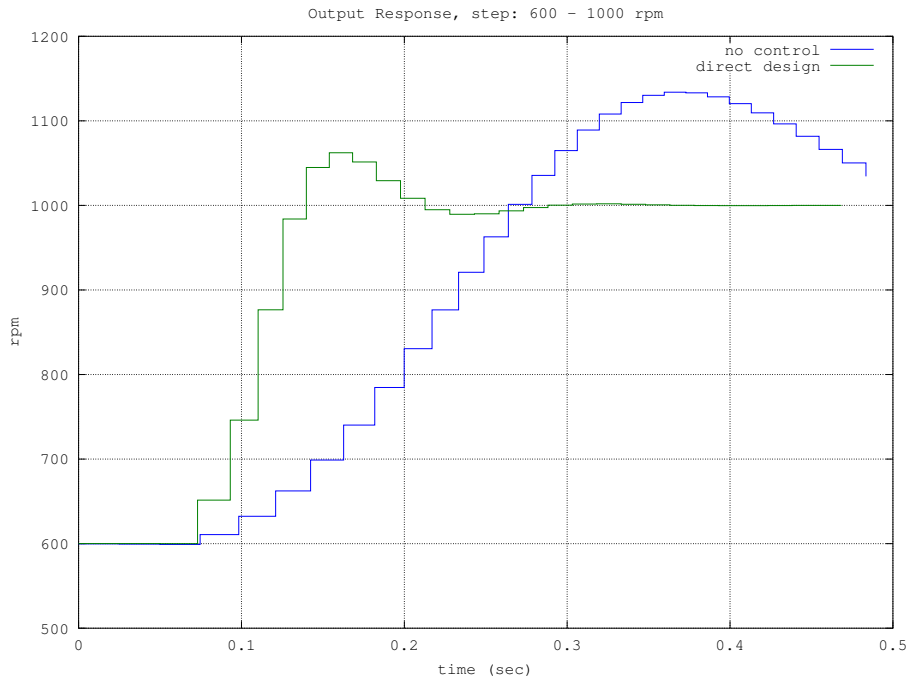

Figure 10: Output response of controller built using Direct Design compared to no control. For no control this is the same plot as in Figure 8 which stabilizes after a second. The Matlab source performing these calculations and producing this plot is given in Appendix F.

While this result is positive it is not apparent whether it is within the mechanical limits of a real engine. In particular the control input should be limited to a range from zero to one. Further investigation is needed to determine whether these limits are being exceeded. There is no doubt, however, that the addition of a controller improved performance. If the limits are being exceeded a different set of roots would have to be chosen which may be less optimal.

3 Control Law, Intuitive Design

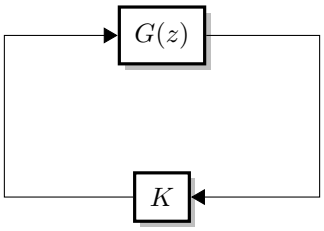

Figure 11: Control Law regulator system. Typically, $-K$ is used to create negative feedback.

A Control Law Design[7] configured as a regulator is shown in Figure 11. Notice that there is no input which means it acts as a regulator as opposed to a servo.

Several requirements must be satisfied in order for this system to be stable. First, the input to $G(z)$ must reflect the output without any offset. Second, the system must stabilize at zero. These requirements can be verified intuitively. Suppose for an input of 1 to $G(z)$ an output of 5 is produced. What value of K will produce a stable value?

$$\begin{aligned} 5K &= 1 \\ K &= \frac{1}{5} \end{aligned}$$

Now suppose a disturbance is introduced causing the output to increase to 6, will the system stabilize?

$$\begin{aligned} 6 \cdot K &= y \\ 6 \cdot \frac{1}{5} &= \frac{6}{5} \\ \frac{6}{5} &> 1 \quad \text{(diverge)} \end{aligned}$$

An increase in the output creates positive feedback which increases the input causing a further increase in the output. The system diverges. This exercise can be repeated with a controller at zero with no input/output offset. The result is a stable negative feedback system.

The linear engine model with zero torque (Equation 2) satisfies neither of these requirements. There is an offset because the input has a range from zero to one which corresponds to a range of 200 to 1200 on the output. And operation at zero rpm is impossible.

To resolve these issues compensation can be added to the system as shown in Figure 12. An input of zero will produce the desired rpm. The rpm is then converted to a control value (R). And the reached rpm (output of $G(z)$) is subtracted from the goal rpm to produce a zero output.

Figure 13 shows the response of this system with an arbitrary value of K. Values less than or equal to one produced a stable response and anything larger will cause it to diverge. The response is mediocre but the design was trivial and required no calculations. Later designs will improve this system by constructing more complex values for K.

[7] G.F. Franklin, J.D. Powell, and M.L. Workman. *Digital Control of Dynamic Systems*. Addison-Wesley world student series. Addison-Wesley Longman, Incorporated, 1998. ISBN: 9780201331530, Pg. 280.

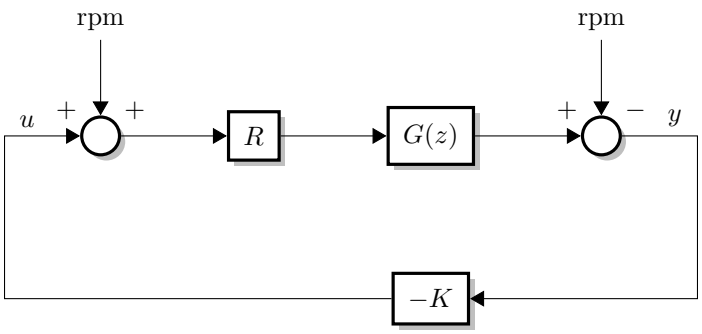

Figure 12: Compensations added to the linear engine model with zero torque ($G(z)$) to construct a regulator system. The transfer function R converts rpm input to control output (0 - 1). The rpm input is the desired idle speed (e.g. 700). Notice that both the output (y) and input (u) will be zero when the desired rpm is reached.

