Design and Testing of an Analogue Controller

David Collins (C10736685)
Programme: DT021
Year: 2
Module: Introduction to Control
Module coordinator: Mr. Gerard Caffrey
March 2012

Abstract

This series of labs looks at two aspects of basic control theory. Firstly, the simplication of higher-order systems is considered — the modelling of third-order systems with first-order approximations specifically. Secondly, three categories of control systems are examined: 1. proportional controllers, 2. proportional+integral controllers, and finally 3. PID (proportional+integral+derivative) controllers. In order to highlight the need for process control — i.e. why controllers are needed in the first place — the concept of load disturbance is introduced briefly.

There are a variety of ways to derive the parameters necessary to implement PID controllers. The formulas used to do so are referred to as 'tuning formulas'. In this lab, relatively simple formulae developed by Ziegler and Nichols will be employed.

Much of the modelling and simulation was performed using MATLAB — using the Simulink package. The use of software to test the models derived greatly facilitates the process, and obviates the need to perform tests with actual hardware (e.g. mechanical, electrical or hydraulic systems).

Following the simulation in software, the system was implemented in hardware using a function generator and process control simulator. Measurements were taken using a digital oscilloscope, and based on these a comparison between the software and hardware implementations was made.

Contents

1 Introduction

1.1 First-Order Lag Plus Deadtime Model of Higher-Order Systems

The time-domain step response of a third-order process is a *sigmoidal* graph. Such a function can be approximated by a simpler expression of the form

$$c'(t) = Ak_p(1 - e^{-t/\tau}) \tag{1}$$

where A is the amplitude of the step input, k_p is the steady-state gain of the original third-order system, and τ is the time-constant of the system.

It is important to note that the approximation in equation 1 must be used in conjunction with a *time delay*, T_d. Thus the correct expression is actually

$$c(t) = Ak_pH(t - T_d)(1 - e^{-(t-T_d)/\tau}), \tag{2}$$

where $H(t)$ denotes the Heaviside step function. Then, according to the t-shift theorem, the transfer function for this system is

$$G_p(s) = \frac{k_pe^{-sT_d}}{1 + \tau s}, \tag{3}$$

and the system itself is referred to as a *first-order lag plus deadtime model*. Our initial task in this series of labs is to determine the parameters τ and T_d, which are the time-constant and time-delay for our first-order model respectively.

1.2 PID Controllers

Consider a feedback system in which the feedback is

$$e(t) = r(t) - c(t),$$

where $c(t)$ is the *actual* system output and $r(t)$ is the set-point or desired output. A proportional controller, then, is one in which the controller transfer function is a single term $k_ce(t)$. A PID controller, on the other hand, also includes terms for both of the derivative and integral of the error, giving

$$u(t) = k_ce(t) + k_i \int e(t)dt + k_d\frac{d}{dt}e(t) \tag{4}$$

Qualitatively, this means that the instantaneous controller gain depends not only on the current value of the error, but also on the rate at which it is changing. Furthermore, due to the integral term, the gain also depends on the history of previous values.

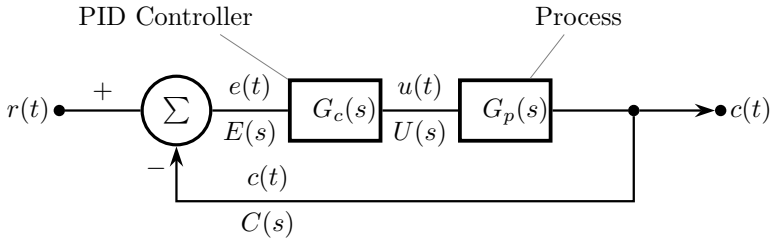


Figure 1: Canonical Block Diagram

1.3 Controller Tuning Formulae

Referring to Figure 1, the second block, $G_p(s)$, is the third-order process we want to model. The first block, $G_c(s)$, is a PID controller. The primary purpose of the controller is to compensate for the effect of *disturbances* on the sytem. During the laboratories, we will determine the parameters necessary to achieve this as effectively as possible (Section 3.2). We will do this using a set of formulae provided by Ziegler and Nichols (Appendix A).

Before we can derive the parameters k_c, τ_i and τ_d necessary for the PID controller, we first need to derive a first-order model of our the third-order process. This will yield the time constant τ and delay time T_d upon which the Ziegler-Nichols formulae depend.

2 Modelling of Third-Order Systems

We begin with a third-order system whose transfer function is

$$G_p(s) = \frac{k_1}{1+\tau_1 s} \cdot \frac{k_2}{1+\tau_2 s} \cdot \frac{k_3}{1+\tau_3 s}, \tag{5}$$

and in this case we chose the simple values

$$k_1 = 1, \quad k_2 = 1, \quad k_3 = 1, \quad \tau_1 = 1, \quad \tau_2 = 2, \quad \tau_3 = 3.$$

Before attempting to determine a first-order approximation of the system, we can place the transfer function in a condensed form for convenience. The above

equation can be rewritten:

$$\begin{aligned} G_p(s) &= \frac{k_1k_2k_3}{(1+\tau_1 s)(1+\tau_2 s)(1+\tau_3 s)} \\ &= \frac{k_1k_2k_3}{(1+\tau_1 s)(1+s(\tau_2+\tau_3)+\tau_2\tau_3 s^2)} \\ &= \frac{k_1k_2k_3}{1+s(\tau_2+\tau_3)+\tau_2\tau_3 s^2+\tau_1 s+s^2(\tau_1\tau_2+\tau_1\tau_3)+\tau_1\tau_2\tau_3 s^3} \\ &= \frac{k_1k_2k_3}{\tau_1\tau_2\tau_3 s^3+s^2(\tau_1\tau_2+\tau_1\tau_3+\tau_2\tau_3)+s(\tau_1+\tau_2+\tau_3)+1} \\ &= \frac{1}{6s^3+11s^2+6s+1}. \end{aligned}$$

Placing the transfer function in this form enables us to replace the three gain blocks in our block diagram with one (Figure 3).

