

Lab Exercise 07 DC motor PI velocity control

Lab 07.1 Objectives

The objectives of this exercise are to:

1. Incorporate many of the hardware and software elements developed previously in this course into an integrated closed-loop control system.
2. Implement a program structure allowing continuous modification of the control parameters without halting the control algorithm.
3. Implement a proportional-integral (PI) velocity control algorithm for the DC motor.

Lab 07.2 Introduction

In this exercise, a closed-loop control system for the DC motor will be developed. This system is similar to actuators used in many types of positioning systems. The primary drive of one axis of an automated machine tool or of one axis of motion of an industrial robot is often a computer-controlled DC motor.

Our system will control the motor speed. The control algorithm will repeatedly compare the actual velocity of the motor V_{act} with the desired reference velocity V_{ref} , and automatically alter the applied control voltage to correct any differences. Although this is not a trivial computer control task, you have developed nearly all the required elements in the preceding six exercises.

The optical encoder (through the FPGA), the D/A converter (connected to the motor amplifier), and the periodic timer interrupt, will be combined to control the DC motor. As in [Lab Exercise 06](#), a separate timer thread will produce an interrupt at the end of each basic time interval (BTI). The ISR will load the `IRQTIMERWRITE` and `IRQTIMERSETTIME` registers to schedule the next BTI, and then call functions to:

1. read the encoder counter and compute the velocity,
2. execute the motor control algorithm, and
3. save the results, as necessary.

The control system will be “table-driven.” That is, the parameters used by the control algorithm (reference speed, system gains, and BTI length) will be kept in a special *table* of values. Through the keypad/LCD, the values of

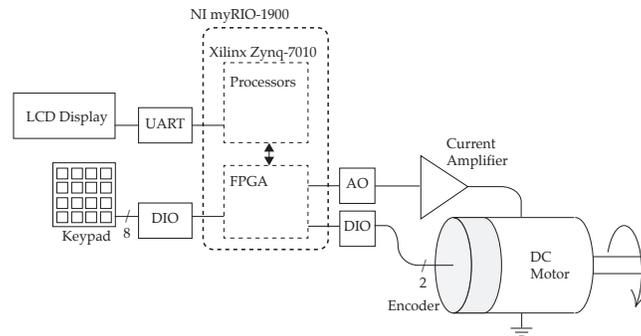


Figure 07.1:

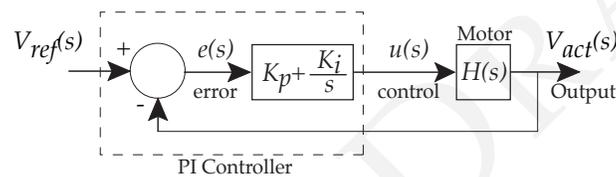


Figure 07.2: continuous representation of the control loop.

the parameters in the table will be altered (interactively) by a “table editor” function called from the main program thread. The *only* tasks of the table editor will be to change the table values in response to commands from the keypad, and to display performance information.

This table-driven structure will allow the program user to change any of the control parameters, at any time, without stopping the execution of the control algorithm. It will *appear* as though two programs, the table editor and the control algorithm, are executing simultaneously.

You will *not* write the table editor. The required function, `ctable2` is described in [Resource 15](#). It has been included in our library, and is automatically linked with your program. Although you will not write this function, it uses the basic keypad/display algorithms that you developed in [Lab Exercises 01, 02 and 03](#). The prototype for `ctable2` is in `ctable2.h`.

The motor will be controlled using a proportional-plus-integral (PI) control law as shown in [Figure 07.2](#). The PI control law relates the error $e(t)$ to the output control signal $u(t)$ using the gain constants K_p and K_i .

Applying Tustin’s method to the continuous controller transfer function

$K_p + \frac{K_i}{s}$, the corresponding discrete transfer function is

$$\frac{U(z)}{E(z)} = \frac{b_0 + b_1 z^{-1}}{a_0 + a_1 z^{-1}}, \quad (07.1)$$

where

$$a_0 = 1, \quad a_1 = -1, \quad b_0 = K_p + \frac{1}{2}K_i T, \quad \text{and} \quad b_1 = -K_p + \frac{1}{2}K_i T. \quad (07.2)$$

where T is the sample time, and the error is

$$e(n) = V_{\text{ref}}(n) - V_{\text{act}}(n)$$

For more on Tustin's method, see [Lecture 06.03.5](#). You will implement the corresponding difference equation using the general-purpose algorithm you developed in [Lab Exercise 06](#).

