A Survey of Control Systems for Stabilizing the Idle of a Spark Ignition Engine

Jeremiah Mahler
jmahler@mail.csuchico.edu

CSU Chico
December 1, 2013

DRAFT

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 shows how control systems methods can be applied to the problem of idle stabilization. Methods include: pole placement, PID, direct design, and various state space designs.

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. A single control input representing the idle air bypass valve

To simplify analysis the linearized model is used as shown in Figure 1.

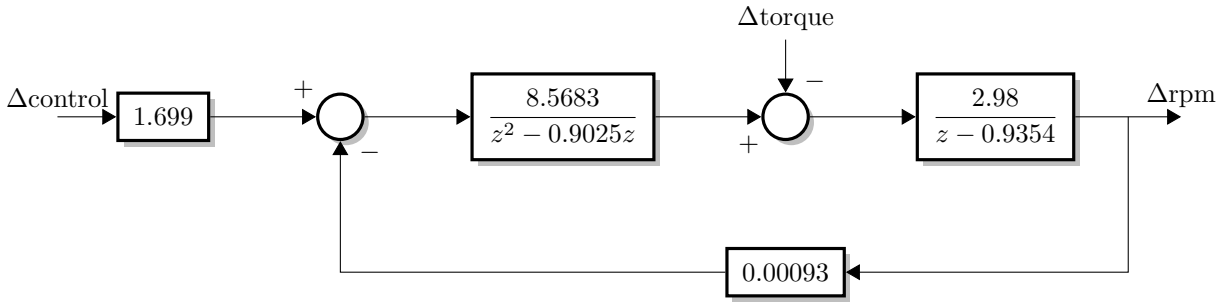

Figure 1: Linear engine model of a modern 4.6L V-8.

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. Often the duty cycle range is in a range from 0% to 100% which corresponds to 0 to 1 (unity).

Because all the inputs and outputs are defined as deltas (Δ) this model cannot be used directly with typical control systems which use steady state values. It is possible convert these deltas to steady state

[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.

equivalents. Figure 2 shows the transfer function to convert steady state values to delta values. Figure 3 shows the transfer function to delta values to steady state values.

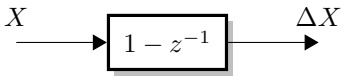

Figure 2: The Z transform used to accumulate the input and convert a steady state input to delta output. Its derivation is given in Appendix A.

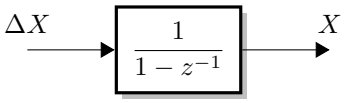

Figure 3: The Z transform used to convert delta input to a steady state output. Its derivation is given in Appendix B.

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.

In order to build a controller the model needs to be simplified but the presence of two inputs 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.

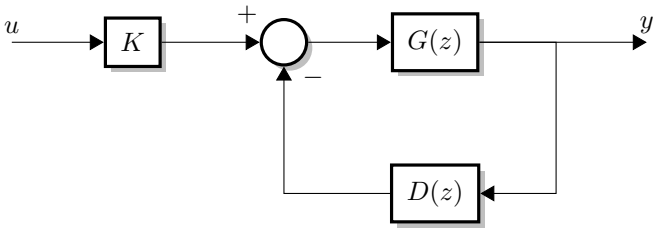

Figure 4: Direct Design system with K as a scaling input, G is the plant and D is the controller.

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

First taking the parts from the engine model (Figure 1).

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

Then substituting them in to Equation 1 and simplifying results in Equation 2.

$$\frac{y}{u} = \frac{(1.699)(8.5683)(2.98)(1)}{(z^2 - 0.9025z)(z - 0.9354)(1) + (8.5683)(2.98)(0.00093)}$$

$$\boxed{\frac{y}{u} = \frac{43.38}{z^3 - 1.838z^2 + 0.8442z + 0.02375}} \tag{2}$$

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 transfer function 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 [3]. Therefore the final transform is still Equation 2. An example of the output response with no control is shown in Figure 6.