The unit step response of the above system is

$$\begin{aligned} C(s) &= R(s)G_p(s) \\ &= \frac{1}{s}\frac{1}{6s^3+11s^2+6s+1}. \end{aligned}$$

Taking the inverse Laplace transform of this expression yields the response in the time domain (Figure 2):

$$\begin{aligned} c(t) &= \mathcal{L}^{-1}\left\{\frac{1}{s}\frac{1}{6s^3+11s^2+6s+1}\right\} \\ &= -\frac{9\,e^{-\frac{t}{3}}}{2}+4\,e^{-\frac{t}{2}}-\frac{e^{-t}}{2}+1 \end{aligned}$$

2.1 First-Order Approximation

As explained in the introduction, we want to determine two parameters for our first-order lag plus dead-time model: the time-constant τ and the deadtime T_d. To do this we firstly determined the two times t_1 and t_2 at which the output $c(t)$ had reached 0.28 and .63 of its steady-state value respectively. From Figure 2 we determined these as $t_1 = 3.51$s and $t_2 = 6.437$s. Then,

- $\tau = \frac{3}{2}(t_2 - t_1) = \frac{3}{2}(6.437 - 3.51) = 4.391$s,
- $T_d = t_2 - \tau = 6.437 - 4.391 = 2.05$s.

$$\boxed{\tau = 4.391}, \quad \boxed{T_d = 2.05}$$

From equation 3, the transfer function for the model is then

$$\begin{aligned} G_{p'}(s) &= \frac{k_p e^{-sT_d}}{1+\tau s} \\ &= \frac{e^{-2.05s}}{1+4.391s}, \end{aligned}$$

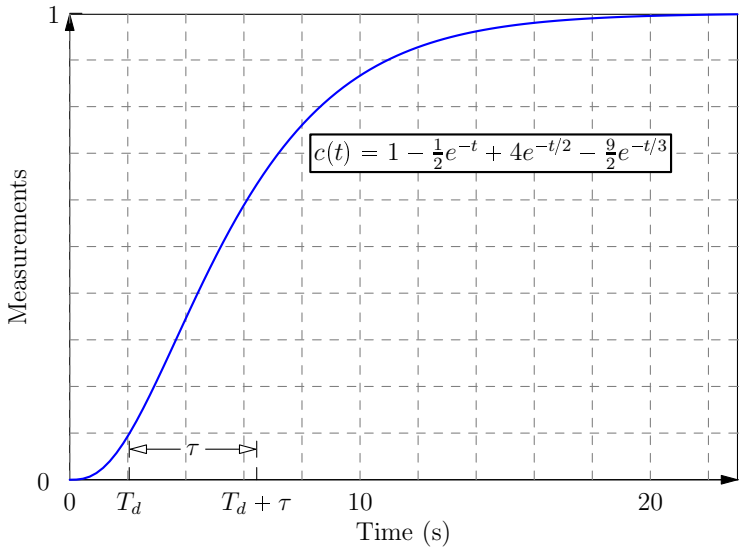
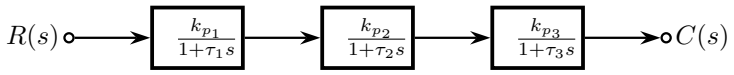
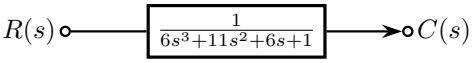


Figure 2: Unit-Step Response of A Third-order Process



(a) Original 3-block Process



(b) Block Diagram Reduction

Figure 3: Open-Loop System (No Controller Block)

since the gain $k_p = 1$, and the response of the process to a unit step input is

$$\begin{aligned} c(t) &= H(t - T_d)\left[1 - e^{-\frac{t-T_d}{\tau}}\right] \\ &= H(t - 2.05)\left[1 - e^{-(t-2.05)/4.39}\right], \end{aligned}$$

where $H(t)$ denotes the Heaviside step function. Having determined both the dead time and the time constant for the first-order approximation, we can plot its unit step response and compare it to that of the original process. This is demonstrated in Figure 5.

The Simulink model used to perform this approximation and comparison is shown in Figure 4. The accompanying MATLAB code is shown in Listing 1.

