Discrete Time Control Implementation

Riley Kenyon

May 7th, 2019

Abstract

This report describes indirect controller design and how to simulate and implement a continuous time controller digitally. The National Instruments myRIO is used with methods of emulation to enact a PI controller with hardware and software to obtain zero steady-state error.

1 Introduction

The RRC circuit (shown in figure 1) previously used as a first-order system in proportional control with the myRIO can be represented in a control loop described in figure 2. The plant G which has an open-loop transfer function described in equation 1 with $R = 5k\Omega$, and $C = 2.2\mu F$. Under realizable proportional control, the step response of the RRC improves with increased proportional gain K. It is important to note that in implementation, the controller cannot output a voltage higher than the max output. In order to

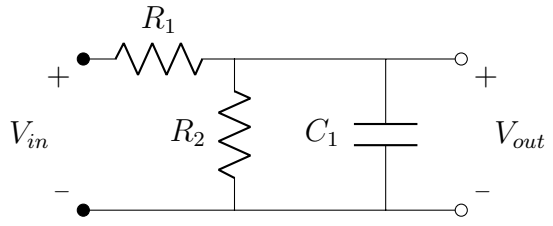

Figure 1: RRC Circuit - a simple first order system representation

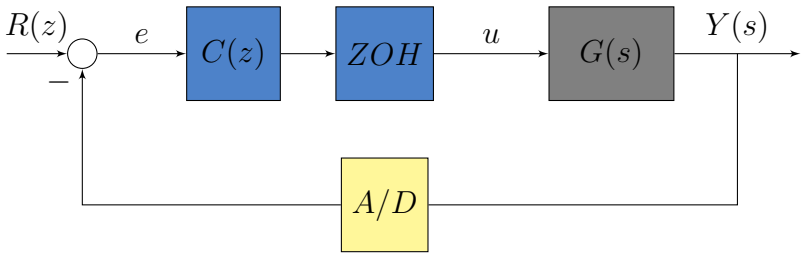

Figure 2: Closed Loop Block Diagram of signal conversion between myRIO(Blue) and input/output of circuit (Gray) using an digital to analog converter(D/A) and an analog to digital converter (A/D).

avoid saturation, the gain of the controller has a physical limit or it becomes non-linear and the control theory can become misaligned. For pure proportional control there still exists a problem in implementation. Although the system perform better (better rise time), the problem of steady-state error still exists. There are two methods to eliminate steady state error, either with infinite gain (perfect reference tracking), or with an internal model. For a step input, this would require a pole at the origin (an integrator). To achieve this, a PI controller will be implemented on the myRIO to obtain better rise time than the open-loop system and achieve zero steady-state error.

$$G(s) = \hat{k}\frac{\sigma}{s+\sigma} = 0.5\frac{2/RC}{s+2/RC}, \tag{1}$$

2 PI Controller Design

In proportional-integral control, the controller $C(s)$ can be described with a proportional and integral constant, as seen in equation 2

$$C(s) = K_p + \frac{K_I}{s} = K_p\frac{s+K_I/K_P}{s}, \tag{2}$$

where K_I is the integral constant and K_P is the proportional constant. For implementing this controller, indirect design is used by designing a continuous time controller $C(s)$ and then implementing it later as a discrete time controller $C(z)$ by using emulation. The first tool used to determine the controller is to look at the zero-pole-gain (zpk) form of the closed-loop transfer function. The controller has a zero at $s = -K_I/K_P$ and a pure integrator.