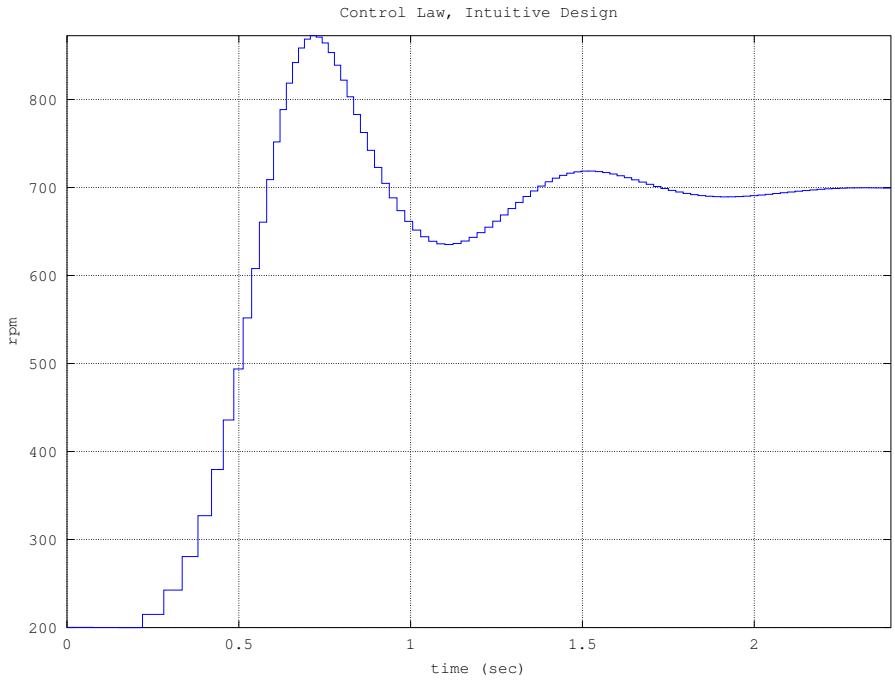

Figure 13: Output response of Control Law regulator with $K = 0.01$ and rpm = 700. Rpm was allowed to stabilize at 200 rpm before applying a step to 700 rpm. Source code is provided in Appendix G.

4 Control Law, Full Feedback, No Prediction

The simplest Control Law[8] design uses full feedback with no prediction. Ackermann's Formula is used to find K for given values of Φ and Γ. Figure 14 shows the system.

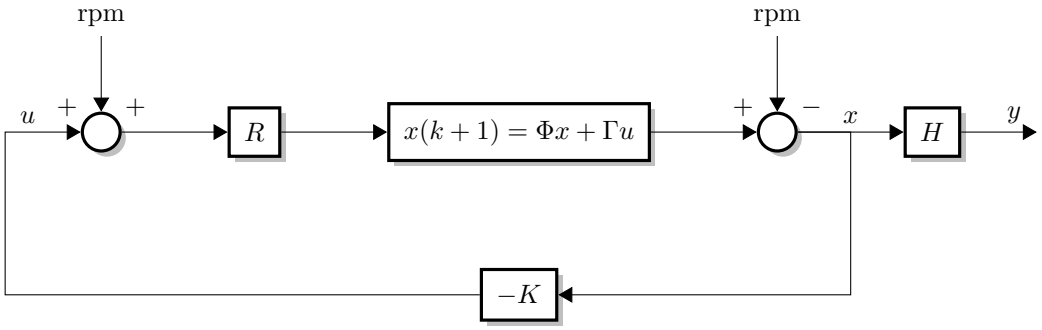

Figure 14: Control Law system for full feedback with no prediction. The system is in the discrete state space form, with Φ and Γ, to allow Ackermann's formula to be used to find K.

Unfortunately the compensations that were added to make it possible to build a Control Law system (Section 3) create complications when applying Ackermann's Formula to find K. A system commonly given in the literature is shown in Figure 15 [9]. Here the scope of Φ and Γ is clear and it is straight forward to apply Ackermann's Formula to find K. If the compensations in Figure 14 could be reduced to an equivalent transfer function with rpm set to a constant value Ackermann's could be equivalently applied. However the sum blocks cannot be reduced unless the constant input is zero, which is not the case here[10].

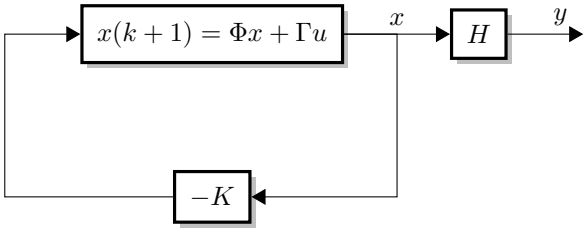

Figure 15: Typical Full Feedback system.

One possible solution is to rearrange the system as shown in Figure 16. This resolves one of the sum blocks but another is still present. If it is assumed that this sum block can be ignored the following values were found[11]. The resulting response is shown in Figure 17.

$$\begin{aligned} \text{roots}: \quad & (0.09 \pm 0.05i) \\ & 0.01 \\ K = [&-6.4313\text{E-}03 \quad 8.6624\text{E-}03 \quad -3.1446\text{E-}04] \end{aligned}$$

It is clear from the response that the sum block plays an important role with regard to the calculation of K using Ackermann's Formula. However it has not been determined how this problem can be resolved to find a valid value.

[8] Ibid., Pg. 280.
[9] Some diagrams omit H but pass x through an identity matrix.
[10] Zero rpm is impossible, 700 rpm is a typical idle speed.
[11] Matlab source is given in Listing 16 of Appendix G

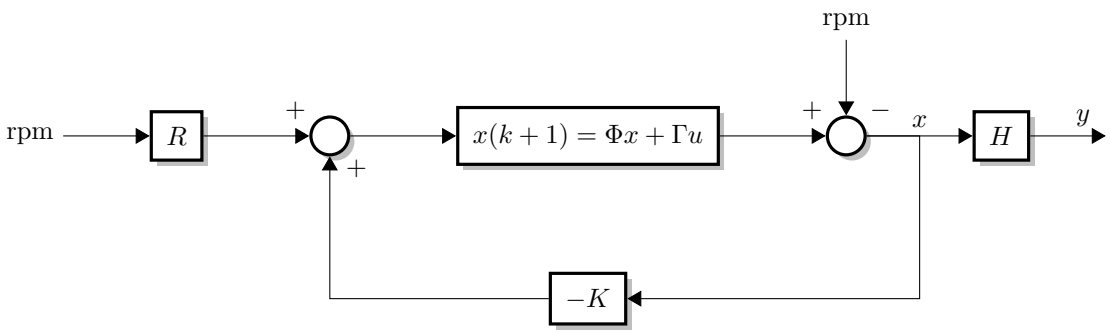

Figure 16: Rearranged Control Law system to exclude one sum block.