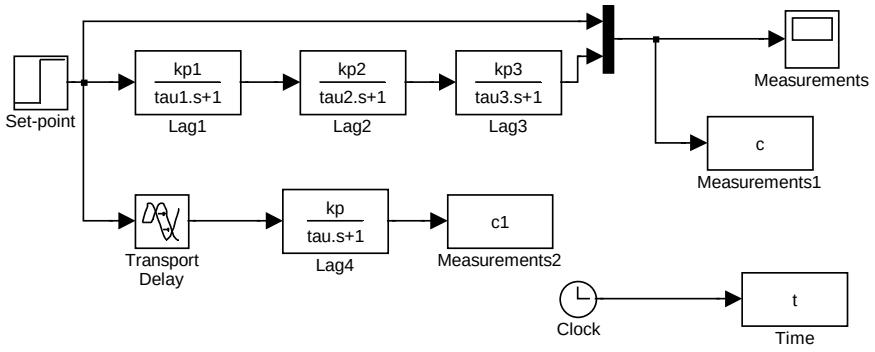


Figure 4: First-Order Model of Third-Order Process — Simulink Diagram

2.2 Disturbances & Feedback

An open-loop system is adequate as long as the process is not *disturbed*. However, if there is a disturbance in the system, the loop needs to be closed in order for the system to compensate (Figure 6b). Without feedback, the system can have no knowledge of the difference between the set-point $r(t)$ and the output $c(t)$. In order to compensate for potential disturbances, we need to 1. provide a 'feedback' path — i.e. route the output $c(t)$ back to a summing junction at the start of the loop, and 2. introduce a *controller* block which responds to this feedback.

Assuming we have unity (inverted) gain feedback, the error signal, $e(t)$, fed into the controller is

$$e(t) = r(t) - c(t).$$

The controller responds to this error to produce an intermediate output

$$u(t) = f(e(t)),$$

where $f(e(t))$ denotes that the controller output is a function of the error $e(t)$. In the case of a PID controller, we know that the general form of this function is

$$u(t) = k_c e(t) + k_i \int e(t)dt + k_d \frac{de(t)}{dt}, \tag{6}$$

where k_c, k_i and k_d are unknown constants which will depend on the system process under consideration. Our task here is to determine values for these three parameters such that the error $e(t)$ is minimised. The Ziegler-Nichols formulae shown in Appendix A will be used for this purpose.

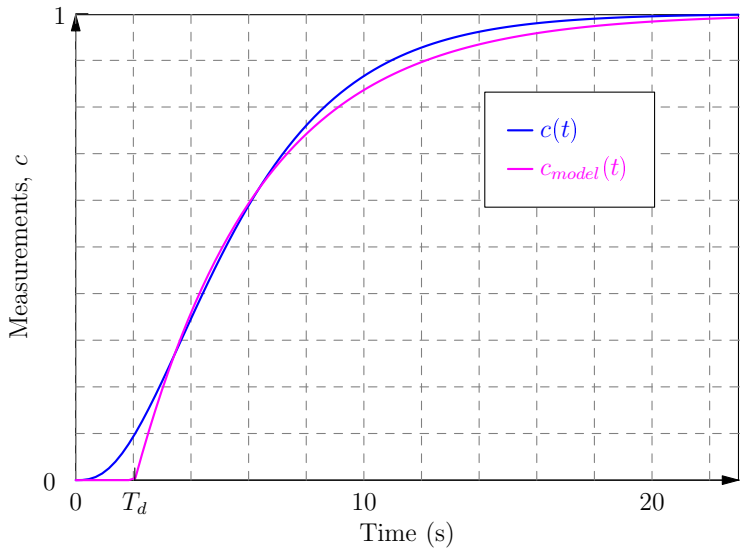
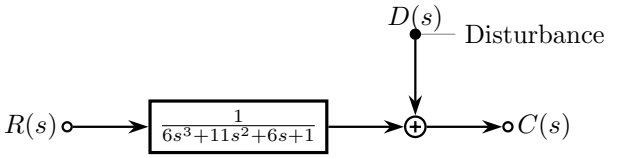
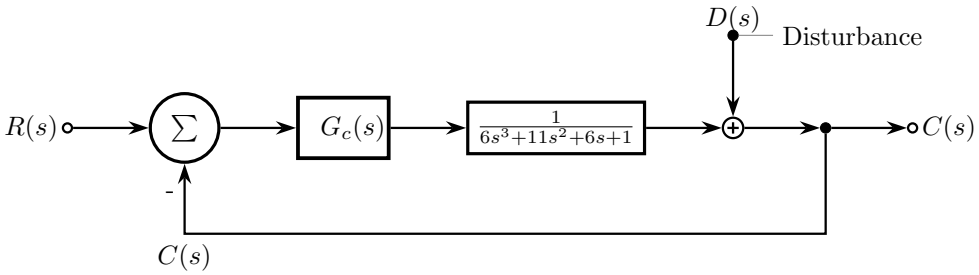


Figure 5: Comparison: Step-response of Third-order Process vs. Step-response of First-order Model



(a) System With Disturbance (No Feedback)



(b) Introduction of Feedback Path and Controller

3 PID Controllers

3.1 Proportional Control

In the case of purely *proportional* control, the time-domain response of the controller is of the form

$$u(t) = k_c e(t). \tag{7}$$

In the Laplace domain, we have

$$U(s) = k_c E(s). \tag{8}$$

Proportional-only control is not as useful as proportional-integral or PID control, however. This can be seen by examining the performance characteristics in Table 2, for example. For this reason, the PID controller will be considered next.