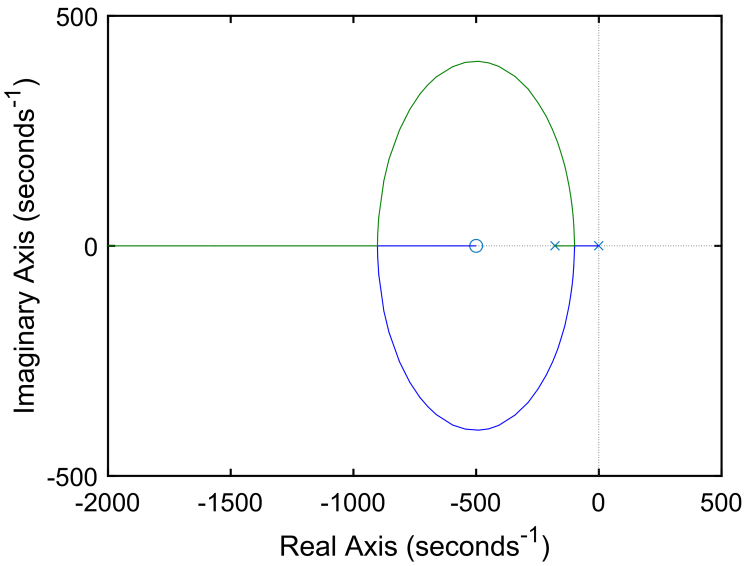

Figure 3: Root locus of the transfer function (controller with plant) with K_P pulled out to determine acceptable gain for the system.

The plant (RRC circuit) has a pole at $s = -2/RC = 181.81$. In order to increase the rise time, the zero needs to be moved to the left. Or have a greater K_I than K_P. As seen in equation 2, by making $K_I > K_P$, the zero will move to the left significantly.

$$H(z) = \frac{C(z)G(z)}{1 + C(z)G(z)}, \tag{3}$$

By keeping the transfer function of the controller with K_P pulled out, the ratio of K_I/K_P can be set and then multiplied with the transfer function of the plant. Root locus analyzes the system as a function of proportional gain in closed loop. Setting the ratio of the two constant, desired performance can be achieved by changing K_P to the value indicated with root locus, and adjusting K_I to compensate in the ratio. For the RRC circuit, the objective is not to overshoot more than one time. Looking at the figure 3, selecting a value of $K_P = 10$ and a ratio of $K_I/K_P = 500$ yield certain characteristics making it theoretically faster than the open loop configuration with a single overshoot.

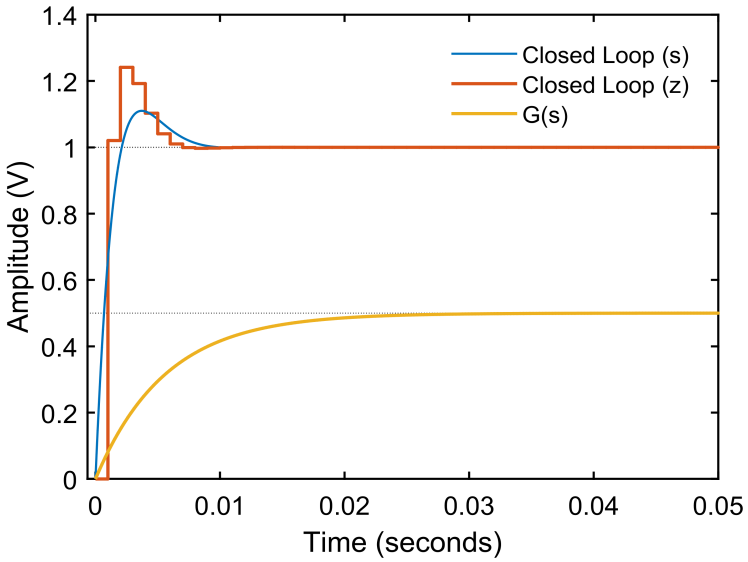

Figure 4: Step response of open-loop and closed loop system with PI controller.

2.1 Step Response Simulation

After selecting a gain with desired performance characteristics from the root locus, the step responses can be simulated. For simulation, the open-loop system was compared to the PI controlled closed loop in the continuous time domain. This verified that the new PI controller conformed to the design restrictions of limited overshoot and better rise-time. It was also useful to see how the controller would be implemented digitally. For that the *Tustin* method of emulation was used for the controller. This choice was made because Tustin maps the stable region in continuous time domain to the stable region in the discrete time domain, and additionally it was not needed to be excessively fast (a sampling period of 1 ms was selected). Note that for emulation, it was important to see how the controller performed in the digital domain so using matlab's built in 'c2d' function with the parameter of 'tustin' the controller was emulated to get $C(z)$ while the continuous time plant $G(s)$ was combined with a Zero Order Hold (ZOH) to get $G(z)$.