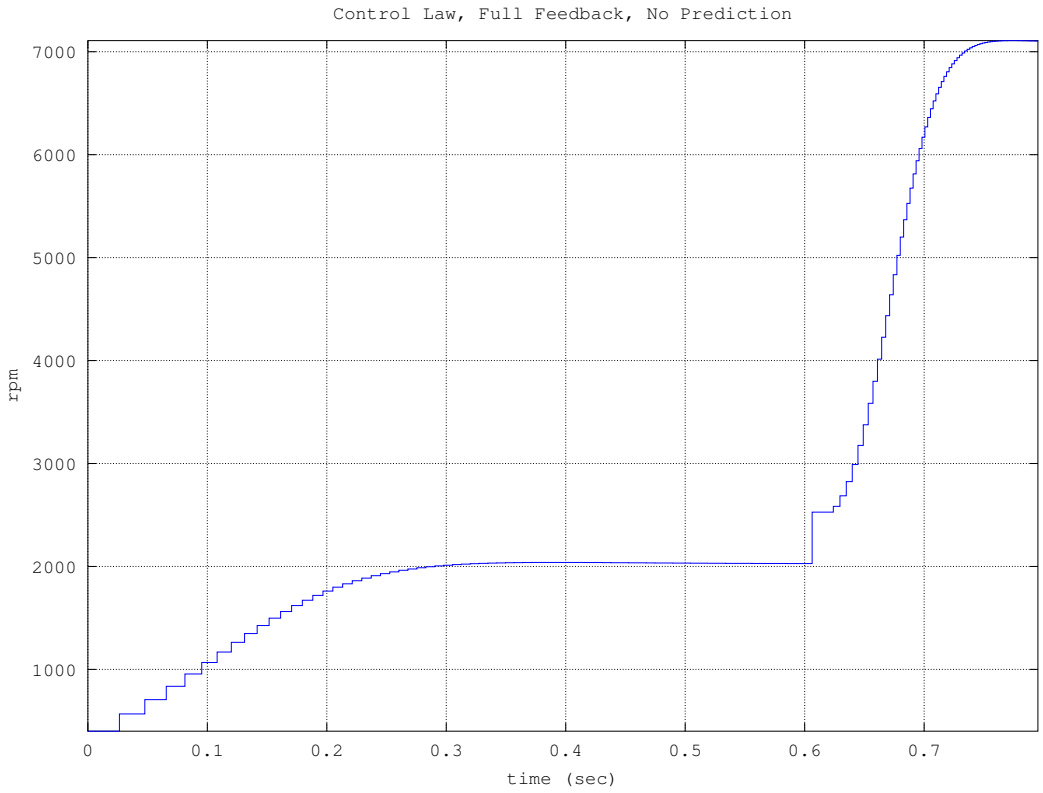

Figure 17: Output response if sum block is ignored and K is found. Rpm input is 200 until 0.6 sec where it steps to 700. The output should reflect the input which this clearly does not. Matlab source is given in Listing 16 of Appendix G.

5 Conclusion

The goal of this paper was to apply a range of control system designs to the problem of idle stabilization with a spark ignition engine. Unfortunately, numerous difficulties were encountered with this real world problem.

The engine model took a torque input to a sum block. To simplify calculations it was assumed to be zero which allowed the simplification of the system in to a single transfer function. A non-zero value would have substantially complicated matters.

The engine model did not accept steady state inputs. Transfer functions had to be found for converting a steady state to a delta and from a delta to a steady state.

The engine model cannot operate at zero rpm because this is a nonsensical value. Similarly a unit step input could not be used because both zero and one rpm are equally nonsensical. The scheme that was devised was to apply a custom input that started at 200 rpm and then stepped to 700 rpm. Then the resulting response could be seen under a typical range.

Several designs were successfully applied. A direct design using pole placement was constructed that had improved performance compared to no control And a Control Law design built using an intuitive design was a success. A Control Law with Full Feedback was attempted but was not a success. The complications created by a sum block prevented a correct K from being found using Ackermann's Formula. No designs beyond this were attempted because they would be equally hampered by the same problems.

A Steady State to Delta Transform Derivation

To accumulate a steady state input to produce a delta output a system can be constructed as shown in Figure 18. Its operation can be confirmed by trying some values. If all values are zero and then a 1 is input on u the output will become 1. On the next time step 1 will be output on v. Since q is zero r will be 1. If the input (u) remains 1 this will be subtracted from r to produce zero on the output (y).

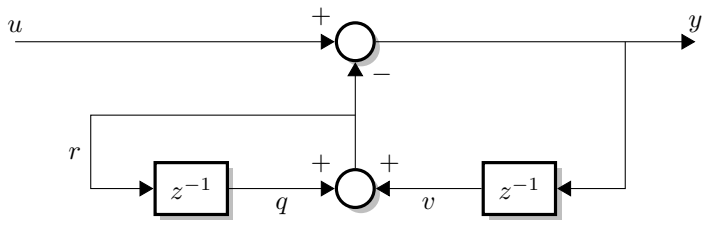

Figure 18: System to accumulate values to convert a steady state input to a delta output.

Figure 19 shows the response of this system given an arbitrary input. It can be seen that if the input is held constant the output (delta) returns to zero as expected.