3.2 Proportional-Integral-Derivative (PID) Control

For a proportional-integral-derivative controller, the time-domain response of the controller, i.e. the output $u(t)$ in terms of input $e(t)$, is

$$u(t) = k_c e(t) + k_i \int e(t) + k_d \frac{d}{dt} e(t) \tag{9}$$

In the Laplace domain this becomes

$$U(s) = G_c(s) E(s),$$

where

$$\begin{aligned} G_c(s) &= k_c + k_i \frac{1}{s} + k_d s && (10)\\ &= k_c \left[1 + \frac{k_i}{k_c s} + \frac{k_d}{k_c} s \right] && (11)\\ &= k_c \left[1 + \frac{1}{k_c/k_i s} + \frac{k_d}{k_c} s \right] && (12)\\ &= k_c \left[1 + \frac{1}{\tau_i s} + \tau_d s \right], && (13) \end{aligned}$$

where $\tau_i = \frac{k_c}{k_i}$ and $\tau_d = \frac{k_d}{k_c}$. Once we determine the appropriate values of τ_i and τ_d using the Ziegler-Nichols formulae (Section A), we can calculate the values k_i and k_d according to

$$k_i = \frac{k_c}{\tau_i}, \quad k_d = \tau_d k_c.$$

Using the formulae in Table 3, as well as the values derived for our first-order lag plus deadtime model, we can determine the required controller parameters. The gain $k_p = 1$ so the term $\frac{1}{k_p}$ can be ignored in all cases. The results are shown in Table 1.

	Time form			Gain form	
	k_c	τ_i	τ_d	k_i	k_d
Proportional only	2.14				
Proportional-integral	1.927	6.833		.282	
PID	2.57	4.10	1.025	.627	2.634

Table 1: Parameters for Different Controller Types

4 Software Implementation

The purpose of the controller, as mentioned previously, is to maintain system stability by responding to the system error $e(t) = r(t) - c(t)$. To accomplish this, the controller block must be integrated in to the system as shown in Figure 6b.

Inititally, the simulation was performed on computer. The MATLAB code used is shown in Appendix B. The Simulink diagram is shown in Figure 6. The results are discussed below.

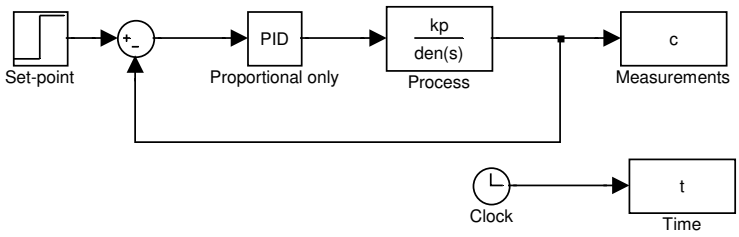


Figure 6: Third-Order Process with Feedback and Controller Block

4.1 Proportional Control

The output of the system with proportional controller is shown in Figure 7. We note that the proportional controller yields a large steady-state error. The percentage overshoot is

$$\frac{c_{max} - c_{ss}}{c_{ss}} \times 100\% = \frac{.86 - .68}{.68} \times 100\% = 26.47\%.$$

4.2 Proportional-integral control

The proportional-integral controller, in contrast, yields a very small steady-state error. This is shown in Figure 8. Despite it's good reference-tracking, it has a

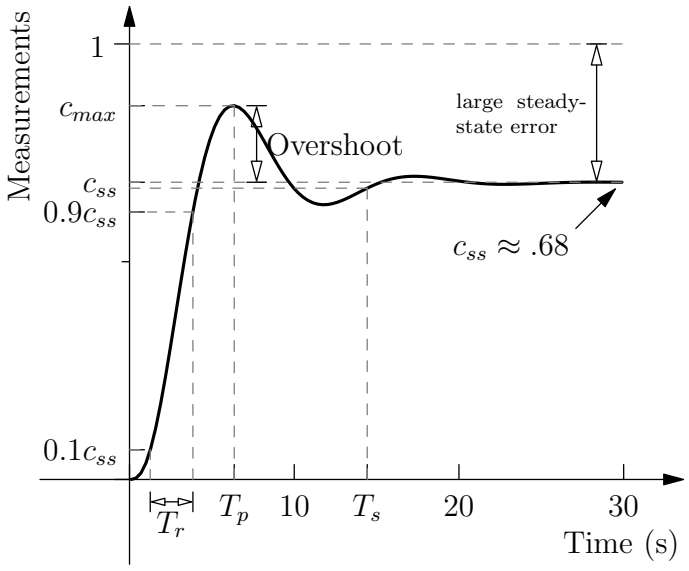


Figure 7: Proportional Controller
$T_r = t_{.9c_{ss}} - t_{.1c_{ss}}$, T_s = time taken for output to reach ±2% of final value.

long rise and settling time (Table 2).

4.3 Proportional-integral-derivative (PID) control

The PID controller has a shorter settling time than either the P or PI controller. It also has a negligible steady-state error. It's primary drawback is that it results in a very large overshoot — much larger than either of the other types of controller (Table 2).