3 Z-Transform

In order to convert to implementable code, the Z-transform was taken of the PI controller. The method of emulation, as described in the previous section was to use Tustin in which s is represented as

$$s = \frac{2}{T}\frac{Z-1}{Z+1}, \tag{4}$$

where T is the sample period. Plugging in this for the PI controller yields

$$C(z) = \frac{2K_P(Z-1) + TK_I(Z+1)}{2(Z-1)} = \frac{(TK_I + 2K_P)Z + (TK_I - 2K_P)}{2Z-2}, \tag{5}$$

and can be further simplified to controller output with respect to error as

$$\frac{U(z)}{E(z)} = \frac{(TK_I + 2K_P) + (TK_I - 2K_P)Z^{-1}}{2 - 2Z^{-1}}. \tag{6}$$

In this form, the controller can be manipulated into a difference equation that is implementable in code on the myRIO. This is because a Z^{-1} is equivalent to a delay. Converting $C(z)$ into a series of analog inputs and outputs results in the difference equation

$$u[k] = \frac{(TK_I + 2K_P)e[k] + (TK_I - 2K_P)e[k-1] + 2u[k-1]}{2}, \tag{7}$$

where u[k] is the current output of the controller (analog-output), and e[k] is the current error between the current value and the reference (analog-input).

4 Implementation (LABVIEW)

The difference equation represented in equation 7 is how to implement controllers through code, which can be done on the myRIO with the use of LabVIEW. Taking a look at some LabVIEW code, the overall loop is a timed loop structure running at a 1 MHz clock speed and 1000 clock ticks per loop which is a 1 kHz sampling rate. The reference to the control loop is a 1.25 Hz square-wave to repeat multiple step inputs to the system. Looking at the main computation section, there is the implementation of the difference

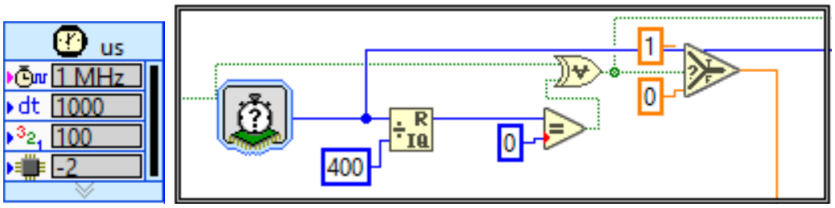

Figure 5: LabVIEW code of experimental setup with timed loop timer configuration (left) and square wave function generator with frequency of 1.25Hz (right)

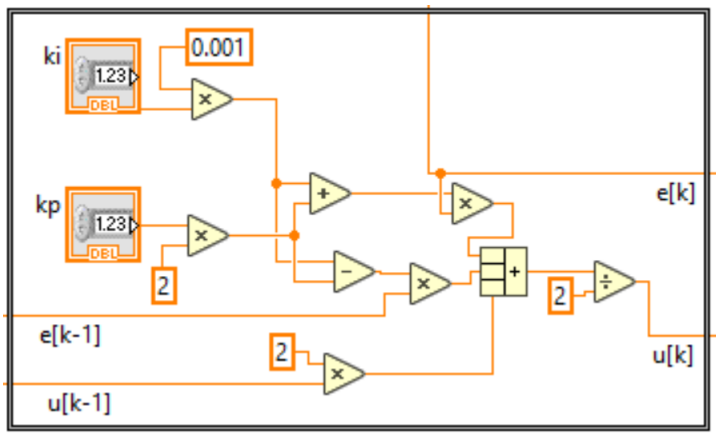

Figure 6: The difference equation written into labVIEW, replicating the Z-transform of the PI controller in terms of input error e[k] and controller output u[k].

equation. The analog input and output are configured to run on the C channel of the myRIO, and read $e[k]$ and output $u[k]$ respectively. In order to obtain previous iterations of the error and controller output, a shift register is used and the previous value is worked into the proper operations to replicate the difference equation.