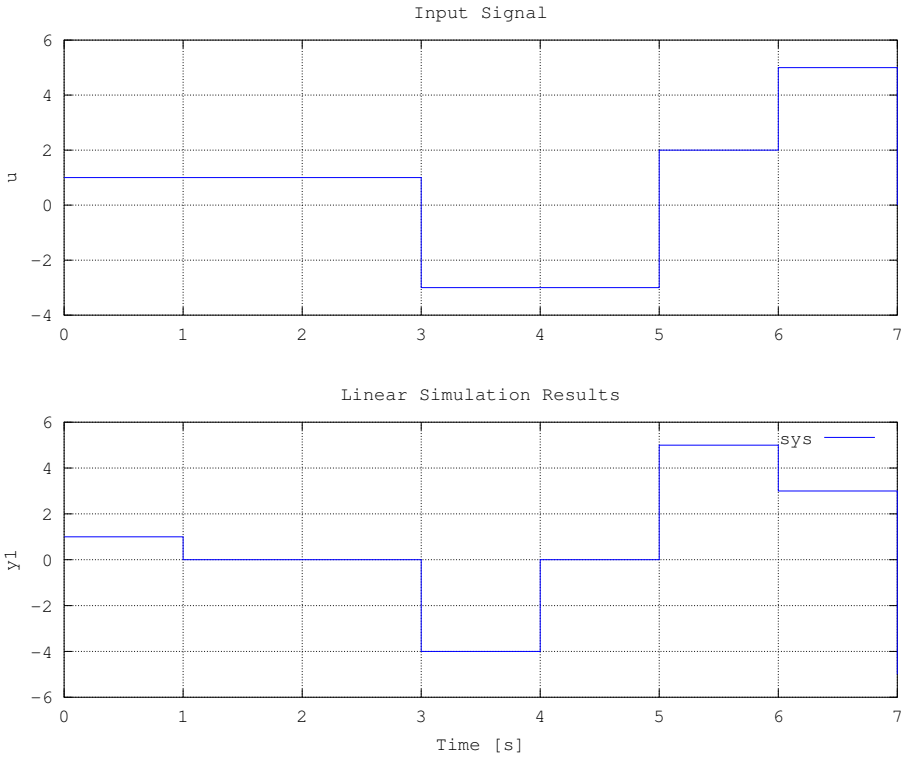

Figure 19: Response of full steady state input to delta output system to an arbitrary input signal. The upper plot is the input signal (u) and the lower plot is the output response (y). The Matlab source code is given in Listing 1.

However this full system can be simplified to a single transfer function. Starting from the equations that define the system

$$r = q + v \tag{4}$$
$$v = y \cdot z^{-1} \tag{5}$$
$$q = r \cdot z^{-1} \tag{6}$$
$$y = u - r \tag{7}$$

these can be algebraically manipulated to find the effective transfer function of the entire system (y/u).

$$\begin{aligned} r &= rz^{-1} + yz^{-1} && (4,5,6) \\ r(1 - z^{-1}) &= yz^{-1} \\ r &= u - y && (7 \\ (u - y)(1 - z^{-1}) &= yz^{-1} \\ u - y - uz^{-1} + yz^{-1} &= yz^{-1} \\ u - y - uz^{-1} &= 0 \\ y &= u(1 - z^{-1}) \end{aligned}$$

$$\boxed{\frac{y}{u} = 1 - z^{-1}} \tag{8}$$

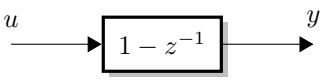

Figure 20: Simplified system to convert a steady state input in to a delta output.

It can be seen in Figure 21 that the simplified system behaves identically to the previous system (Figure 19).

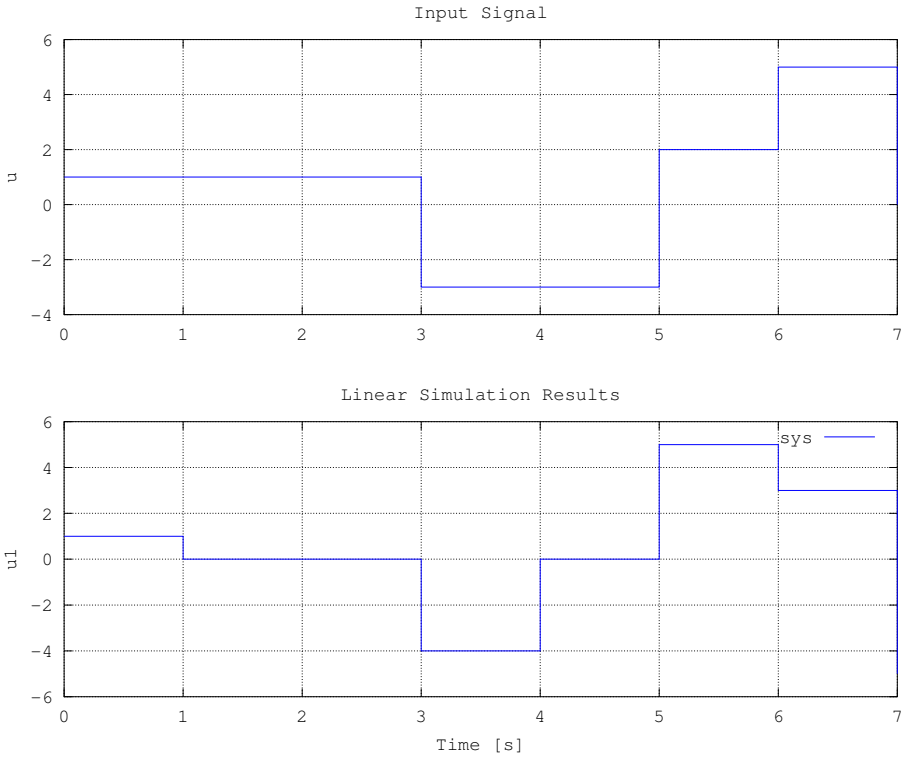

Figure 21: Response of simplified steady state input to delta output system to an arbitrary input signal. The upper plot is the input signal (u) and the lower plot is the output response (y). Response is identical to the full system in Figure 19 as expected. The Matlab source code is given in Listing 2.

A.1 Matlab Source

```
%
% cd_plot1.m
%

clear;

T = 1;  % time step

D1 = tf([1], [1 0], T, 'InputName', 'y1', 'OutputName', 'v1');
D2 = tf([1], [1 0], T, 'InputName', 'r1', 'OutputName', 'q1');
sum1 = sumblk('y1 = u1 - r1');
sum2 = sumblk('r1 = v1 + q1');
sys = connect(D1, D2, sum1, sum2, 'u1', 'y1');

u = [1 1 1 -3 -3 2 5 0];
t = 0:(size(u,2)-1);  % start at zero

figure;
subplot(2,1,1);
stairs(t,u);
grid on;
axis auto;
title('Input Signal');
ylabel('u');

subplot(2,1,2);
lsim(sys, u);
grid on;
axis auto;

print('cd_plot1.eps', '-depsc2');
```

Listing 1: Matlab code to plot the full steady state to delta system.

```
%
% cd_plot2.m
%

clear;

T = 1;  % time step

sys = tf([1 -1], [1 0], T, 'InputName', 'y1', 'OutputName', 'u1');

u = [1 1 1 -3 -3 2 5 0];
t = 0:(size(u,2)-1);  % start at zero

figure;
subplot(2,1,1);
stairs(t,u);
grid on;
axis auto;
title('Input Signal');
ylabel('u');

subplot(2,1,2);
lsim(sys, u);
grid on;
axis auto;

print('cd_plot2.eps', '-depsc2');
```

Listing 2: Matlab code to plot the simplified steady state to delta system.