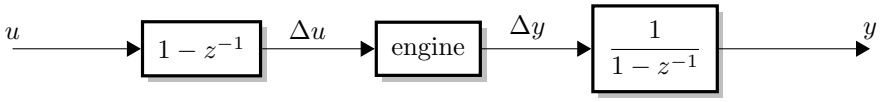

Figure 5: Engine model with transforms for converting from steady state to delta and vice versa.

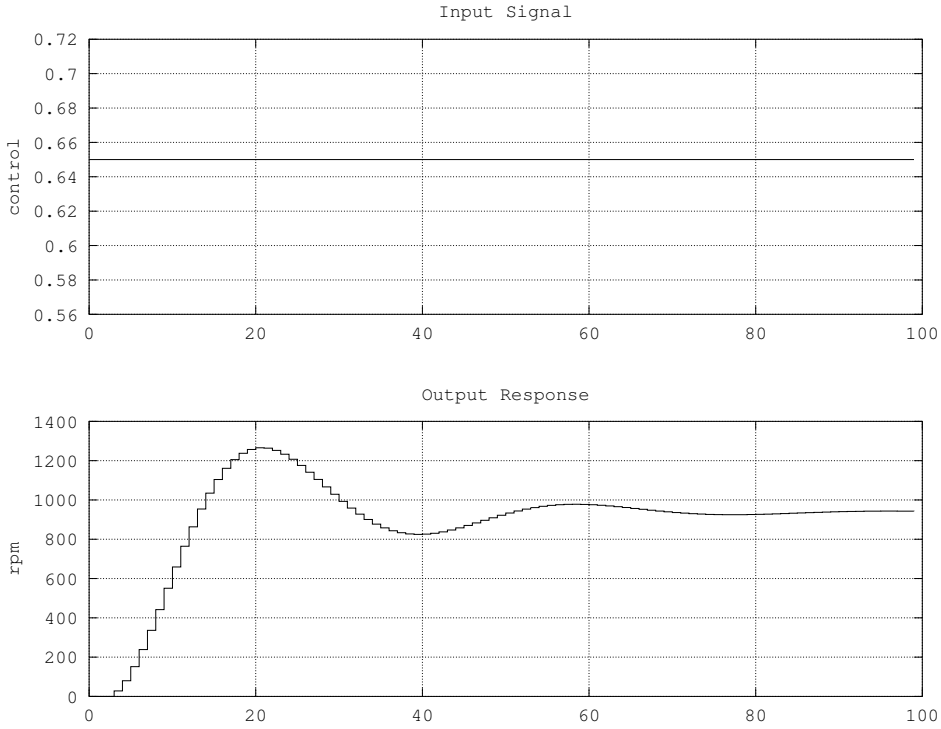

Figure 6: Output response of engine model with no control. The control can range from zero to one. Here the control is constant at 0.65 which resulted in a stable rpm near 900.[5]

[3] In the more general case, when torque is non zero, the transforms would not cancel.

[5] The Matlab source to produced these plots is in Appendix C.

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

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

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

Applying this equation results in the response shown in Figure 7. It can be seen it takes over two seconds for the system to stabilize with no control. Clearly this is unacceptably slow and performance could be improved by using a controller.