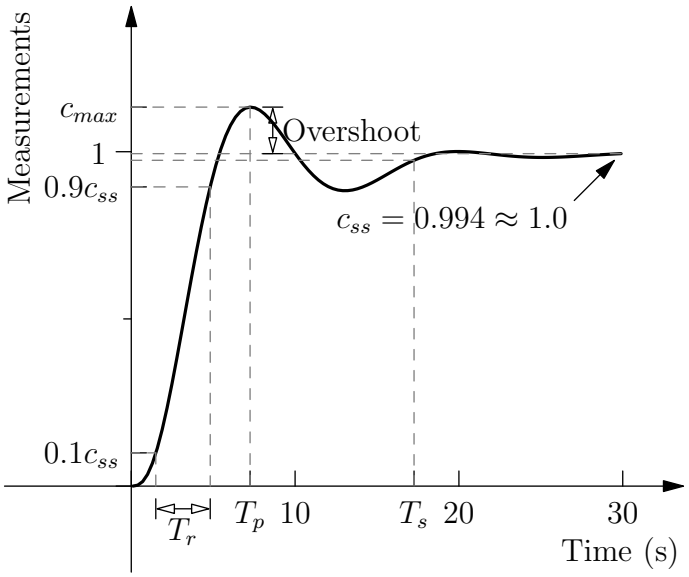


Figure 8: Proportional-Integral Controller

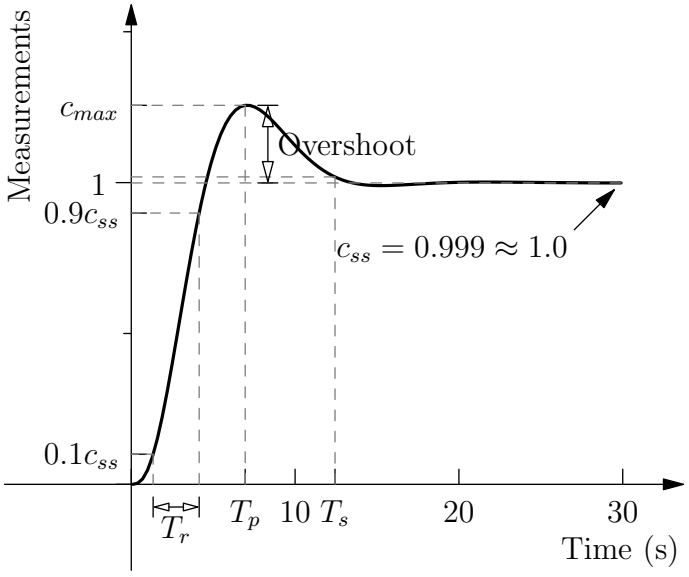


Figure 9: PID Controller

5 Hardware Implementation

Having successfully simulated our control system in software, the next step was to implement it in hardware. The set-point was provided by a function generator supplying a square wave signal with 2 V peak-to-peak amplitude. The frequency was set to 0.02 Hz, yielding a period of 50 s. This essentially provided a unit-step function which was 'reset' (and changed direction) every 25 s. The settling time of all our controller variations (Table 2) was less than 18 s, so a signal of 25 s in duration is adequate to examine the system behaviour.

The output of the proportional, proportional-integral, and PID hardware controllers are shown in Figures 10, 11 and 12 respectively. The corresponding metrics are shown in Table 2