B Delta to Steady State Transform Derivation

To convert a delta input to a steady state output it should sum the history of values. Figure 22 shows the system.

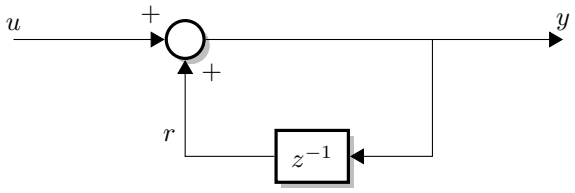

Figure 22: System to convert delta input to steady state output.

This system can be simplified in to a single transfer function as given by Equation 9 and shown in Figure 23.

$$\begin{aligned} y &= u + r \\ r &= y \cdot z^{-1} \\ y &= u + yz^{-1} \\ u &= y(1 - z^{-1}) \\ \boxed{\frac{y}{u} &= \frac{1}{1 - z^{-1}}} \end{aligned} \tag{9}$$

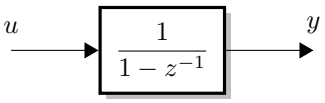

Figure 23: Simplified system to convert a delta input to a steady state output.

Figure 24 shows the response of this system given an arbitrary input [12]. It can be seen that the output is held in a steady state according to the delta inputs as expected.

[12] This input is actually the output of the steady state input to delta output given in Appendix A. They are equal and opposite as expected.

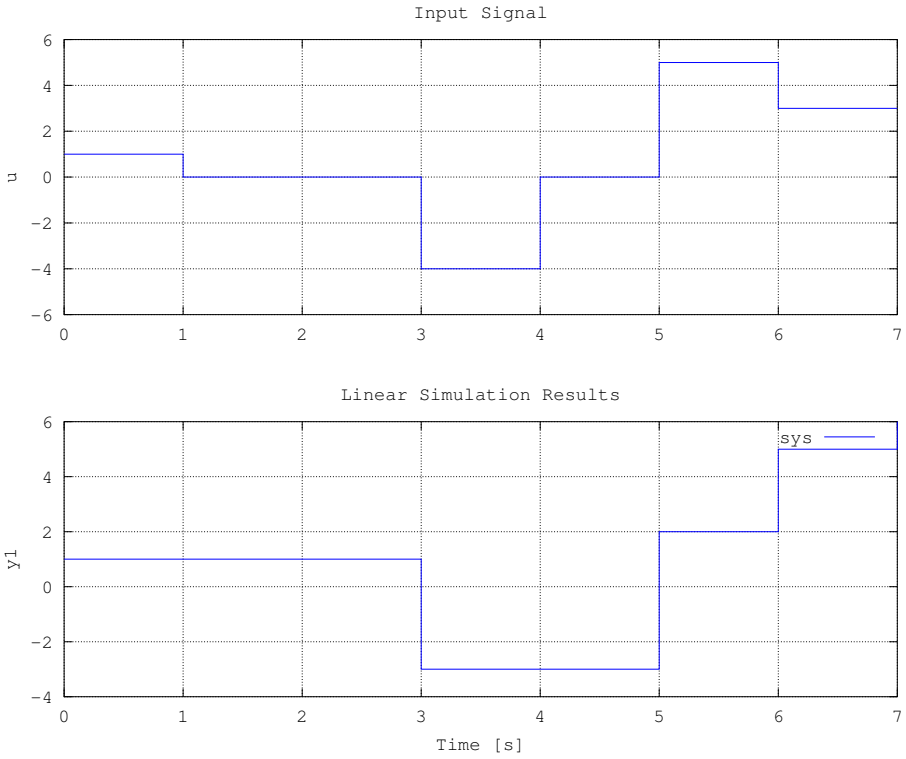

Figure 24: Response of delta input to steady state output system when given an arbitrary input signal. The upper plot is the input signal (u) and the lower plot is the output response (y). The Matlab source code is given in Listing 3.

B.1 Matlab Source

```
%
% dc_plot2.m
%

clear;

T = 1;  % time step

sys = tf([1 0], [1 -1], T, 'InputName', 'u1', 'OutputName', 'y1');

u = [1 0 0 -4 0 5 3 3];
t = 0:(size(u,2)-1);  % start at zero

figure;
subplot(2,1,1);
stairs(t,u);
grid on;
%axis auto;
axis([0 7 -6 6]);
title('Input Signal');
ylabel('u');

subplot(2,1,2);
lsim(sys, u);
grid on;
axis auto;
axis([0 7 -4 6]);

print('dc_plot2.eps', '-depsc2');
```

Listing 3: Matlab code to plot the simplified steady state to delta system.

C Control Response

For a given control input and zero torque the engine model will reach stable rpm values as shown in Figure 25. Equation 10 describes its behavior. Listing 4 shows how this was calculated.

$$y = 1450.1x \tag{10}$$
$$y : \text{rpm (200 - 1200)}$$
$$x : \text{control (0 - 1)}$$

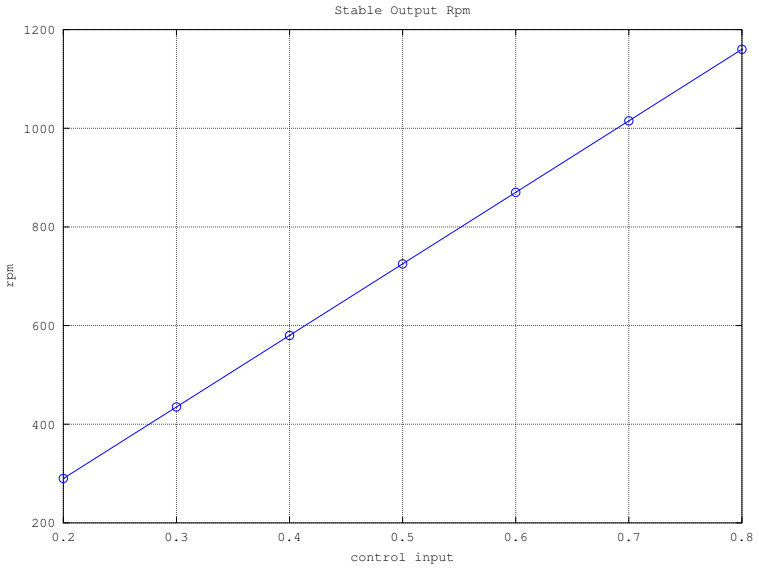

Figure 25: Stable rpm reached for a given constant control input.

```
%
% em_plot3.m
%
% Plot of stable output rpm's for a given control input.
%

clear;

T = 1;
Gz = engine_model(T);

% range of inputs to try
us = linspace(0.2, 0.8, 7);

y = zeros(0, length(us));
for i = 1:length(us)
        % response for a constant input
        u = [us(i)*ones(1,100)];
        [yp, t, x] = lsim(Gz, u);

        % save the stable value
        y(i) = yp(end);
end

figure(1);

plot(us, y, '-o');
grid on;
axis auto;
title('Stable Output Rpm');
xlabel('control input');
ylabel('rpm');

print('em_plot3.eps', '-depsc2');
```

Listing 4: Matlab source to find the stable rpm for a given control input.