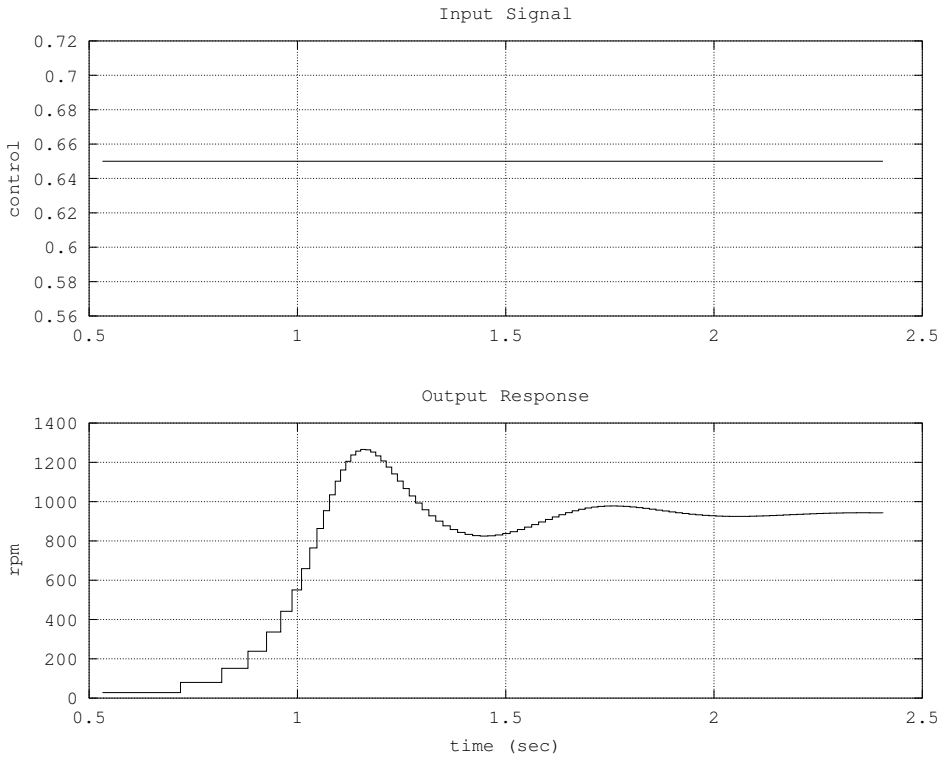

Figure 7: Engine rpm response compared to time with a constant 0.65 input control.[6]

There is yet another complication with this model. Typical systems are in a normalized so that it is possible to achieve an output of one given a step input. With this engine model an output of 1 rpm is an impossibly slow value. And an input of zero or one rpm are similarly impossible.

[6]The Matlab source of the RPM vs time plot is in Appendix C.

To resolve this issue the values will be scaled in to a normal range. A normal idle rpm range is from 600 to 1200 rpm. To scale these values so that zero equals 600 and one equals 1000 Equation 4 can be used.

$$y = \frac{1}{400}x - \frac{3}{2} \tag{4}$$

Figure 8 shows the stable rpms for a given control input. Its behavior is described by Equation 5.

$$y = 1450.1x \tag{5}$$

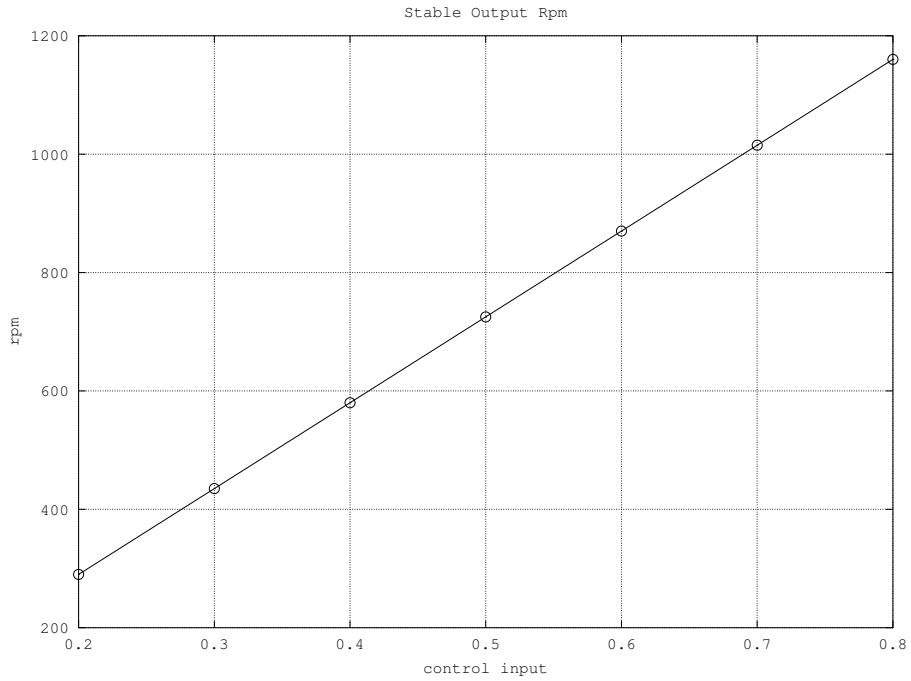

Figure 8: Stable rpm reached for a given constant control input.