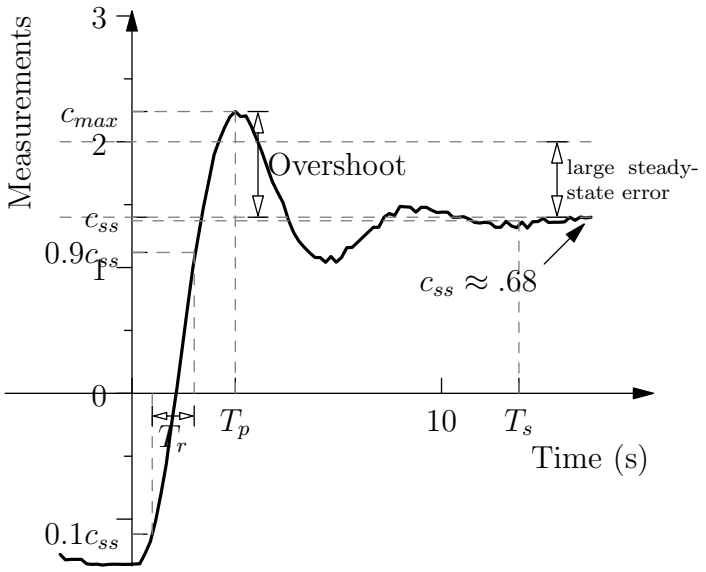


Figure 10: Hardware Proportional Controller

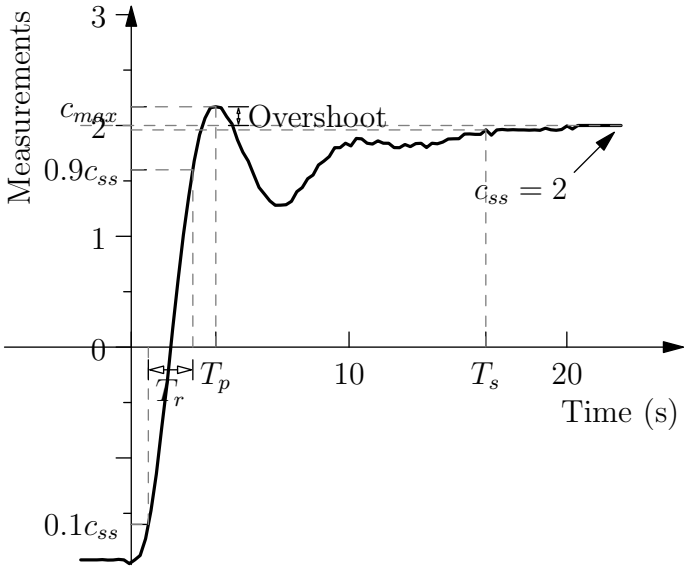


Figure 11: Hardware Proportional-integral Controller

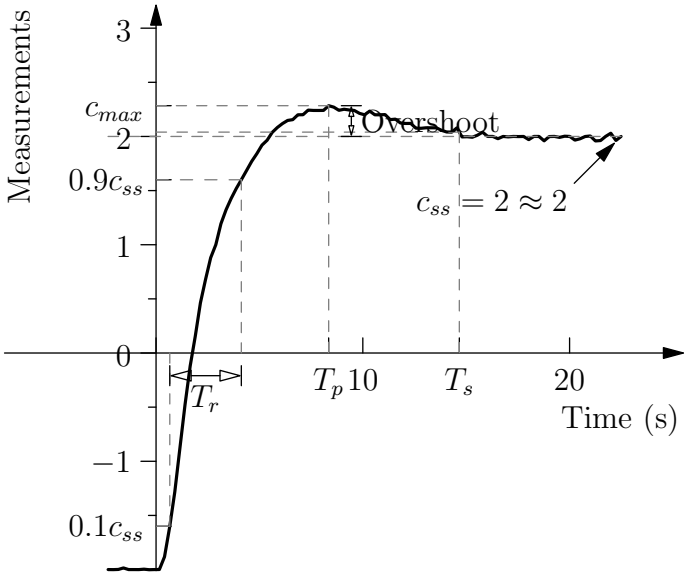


Figure 12: Hardware PID Controller

6 Comparison

6.1 Comparison of Controller Types

Figure 13 provides a visual comparison of the output of each controller as implemented in software.

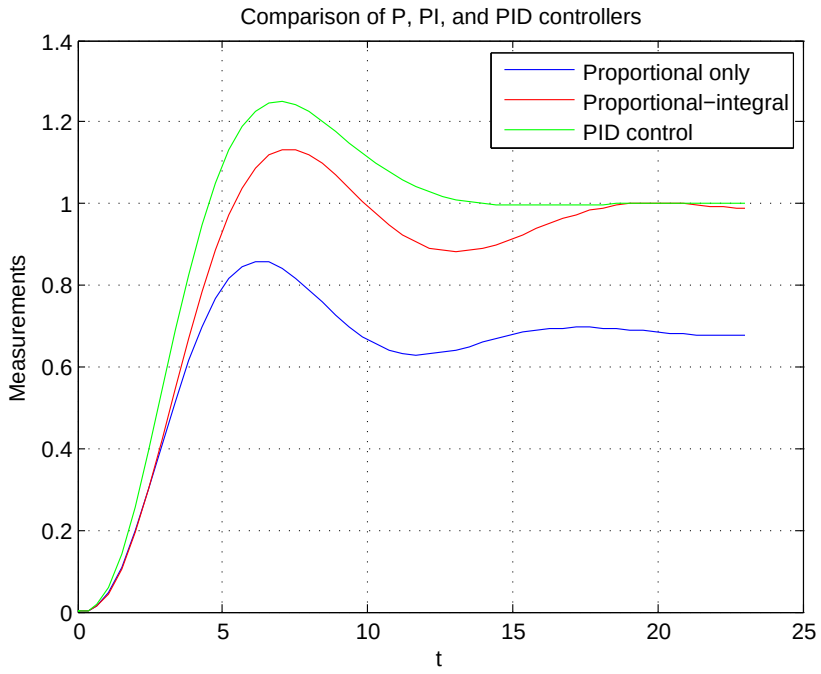


Figure 13: Unit-Step Response Using Different Controller-Types

6.2 Hardware vs. Software

As well as comparing different types of controller (proportional vs. proportional-integral vs. PID), we can compare the results of the software implementation with those of the hardware implementation. Table 2 gives the numerical values of various performance characteristics for each controller type — both the hardware and software implementation. The results of the hardware controllers correspond reasonably well to those of the software implementations. Possible reasons for the discrepancy are discussed in section 7.3.

Controller type		**Steady-state error**	**Overshoot**		T_r (s)	T_p (s)	T_s (s)
		Absolute	Absolute	%			
P	software	0.32	0.18	26.5	2.59	6.36	14.43
	hardware	-	-	30.0	1.36	3.34	12.5
PI	software	0.0	0.12	12	3.32	7.25	17.26
	hardware	-	-	4.22	2.04	3.88	16.28
PID	software	0.0	0.25	25	2.81	6.95	12.44
	hardware	-	-	7.10	3.46	8.35	14.67

T_r = rise time, T_p = peak time, T_s = settling time.

Table 2: Metrics of Systems with Different Controller Types

7 Conclusion

This article highlighted a number of important points relating to first-order models and PID controllers.

7.1 Approximation of Third-Order Systems

Firstly, we have demonstrated that a third-order process can be approximated to a high degree by determining only two parameters, the time constant and the dead-time (Figure 5).

7.2 PID Controllers

Secondly, we demonstrated that — given a particular third-order process (or its first-order approximation) — there exist clearly-defined methods for designing a suitable controller. We examined one such method in this article, and demonstrated its effectiveness in maintaining system stability.

7.3 Sources of discrepancy between hardware and software implementations

The discrepancies between the implementations in hardware and software can be attributed to the following factors.

hardware PID parameters For the hardware implementation the values of k_c, τ_i and τ_d were set using analog controls. These were subject to a large degree of uncertainty and may significantly affected the output values.

set-point In the hardware implementation the set-point was provided by a function generator. Mathematically, the input in this case was a square wave which was oscillated between values of -2 and 2. In the software implementation, in contrast, the input was a unit-step. Considering the different nature of the inputs, we can not expect the same outputs.