D Engine Model Matlab Source

```
function [sys] = engine_model(T);
%  ENGINE_MODEL
%
% Linear engine model with zero torque.
%

%T = 1;  % time step

K = tf([1.699], [1], T);
D1= tf([8.5683], [1 -0.9025 0], T);
D2= tf([2.98], [1 -0.9354], T);
G = D1*D2;
D = tf([0.00093], [1], T);

H = K*G/(1 + D*G);

sys = minreal(H);  % cancel common roots
```

Listing 5: Matlab source to calculate the engine model system with zero torque. This function is called by other scripts that use this model.

```
%
% em_plot.m
%
% Plot of engine model with zero torque and no
% control.  x axis is in ticks (not time).
%

clear;

T = 1;  % time step
Gz = engine_model(T);

u = [0.65*ones(1,100)];
[y, t, x] = lsim(Gz, u);

figure;
subplot(2,1,1);

stairs(t, u);
grid on;
axis([t(1) t(end)]);
title('Input Signal');
ylabel('control');

subplot(2,1,2);
stairs(t, y);
grid on;
axis([t(1) t(end)]);
title('Output Response');
ylabel('rpm');
xlabel('ticks');

print('em_plot1.eps', '-depsc2');
```

Listing 6: Matlab source to plot the response of engine model with no control. The x axis is in ticks.

```
%
% em_plot2.m
%
% Engine model with no control and x axis in time.
%

clear;

ncyl = 8;

T = 1;  % time step
Gz = engine_model(T);

u = [0.65*ones(1,100)];
[rpm, t, x] = lsim(Gz, u);

% An rpm of zero makes the rpmtime huge and
% skews the results.
% Remove these initial zeros.
i = find(rpm~=0, 1, 'first');
rpm = rpm(i:end);
u = u(i:end);
t = t(i:end);

% Convert each rpm point to time instant.
% The cummulative sum is then the time it takes
% to get to that particular point.
rt = rpmtime(rpm, 8);

figure;
subplot(2,1,1);
stairs(rt, u);
grid on;
axis([rt(1), rt(end)]);
title('Input Signal');
ylabel('control');

subplot(2,1,2);
stairs(rt, rpm);
grid on;
axis([rt(1), rt(end)]);
title('Output Response');
ylabel('rpm');
xlabel('time (sec)');

print('em_plot2.eps', '-depsc2');
```

Listing 7: Matlab source to plot the response of engine model with no control. The x axis is in time.

```
%
% em_plot4.m
%
% Engine model with no control.
% Allowed to stablize before step is applied.
%

T = 1;

Gz = engine_model(T);

% input signal (control)
u_lo = 600/1450.1;
u_hi = 1000/1450.1;
n = 200;  % number of points
u = [u_lo*ones(1,n/2) u_hi*ones(1,n/2)];

% output response
[y, t, x,i] = lsim1(Gz, u);

rt = rpmtime(y, 8);

% remove data before the step
rt = rt((n/2):end);
y = y((n/2):end);

% reset start time to zero
rt = zerotime(rt);

figure(1);

stairs(rt, y);
grid on;
axis([rt(1), rt(end)]);
title('Engine Response With Control From 0.41 to 0.69');
xlabel('time (sec)');
ylabel('rpm');

print('em_plot4.eps', '-depsc2');
```

Listing 8: Matlab source to plot the response of engine model versus time for a typical 600 to 1000 rpm range.

E General Matlab Functions

This sections includes Matlab source for functions which are used among several methods.

```
function [E] = sylvester(A, B)
% SYLVESTER Construct a Sylvester matrix of the two vectors.
%
% Given two vectors, the largest order is determined.
% Then a Sylvester matrix is constructed for that order.
%

        nA = max(size(A));
        nB = max(size(B));

        n = max(nA, nB) - 1;  % order

        % First create a matrix of zeros
        E = zeros((n)*2, (n)*2);

        % Then assign specific values for A and B

        % A
        for (col = 1:n)
                for (i = 1:nA)
                        row = (nA - i) + col;
                        E(row, col) = A(i);
                end
        end

        % B
        for (col = (n+1):n*2)
                for (i = 1:nB)
                        row = (nB - i) + (col - n);
                        E(row, col) = B(i);
                end
        end
```

Listing 9: Matlab source of function to calculate a Sylvester Matrix.

```
function [ts] = zerotime(tx);
% ZEROTIME - Given a vector of incrementing times
%  recalculate the times so that it starts at zero.
%

ts = zeros(1,length(tx));
for (i = 2:length(tx))
        ts(i) = (tx(i) - tx(i-1)) + ts(i-1);
end
```

Listing 10: Matlab source of function to calculate zero time.

```
function [y, t, x, i] = lsim1(Hz, u, t, x0);
% LSIM1 - Just like lsim excpt small values are removed.
%
% "small" means less than or equal to 10.
%

switch nargin
    case 2
        t = [];
        x0=false;
    case 3
        x0=false;
end

[y, t, x] = lsim(Hz, u, t, x0);

% Find the first valid value
for i = 1:length(y)
        x = abs(y(i));
        if (x > 10)
                break;
        end
end

% adjust for removed values
y = y(i:end);
u = u(i:end);
t = t(i:end);

i = i - 1;
```

Listing 11: Matlab source of modified lsim() function.

```
function [ts] = rpmtime(rpm, ncyl)
% RPMTIME Calculate the cummulative time (in seconds) between
%  ignition events at a given rpm.
%
%  [ts] = rpmtime(rpm, ncyl);
%  [ts  = rpmtime(400, 8);
%

%ncyl;                              % number of cylinders
%rpm;                               % rpm
ign_rev = ncyl/2;          % igntion points per revolution
min_sec = 1/60;            % 1 minute per 60 seconds
%ts      ;                                  % seconds per ignition event

ts = zeros(0,length(rpm));  % initial empty vector
for i = 1:length(rpm)
        x = (rpm(i)*min_sec*ign_rev);

        if (0 == x)
                ts(i) = inf;
        else
                ts(i) = 1/x;
        end
end

ts = cumsum(ts);
```

Listing 12: Matlab source of rpmtime() function.