With these normalization corrections applied the response can be seen in Figure 9. To avoid the zero rpm case it is allowed to stabilize before the step is applied. The lowest plot shows the response after the step. It takes slightly over one second to stabilize, which is better than the inaccurate case from zero rpm, but it is still slow and can be improved.

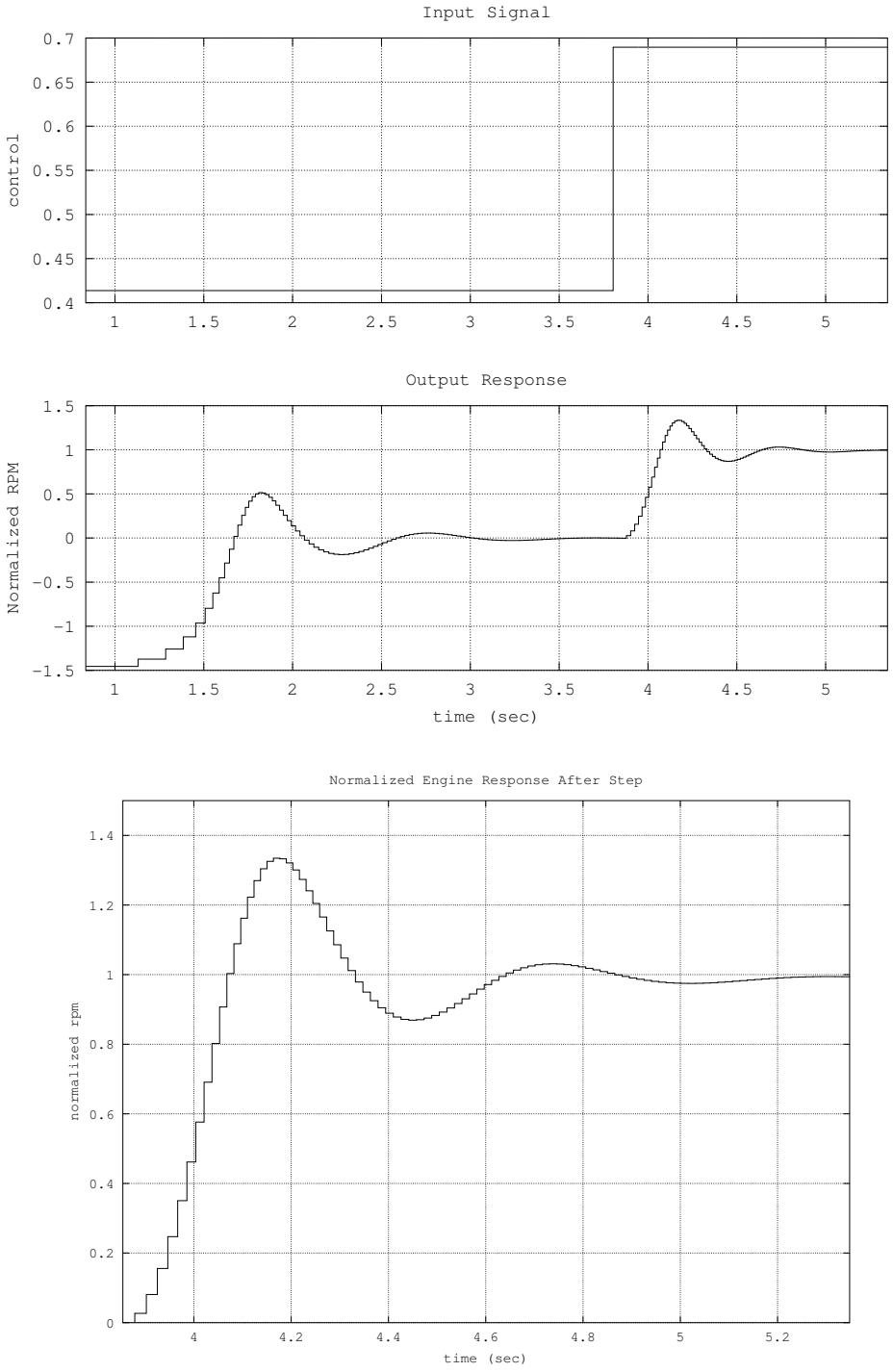

Figure 9: Normalized engine model response. After it stabilizes at unit step is applied to produce the normalized response. The lowest plot is a close up of the step response in the center plot.

2 Pole Placement

3 Regulator Control System

What is needed is a regulator system where the input is applied as torque. The general system is shown in Figure 10.

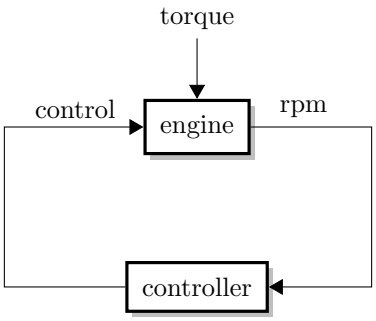

Figure 10: General system for regulating engine rpm.

4 Conclusion

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.

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.

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 11. 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).

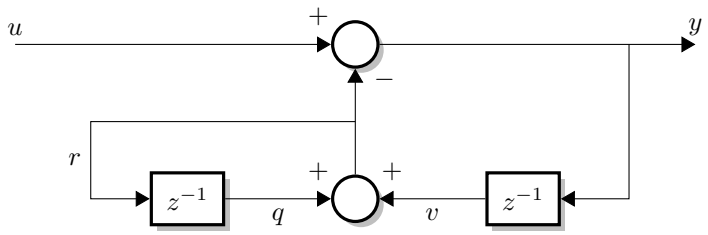

Figure 11: System to accumulate values to convert a steady state input to a delta output.

Figure 12 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.