5 Digital-Controller Results

Looking at the continuous time controller and the emulated version seen in Figure 4, there is a noticeable discrepancy between the two. The step response of the digital controller had more overshoot due to the phase shift as-

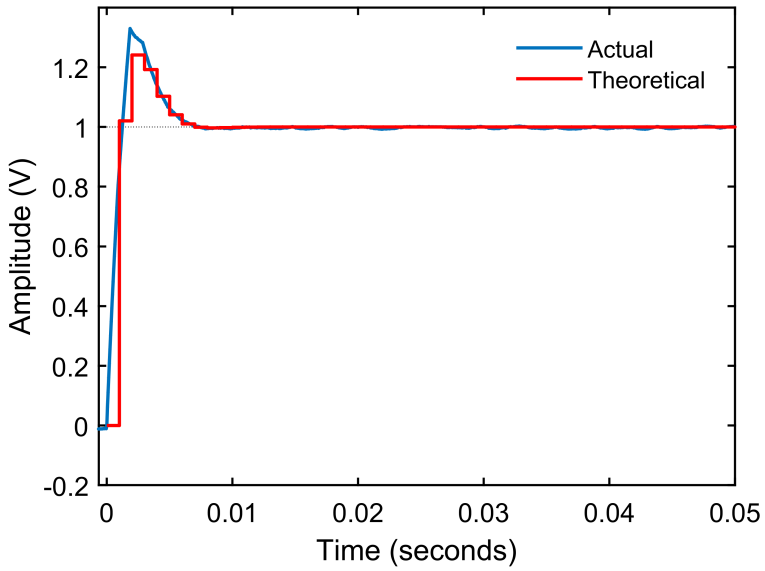

Figure 7: Simulated step response of the RRC circuit under PI control as it compares to the actual step response using the digital controller.

sociated with sampling. Upon implementation in LabVIEW on the myRIO, a sampling rate of 1 ms was used to interpret analog input and deliver an output to the RRC circuit. The simplest way to represent the output of a digital controller to a plant is to use a Zero Order Hold (ZOH), or a digital to analog converter (D/A). To perform simulations of the closed loop system, the ZOH is combined with the plant $G(S)$ to get the Z-transform interpretation of the plant. This representation $G(z)$ is used with the emulated controller $C(z)$ to simulate how the system will react in a digital control setting. The controller was emulated using the Tustin method to ensure stability, and the simulation was compared to the continuous theoretical step response in Figure 4. Looking at how the theoretical model compares to the actual implementation, the step response was recorded on an oscilloscope and compared to the step response of the digital system in Matlab. The overshoot observed on the oscilloscope is slightly greater than the simulated overshoot, however this more closely follows the theoretical step response rather than the step response of the continuous closed loop system.

6 Conclusion

The theory behind the controller and the digital implementation and the of the system matched up nicely. With the Tustin method of emulation, the stable controller was successfully mapped to the digital domain in the stable region. The performance of the controller met the loose standards of limited overshoot (not multiple oscillations) and a rise time of 1.5 ms, performing faster than the open-loop system 12.3 ms. By using a pure integrator, the steady-state error to a step was reduced to zero as seen in Figure 7.

A Code

A.1 Creating Plots

```
%--------------------------------------------------------------------------
%% Industrial Automation Lab 5 - FPGA
% Riley Kenyon 4/30/2019
%% Experimental Data
clear all; clc; close all;
% LabVIEW
data = load('kp10ki5000.mat');
data1 = data.kp5000ki;
data1(:,1) = (data1(:,1) - data1(1,1) - 318)*0.001;

% oscilloscope
exp01 = csvread('experimental_kp10.csv',2,0);

% simulation
R1 = 5100; % ohm
C1 = 2.2e-6; % uF
G = tf(1,[R1*C1 2]); % RRC

%PID controller
kp = 10;
ratio = 500;
Ts = 0.001;
ki = ratio*kp;
```

```
C2 = tf([kp ki],[1 0]);
Gz = c2d(G,Ts,'zoh');
L2 = C2*G;
CL = feedback(L2,1);
Cz = c2d(C2,Ts,'tustin');
CLz = feedback(Cz*Gz,1);

% plotting
figure()
hold on
% plot(data1(:,1),data1(:,2),'LineWidth',1.5);
step(CL)
step(CLz)
step(G)
xlim([-0.00065,0.05]);
legend("CL","CLz","G")

figure()
hold on
plot(exp01(:,1)+0.00065,exp01(:,2),'LineWidth',1.5);
step(CLz)
xlim([-0.00065,0.05]);
legend("Actual","Theoretical")
```

A.2 LabVIEW Code