F Direct Design Matlab Source

Matlab source code used to build a controller using Direct Design and plot the output. The `engine_model` is given in Appendix D and other functions are given in Appendix E.

```
function [Hz, Dz, K] = direct_design(Gz, roots, order, limit=1);
% DIRECT_DESIGN - Construct a controller using Direct Design.
%
% 'Gz' is some plant to be controlled.
% 'order' is the order of the system need for the Sylvester Matrix.
% 'roots' are the designer provided roots to achieve in the system (Hz).
%
% The entire system is returned (Hz) along with the controller (Dz)
% and the constant (K) that were used.
%

% Designer provided specifications.
%
n = order;
D = poly(roots);
if (isrow(D))
        D = transpose(D);
end
% Reverse D so Alpha and Beta are in the correct order.
D = flipud(D);

[Bz, Az, T] = tfdata(Gz, 'v');

E = sylvester(Az, Bz);

%M = E^-1*D;
M = E\D;

% Alpha = a0*z + a1
% Beta = b0*z + b1
Alpha = fliplr(transpose(M(1:n)));
Beta = fliplr(transpose(M((n+1):end)));

Dz = tf(Beta, Alpha, T);

% To find K, the limit should go to 1
% for a step input.
% (refer to the notes for a better description)
K = limit*polyval(D, limit)/polyval(conv(Bz, Alpha), limit);

Hz = K*Gz/(1 + Dz*Gz);

endfunction
```

Listing 13: Matlab source of function is to build a controller using Direct Design.

```
%
% dd_plot2.m
%
% Plot of solution using direct design compared to no control.
%

clear;

T = 1;
ncyl = 8;

% input signal
u_lo = 600;              % rpm before step
u_hi = 1000;     % rpm after step
% different systems take longer to stabilize (time lo)
n1_lo = 30;              % nth tick to step from lo to hi
n2_lo = 100;
n_hi = 30;               % ticks after step
u1 = [u_lo*ones(1,n1_lo) u_hi*ones(1,n_hi)];
% convert to control values (without control)
u2 = [(u_lo/1450.1)*ones(1,n2_lo) (u_hi/1450.1)*ones(1,n_hi)];

% Build a controller, and the resulting system.
Gz1 = engine_model(T);
% These roots are guess, taken from some other example.
roots = [(0.6 + 0.4i) (0.6 - 0.4i) 0 0 0];
order = 3;
[Hz1, Dz1, K1] = direct_design(Gz1, roots, order);

% System with no control.
Gz2 = engine_model(T);

% output response
[y1, t1, x, i1] = lsim1(Hz1, u1);
[y2, t2, x, i2] = lsim1(Gz2, u2);  % no control
% adjust the step point for any removed from the beginning
i = max([i1 i2]);

% convert rpm to time
rt1 = rpmtime(y1, ncyl);
rt2 = rpmtime(y2, ncyl);

% remove data before step is applied
p1_start = (length(rt1) - n_hi) + 1;
rt1 = rt1(p1_start:end);
y1 = y1(p1_start:end);

p2_start = (length(rt2) - n_hi) + 1;
rt2 = rt2(p2_start:end);
y2 = y2(p2_start:end);

% reset times to start at zero
rt1 = zerotime(rt1);
rt2 = zerotime(rt2);

% plot

clf;
figure(1);

[xs1, ys1] = stairs(rt1, y1);
[xs2, ys2] = stairs(rt2, y2);
plot(xs2, ys2, xs1, ys1);

grid on;
```

```
66  title(sprintf('Output Response, step: %d - %d rpm', u_lo, u_hi));
67  ylabel('rpm');
68  xlabel('time (sec)');
69  legend('no control', 'direct design');
70
71  print('dd_plot2.eps', '-depsc2');
```

Listing 14: Matlab source used to plot the controller built using Direct Design compared to no control.

G Control Law Matlab Source

The following Matlab source code was using for the Control Law Designs (Section 3, 4).

```
%
% cl_ff01.m
%
% Control Law, Intuitive Design.
%

clear;

T = 1;  % time step
Tend = 200;  % end in ticks
ncyl = 8;  % number of cylinders
% stabilizes at rpm_lo then steps to rpm_hi
rpm_hi = 700;  % idle rpm
rpm_lo = 200;

% choose any K <= 1
% larger values increase overshoot
K = 0.01;

Gz = engine_model(T);

% Build System
% upper branch
sum1 = sumblk('e1 = u + r1');
% convert rpm to control
Rz = tf([1], [1450.1], T, 'InputName', 'e1', 'OutputName', 'x1');
set(Gz, 'InputName', 'x1', 'OutputName', 'x2');
sum2 = sumblk('y = x2 - r2');
sys1 = connect(sum1, Rz, Gz, sum2, {'u', 'r1', 'r2'}, {'y'});
% feedback branch
Kz = tf([-K], [1], T, 'InputName', 'y', 'OutputName', 'u');
% full system
sysX = feedback(sys1, Kz, [1], [1], +1);
sysX = sminreal(sysX);

% Simulate
x0 = [1; zeros(length(sysX.a) - 1, 1)];
u1 = 0*ones([Tend/T 1]);
c1 = Tend/T;
stepc = c1/2;  % point at which step occurs
r1 = [rpm_lo*ones([stepc 1]); rpm_hi*ones([stepc 1])];;
r2 = r1;
u = [u1 r1 r2];
[y1,t1] = lsim1(sysX, u, [], x0);
y2 = y1 + r2;  % compensate for zero output in plot

% be sure to avoid zeros, rpmtime doesn't handle them
%yX = y2(10:end);  % all data, except first zeros (approx)
yX = y2((stepc+1):end);  % data after step
rt = rpmtime(yX, ncyl);
rt = zerotime(rt);

% Plot
clf;
figure(1);
[ts1,ys1] = stairs(rt,yX);
plot(ts1,ys1);
grid on;
axis tight;
title('Control Law, Intuitive Design');
```

```
61 xlabel('time (sec)');
62 ylabel('rpm');
63
64 print('cl_ff01.eps', '-depsc2');
```