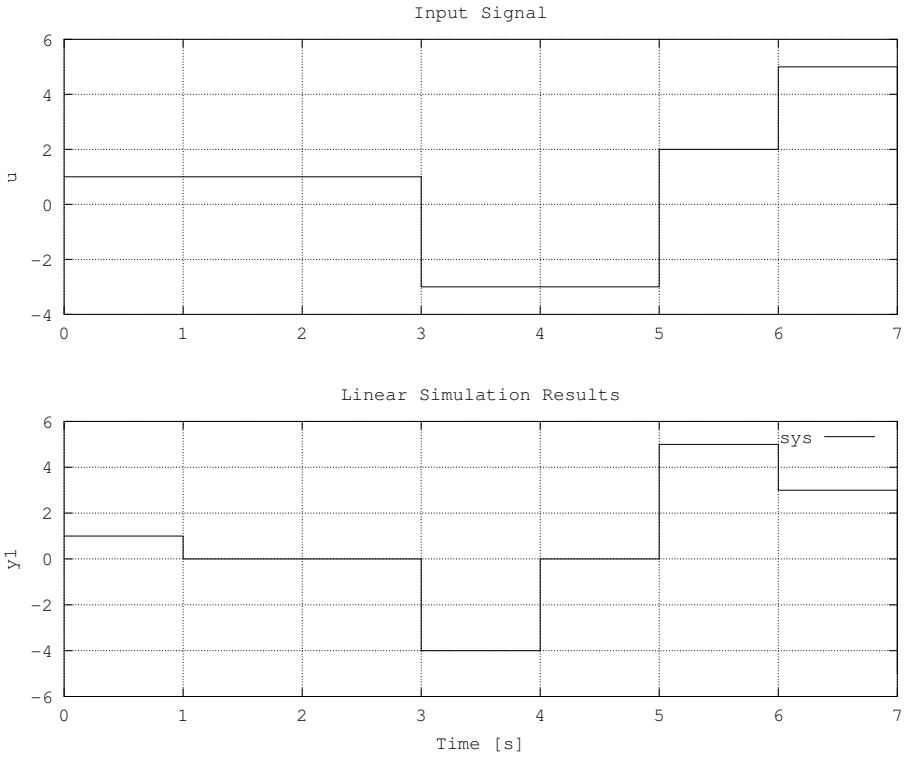

Figure 12: Response of full steady state input to delta output system to an arbitrary input signal. The upper plot is the input signal (u) and the lower plot is the output response (y). The Matlab source code is given in Listing 1 and 2.

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

$$r = q + v \tag{6}$$
$$v = y \cdot z^{-1} \tag{7}$$
$$q = r \cdot z^{-1} \tag{8}$$
$$y = u - r \tag{9}$$

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

$$\begin{aligned} r &= rz^{-1} + yz^{-1} && (6, 7, 8) \\ r(1 - z^{-1}) &= yz^{-1} \\ r &= u - y && (9 \\ (u - y)(1 - z^{-1}) &= yz^{-1} \\ u - y - uz^{-1} + yz^{-1} &= yz^{-1} \\ u - y - uz^{-1} &= 0 \\ y &= u(1 - z^{-1}) \end{aligned}$$