A A. Ziegler-Nichols Tuning Method

Control systems have two antagonistic goals:

1. fast response, and
2. stability.

The Ziegler-Nichols method yield good system stability, at the expense of poor response time. The formulae developed by Ziegler and Nichols are as shown in Table 3.

	Proportional gain, k_c	**Integral time**, τ_i	**Derivative time**, τ_d
Proportional controller	$\frac{1}{k_p}\frac{\tau}{T_d}$		
Proportional + integral controller	$\frac{0.9}{k_p}\frac{\tau}{T_d}$	$\frac{T_d}{0.3}$	
PID controller	$\frac{1.2}{k_p}\frac{\tau}{T_d}$	$2T_d$	$\frac{T_d}{2}$

Table 3: Tuning-Formulae for Various Controllers

B B. Source Code

B.1 First-Order Modelling

The first part of the project consisted of establishing the parameters for our first-order lag plus deadtime approximation of the third-order systme. The MATLAB code used to accomplish this is presented in Listing 1.

Listing 1: MATLAB Code for First-Order Modelling of A Third-Order System

```
%Third-order system
%
% Description: Simulation of a third order
% process using Simulink.
%
%
% Date: February 7, 2012
% Author: David Collins
%
clc;close all;clear all
%
%--- Constants ------------------
kp1 = 1;
kp2 = 1;
kp3 = 1;
tau1 = 1;
```

```
tau2 = 2;
tau3 = 3;
% --- Denominator coefficients of simplified transfer function
    ---
kp = kp1*kp2*kp3;
c0 = 1;
c1 = tau1 + tau2 + tau3;
c2 = tau1*tau2 + tau1*tau3 + tau2*tau3;
c3 = tau1*tau2*tau3;
% --- First-order model ----------
kp = 1;
tau = 4.39; % time constant
td = 2.05; % dead time
% --- Actual program -------------
sim('third_order_process_w_model');
% --- Plotting ------------------
% --- Third-order process -------
figure;
plot(t,c);
grid;
xlabel('Time (s)');
ylabel('Measurements');
title('Unit-Step Response Of A Third-Order Process');
% --- First-order approximation -
figure;
plot(t,c);
hold on;
plot(t,c1);
grid;
xlabel('Time (s)');
ylabel('Measurements');
title('First-Order Approximation of a Third-Order Process');
```

B.2 PID Controller

Listing 2: MATLAB Code for PID Controllers

```
%Analogue controllers
%
% Description: Proportional (P), Proportional-integral
%   (PI) and Proportional-integral-derivative (PID)
%   controllers
%
% Date: February 21, 2012
% Author: David Collins
%
kp = 1;
tau = 4.39; % time constant
td = 2.05; % dead time/delay time
```

```
% Proportional control
% Note: there is no need to
% determine the integral time and derivative time
% since they are not relevant in this case.
kc_p = tau / td;

% Proportional + integral control
kc_pi = 0.9 * tau / td;
Ti_pi = 3.33 * td;
ki_pi = kc_pi/Ti_pi;

% Proportional + integral + derivative (PID) control
kc = 1.2 * tau / td
Ti = 2*td;
ki = kc/Ti
Td = .5*td;
kd = kc*Td

sim('pid_controller');

figure();

plot(t, c);
hold();
plot(t, c1, 'r');
plot(t, c2, 'g');
grid();
title('Comparison of P, PI, and PID controllers');
xlabel('t');
ylabel('Measurements');
legend('Proportional only', 'Proportional-integral', 'PID
    control');
```

B.3 Plotting & Performance Metrics

The system outputs were plotted in Asymptote.[1] This facilitated more accurate annotation than MATLAB, as well as reliable measurement of performance metrics (Listing 3).

Listing 3: Plotting and Performance Metrics in Asymptote

```
/*
  Description: Plotting and Performance Metrics
  Author: David Collins
  Date: March 2012
*/
```