Listing 15: Matlab source used to plot the Control Law regulator.

```
1  %
2  % cl_fonp03.m
3  %
4  % Control Law, Full Order, No Prediction
5  %
6
7  clear;
8
9  T = 1;  % time step
10 ncyl = 8;  % number of cylinders
11 % stabilizes at rpm_lo then steps to rpm_hi
12 rpm_hi = 700;  % idle rpm
13 rpm_lo = 200;
14
15 % Engine Model
16 Gy = engine_model(T);
17 [Phi, Gamma, H, J] = ssdata(Gy);
18 n = length(Phi);  % order
19 % engine model that outputs x instead of y
20 Gx = ss(Phi, Gamma, eye(n), zeros(n,1), T);
21
22 % Rz converts rpm to control (0-1)
23 Rz = ss(0, 0, 0, [1/1450.1], T);
24
25 % Hz gain, y = H*x
26 Hz = ss(0, zeros(1,n), 0, H, T);
27
28 % Find K
29 % roots (arbitrary)
30 %z1 = [(0.9 + 0.05i) (0.9 - 0.5i) 0];
31 z1 = [(0.9 + 0.05i) (0.9 - 0.05i) 0.01];
32 %z1 = [0.5 0.4 0];
33 %z1 = [0.05 0.04 0];
34 K1 = place(Phi, Gamma, z1);
35 % create a gain of -K using D and zeroing others
36 K1z = ss(zeros(1,1), zeros(1,n), zeros(1,1), -K1, T);
37
38 % Build System
39 % uncompensated feedback loop
40 sum1 = sumblk('x2 = x1 - rpm1');
41 sum2 = sumblk('x2b = x1b - rpm1b');
42 sum3 = sumblk('x2c = x1c - rpm1c');
43 set(Gx, 'InputName', 'e1', 'OutputName', {'x1', 'x1b', 'x1c'});
44 sys1 = connect(sum1, sum2, sum3, Gx, {'e1', 'rpm1', 'rpm1b', 'rpm1c'}, {'x2', 'x2b', 'x2c'});
45 sys2 = feedback(sys1, K1z, [1], [1 2 3], +1);
46 set(sys2, 'InputName', {'u2', 'rpm1', 'rpm1b', 'rpm1c'}, 'OutputName', {'x2', 'x2b', 'x2c'});
47 set(Rz, 'InputName', {'u1'}, 'OutputName', {'u2'});
48 sys3 = connect(sys2, Rz, {'u1', 'rpm1', 'rpm1b', 'rpm1c'}, {'x2', 'x2b', 'x2c'});
49 % add y = H*x
50 sys4 = series(sys3, Hz);
51 sys4 = sminreal(sys4);
52
53 % Input signal, rpm_lo then steps to rpm_hi
54 Tend = 140;  % end in ticks
55 c1 = Tend/T;
56 sc = c1/2;  % point at which step occurs
57 u1 = rpm_lo*ones([sc 1]);
58 u2 = rpm_hi*ones([sc 1]);
```

```
59  ux = [u1; u2];
60  u = [ux ux ux ux];
61
62  % Simulate
63  % initial x0
64  x0 = rpm_lo*H;
65  [y1,t1] = lsim1(sys4, u, [], x0);
66
67  % be sure to avoid zeros, rpmtime doesn't handle them
68  yX = y1;
69  %yX = y2(10:end);  % all data, except first zeros (approx)
70  %yX = y2((stepc+1):end);  % data after step
71  rt = rpmtime(yX, ncyl);
72  rt = zerotime(rt);
73
74  % Plot
75  clf;
76  figure(1);
77  [ts1,ys1] = stairs(rt,yX);
78  plot(ts1,ys1);
79  grid on;
80  axis tight;
81  title('Control Law, Full Feedback, No Prediction');
82  xlabel('time (sec)');
83  ylabel('rpm');
84
85  print('cl_fonp03.eps', '-depsc2');
```

Listing 16: Matlab source used to plot the Control Law with no predicition if sum block is ignored (Figure 17).

H Colophon

This written in LaTeXand built using Pdflatex. The block diagrams were written using TikZ.[13] All calculations and plots were run using version 3.6.4 of Octave,[14] an open source Matlab clone. Except for a few instances (`cl_fonp02.m`, `cl_fonp03.m`) the M-Files are completely compatible between both Octave and Matlab. The entire control system is built within the M-File and has no dependence on Simulink. Octave uses the Octave Control (CACSD) Tools[15] which have equivalents in Matlab. In a few cases an equivalent Simulink diagram was built and in each of the cases the results agreed. This document and all of its code are released under the GNU General Public License and freely available to download and modify.[16]

[13] Till Tantau. *TikZ*. version 2.10. 2013. URL: `http://www.ctan.org/pkg/pgf`.

[14] Octave community. *GNU/Octave*. 2012. URL: `www.gnu.org/software/octave/`.

[15] Lukas F. Reichlin. *Computer-Aided Control System Design (CACSD) Tools for GNU Octave*. Version 2.6.0. 2013. URL: `http://octave.sourceforge.net/control/`.

[16] Jeremiah Mahler. *Control System Design Applied to Idle Stabilization of a Spark Ignition Engine*. 2013. URL: `http://www.github.com/jmahler/idle-control`.

References

Butts, K., N. Sivashankar, and J. Sun. "Feedforward and feedback design for engine idle speed control using l1 optimization". In: *American Control Conference, Proceedings of the 1995*. Vol. 4. 1995, 2587–2590 vol.4. DOI: 10.1109/ACC.1995.532315.

Franklin, G.F., J.D. Powell, and M.L. Workman. *Digital Control of Dynamic Systems*. Addison-Wesley world student series. Addison-Wesley Longman, Incorporated, 1998. ISBN: 9780201331530.

Mahler, Jeremiah. *Control System Design*. 2013. URL: http://www.github.com/jmahler/control-system-design.

– *Control System Design Applied to Idle Stabilization of a Spark Ignition Engine*. 2013. URL: http://www.github.com/jmahler/idle-control.

Octave community. *GNU/Octave*. 2012. URL: www.gnu.org/software/octave/.

Powell, B.K. and J. A. Cook. "Nonlinear Low Frequency Phenomenological Engine Modeling and Analysis". In: *American Control Conference, 1987*. 1987, pp. 332–340.

Reichlin, Lukas F. *Computer-Aided Control System Design (CACSD) Tools for GNU Octave*. Version 2.6.0. 2013. URL: http://octave.sourceforge.net/control/.

Tantau, Till. *TikZ*. Version 2.10. 2013. URL: http://www.ctan.org/pkg/pgf.