$$\boxed{\frac{y}{u} = 1 - z^{-1}} \tag{10}$$

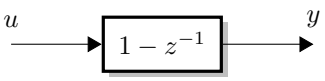

Figure 13: Simplified system to convert a steady state input in to a delta output.

It can be seen in Figure 14 that the simplified system behaves identically to the previous system (Figure 12).

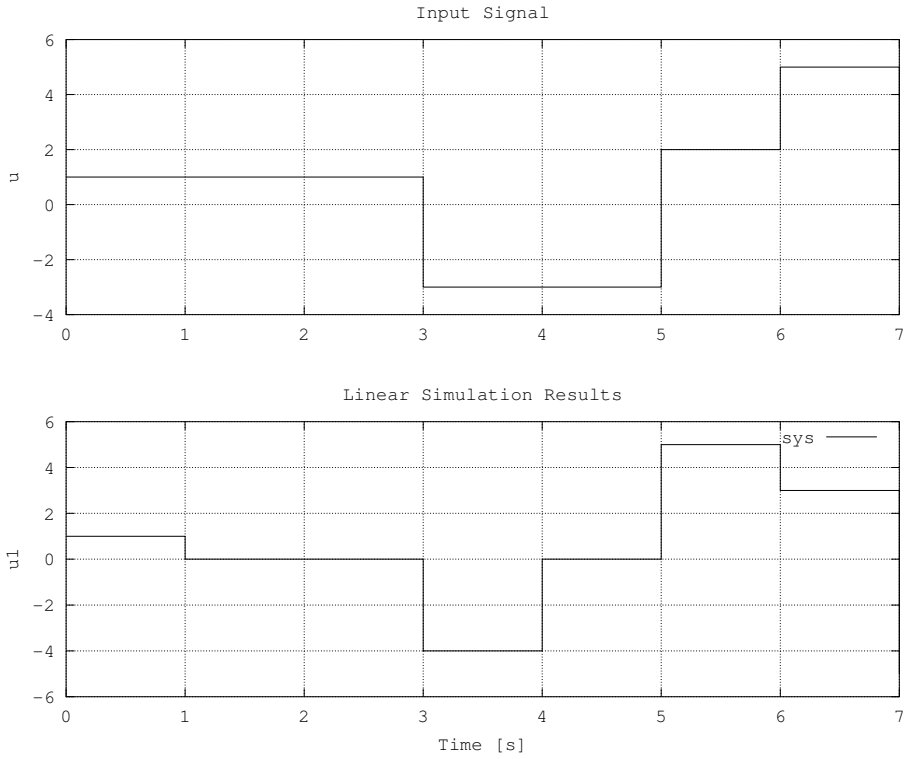

Figure 14: Response of simplified steady state input to delta output system to an arbitrary input signal. The upper plot is the input signal (u) and the lower plot is the output response (y). Response is identical to the full system in Figure 12 as expected. The Matlab source code is given in Listing 3 and 4.