[1] See `http://asymptote.sourceforge.net/`.

```
// Import necessary modules
import graph;
import geometry;
import interpolate;
// Set output format
settings.outformat = "pdf";
// Image size
size(9cm, 7.5cm, IgnoreAspect);
// Read output values from CSV file
string filename = "controller-p.csv";
file myfile = input(filename);
pair p[] = myfile;
// Define line styles
pen hlines = gray+dashed;
pen dpen = black+linewidth(1.2pt);
// Define path/curve based on values in file
real f(pair A) { return A.x; }
real g(pair A) { return A.y; }
real times[] = map(f, p);
real measurements[] = map(g, p);
real s(real) = fspline(times, measurements);
path p1 = graph(s, min(times), max(times));
// Performance metrics
real cmax = max(p1).y;
real css = s(max(times));
real t1 = point(p1, times(p1, (0,css*.1))[0]).x;
real t2 = point(p1, times(p1, (0,css*.9))[0]).x;
real Tp = point(p1, times(p1, (0,cmax))[0]).x; // peak time
real Ts = point(p1, times(p1, (0,css*.98))[2]).x; // settling
    time
real Tr = t2 - t1;
write(cmax);
write(css);
write(Tr);
write(Ts);
write(Tp);
// Draw curve and axes
draw(p1, dpen);
xaxis("Time (s)", LeftTicks(beginlabel=false, step=0, Step=10)
    , Arrow);
yaxis(Label("Measurements", align=6*W), LeftTicks(beginlabel=
    false, Step=1, step=0.5), Arrow);
// Annotations
xtick("$T_p$", Tp);
xtick("$T_s$", Ts);
ytick(Label("$0.1 c_{ss}$", align=W), 0.1*css);
ytick(Label("$0.9 c_{ss}$", align=W), 0.9*css);
ytick(Label("$c_{ss}$", align=W), css);
ytick("$c_{max}$", cmax);
arrow("$c_{ss} \approx .68$", (max(times),css), SW);
```

```
xequals(t1, 0, 0.1*css, hlines);
xequals(t2, 0, 0.9*css, hlines);
xequals(Tp, 0, cmax, hlines);
xequals(Ts, 0, 0.98*css, hlines);
yequals(0.1*css, 0, t1, hlines);
yequals(0.9*css, 0, t2, hlines);
yequals(css, hlines);
yequals(cmax, 0, Tp, hlines);
yequals(0.98*css, 0, Ts, hlines);
yequals(1, hlines);
distance(rotate(-90)*Label("Overshoot", align=E), (Tp, css), (
    Tp, cmax));
distance(Label("$T_r$", align=S), (t1, 0), (t2, 0));
distance(rotate(-90)*Label(minipage("\scriptsize{large steady-
    state error}", 50pt), align=W), (27,css), (27,1));
```

The metrics can be determined partly by visual inspection. It is easier to use software to do this however. In this case, the script shown in Listing 3 was used.

C C. Disturbance Rejection

As mentioned in Section 2.2, one of the primary reasons for implementing controllers is to counteract the effects of *disturbances* on a process. Consider the system output shown in Figure 14, for example. The set-point in this case is simply 1, and the output initially follows the set-point. However, at time $t = 27$s, there is a system disturbance — after which the output no longer tracks the output.

To counteract this disturbance, feedback and a controller are required. The block diagram then becomes that shown in Figure 6b, and the output is shown in Figure 15. Due to the feedback and controller, the system quickly compensates for the disturbance.

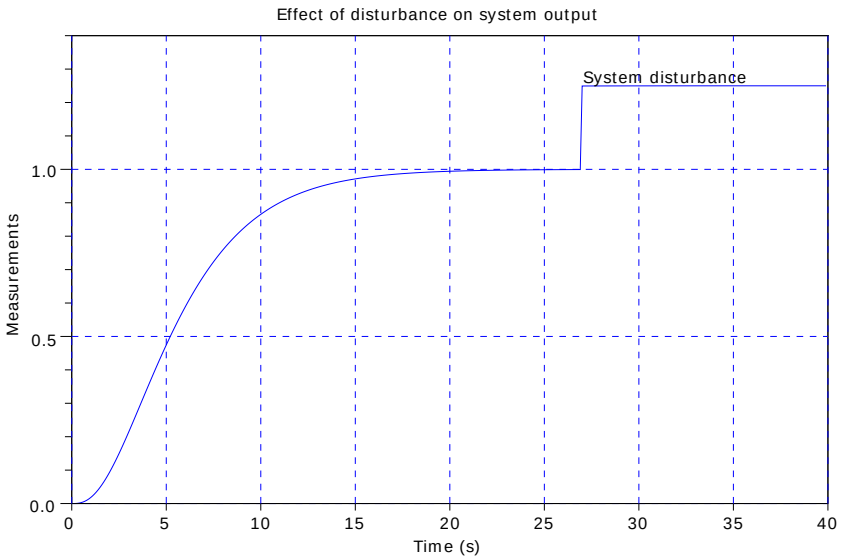


Figure 14: Disturbance with No Feedback

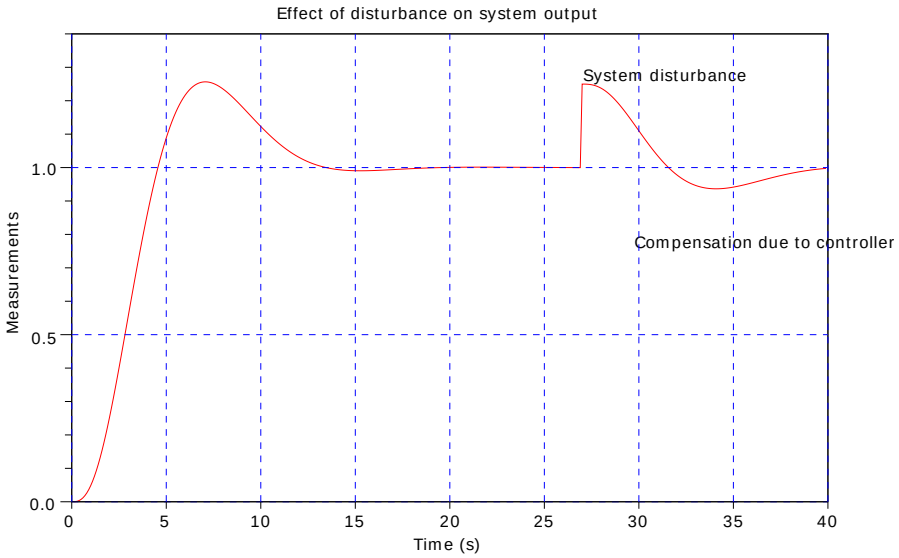


Figure 15: Disturbance-rejection Due to Feedback and Controller