A.1 Matlab Source

The following code has been tested using Octave,[7] an open source Matlab clone.

```
%
% cd1_init.m
%
% Convert a steady state input to a delta output.
%
% Full System.
%

T = 1;  % time step

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

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

```
%
% cd1_plot.m
%

clear;

cd1_init;

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('cd1_plot.eps', '-deps');
```

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

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

```
%
% cd2_init.m
%
% Convert a steady state input to a delta output.
%
% Simplified System.
%

T = 1;  % time step

sys = tf([1 -1], [1 0], T, 'inname', 'y1', 'outname', 'u1');
```

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

```
%
% cd2_plot.m
%

clear;

cd2_init;

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('cd2_plot.eps', '-deps');
```

Listing 4: 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 15 shows the system.

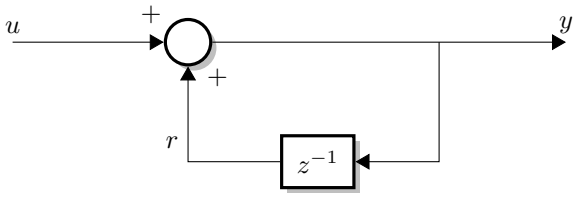

Figure 15: System to convert delta input to steady state output.

This system can be simplified in to a single transfer function as given by Equation 11 and shown in Figure 16.

$$\begin{aligned} y &= u + r \\ r &= y \cdot z^{-1} \\ y &= u + yz^{-1} \\ u &= y(1 - z^{-1}) \\ \boxed{\frac{y}{u} &= \frac{1}{1 - z^{-1}}} \end{aligned} \tag{11}$$

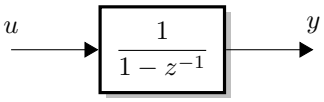

Figure 16: Simplified system to convert a delta input to a steady state output.

Figure 17 shows the response of this system given an arbitrary input [8]. It can be seen that the output is held in a steady state according to the delta inputs as expected.

[8] 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.

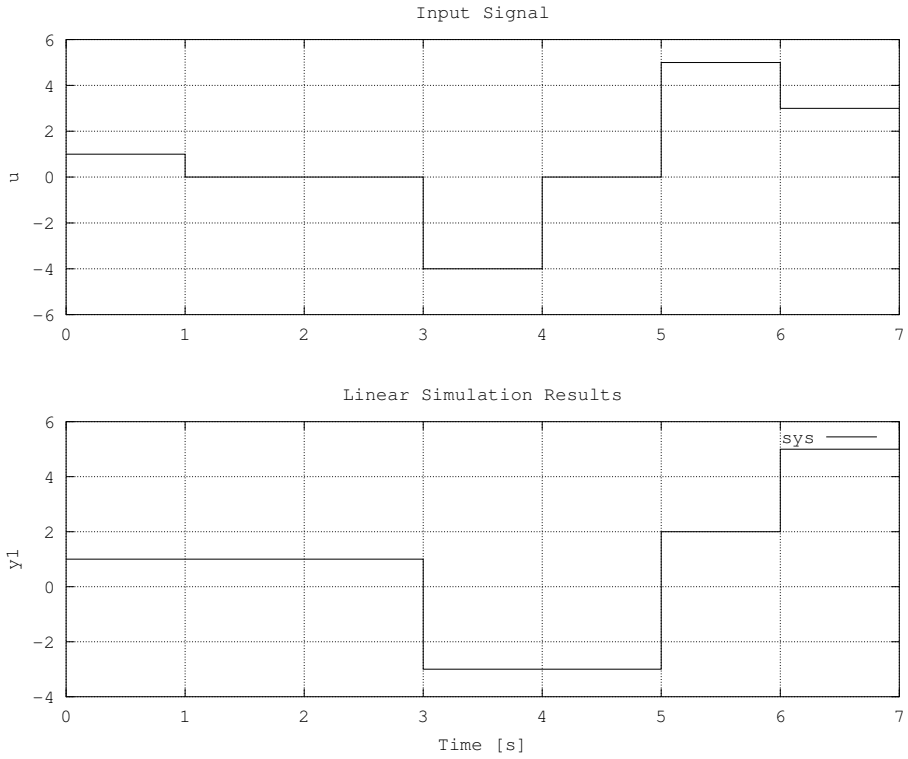

Figure 17: Response of delta input to steady state output system when given an arbitrary input signal. The upper plot is the input signal (u) and the lower plot is the output response (y). The Matlab source code is given in Listing 5 and 6.

B.1 Matlab Source

The following code has been tested using Octave,[9] an open source Matlab clone.

```
%
% dc1_init.m
%
% Convert a delta input to a steady state output.
%
% Simplified System.
%

T = 1;  % time step

sys = tf([1 0], [1 -1], T, 'inname', 'u1', 'outname', 'y1');
```

Listing 5: Matlab code to initialize the simplified steady state to delta system.

```
%
% dc2_plot.m
%

clear;

dc2_init;

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('dc2_plot.eps', '-deps');
```

Listing 6: Matlab code to plot the simplified steady state to delta system.

[9] Octave community, see n. 7.

C Engine Model Matlab Source

```
%
% engine_model.m
%
% 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);

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

Listing 7: Matlab source to calculate the engine model system with zero torque.

```
%
% engine_model_plot.m
%

clear;

engine_model;

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

figure;
subplot(2,1,1);

stairs(t, u);
grid on;
axis auto;
title('Input Signal');
ylabel('control');

subplot(2,1,2);
stairs(t, y);
grid on;
axis auto;
title('Output Response');
ylabel('rpm');

print('engine_model_plot.eps', '-deps');
```

Listing 8: Matlab source to plot the response of engine model with no control.

```
%
% engine_model_rpmtime_plot.m
%

clear;

ncyl = 8;

engine_model;

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

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

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

print('engine_model_rpmtime_plot.eps', '-deps');
```

Listing 9: Matlab source to plot the response of engine model versus time.

```
%
% norm_engine_model_plot.m
%

clear;

ncyl = 8;

engine_model;
% eng_sys

u0 = 600*1/1450.1;
u1 = 1000*1/1450.1;

% input signal
u = [u0*ones(1,100) u1*ones(1,100)];

% output response
[y, t, x] = lsim(eng_sys, u);

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

rt = rpmtime(y, 8);
rt = cumsum(rt);

% normalized output response
yn = y*(1/400) - 3/2;

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, yn);
ylabel('Normalized RPM');
grid on;
axis ([rt(1), rt(end)]);
title('Output Response');
xlabel('time (sec)');

print('norm_engine_model_plot.eps', '-deps');
```

Listing 10: Matlab source to plot the response of the normalized engine model.

```
%
% norm_engine_plot_close.m
%

clear;

ncyl = 8;

engine_model;
% eng_sys

u0 = 600*1/1450.1;
u1 = 1000*1/1450.1;

% input signal
u = [u0*ones(1,100) u1*ones(1,100)];

% output response
[y, t, x] = lsim(eng_sys, u);

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

rt = rpmtime(y, 8);
rt = cumsum(rt);

% normalized output response
yn = y*(1/400) - 3/2;


figure;

stairs(rt(100:end), yn(100:end));
grid on;
axis([rt(100), rt(end) 0 1.5]);
title('Normalized Engine Response After Step');
xlabel('time (sec)');
ylabel('normalized rpm');

print('norm_engine_plot_close.eps', '-deps');
```

Listing 11: Matlab source to plot the response of the normalized engine model after step is applied.