Lab 07.3 Pre-laboratory preparation

Drawing on your previous work, write two threads to: (1) communicate with the user and (2) control the motor.

Lab 07.3.1 Two threads

Main program thread The main program performs these tasks:

1. Initialize the table editor variables.
2. Set up and enable the timer IRQ interrupt (as in [Lab Exercise 06](#)).
3. As in [Lab Exercise 06](#), register the Timer Thread and create the thread to catch the Timer Interrupt. In this lab, the Timer Thread will gain access to the table data through a pointer. Modify the Timer Thread resource to include a pointer to the table. For example,

```
typedef struct {
    NiFpga_IrqContext irqContext; // context
    table *a_table;             // table
    NiFpga_Bool irqThreadRdy;    // ready flag
} ThreadResource;
```

4. Call the table editor. The table should contain six values, labeled as shown:

```

V_ref:  rpm      {edit}
V_act:  rpm      {show}
VDAout: mV      {show}
Kp:    V-s/r    {edit}
Ki:    V/r      {edit}
BTI:   ms       {edit}

```

All of the table edit values should be initialized to zero, except for the BTI length, which should be 5 ms. Note the units.

After the main program calls the table editor, the user may edit and view the table values whenever desired.

- When the table editor exits, signal the Timer Thread to terminate. Wait for it to terminate.

Timer thread – ISR At the beginning of the starting function, declare convenient names for the table entries from the table pointer:

```

double *vref = &((threadResource->a_table+0)->value);
double *vact = &((threadResource->a_table+1)->value);
// ...etc.

```

As in [Lab Exercise 06](#), the Timer Thread includes a main loop timed by the IRQ, and is terminated only by its ready flag.

Before the loop begins:

- initialize the analog I/O, and set the motor voltage to zero, using `Aio_Write` (as is [Lab Exercise 06](#)).
- Set up the encoder counter interface (as in [Lab Exercise 04](#)).

Each time through the loop of the it should:

- Get ready for the next interrupt by: waiting for the IRQ to assert, writing the Timer Write Register, and writing TRUE to the Timer Set Time Register.
- Call `vel`, from [Lab Exercise 04](#), to measure the velocity of the motor.
- Compute the current coefficients (a's and b's) for the PI control law from the current values of K_p and K_i . See [Equation 07.1](#). Update the values of the `biquad` structure.
- Compute the current error $V_{ref} - V_{act}$.
- Call `cascade` to compute the control value from the current error using the difference equation for PI control law. *Important*: limit the computed control value to the range $[-7.5, 7.5]$ V.

6. Send the control value to the D/A converter C00 using `Aio_Write`.
7. Change the show values in the table to reflect the current conditions of the controller.
8. Save the results of this BTI for later analysis. (See below.)
9. Acknowledge the interrupt.

Lab 07.3.2 Functions

`cascade` – The `cascade` function, called once, from the ISR, during each BTI, implements the general-purpose linear difference equation algorithm from Lab 06. For this lab use the *same C code* that you used in [Lab Exercise 04](#). In this case, the number of biquad sections will be one.

Note that, as in [Lab Exercise 06](#), all calculations should be made in (**double**) floating-point arithmetic.

`vel` – Use the `vel` function, developed in [Lab Exercise 04](#), to read the encoder counter and estimate the angular velocity in units of BDI/BTI.

Lab 07.3.3 Saving the Responses

A convenient method of saving the data is to define data arrays in the ISR for both the velocity and the torque. Then an auto-incremented index variable is used to store the data in the arrays during each BTI. Increment the index as needed, stopping when index reaches the length of the arrays. A convenient length would be 250 points each.

Since our program runs continuously, you may wish to save the response whenever the reference velocity is changed. This is easily accomplished by checking to see if the reference velocity has changed since the last BTI, and resetting the index to zero if it has. Since the index is then less than the length of the arrays, the arrays will be refilled. This is equivalent to recording the response to a step input in the reference velocity.

In addition, when the ISR resets the index to zero, save the previous value of the reference velocity. That value, along with other system parameters, will be used in MATLAB to predict the theoretical model response.

After the main loop terminates, but while still in the Timer Thread, write the results to the `Lab7.mat` file. The results should include:

1. your name (string),
2. the actual velocity array (rad/s),
3. the torque array (N-m),
4. the current *and* the last, previous reference velocities (rad/s),

Table 07.1: the base set of controller parameters.

V_{ref}	± 200	rpm
BTI length	5	ms
K_p	0.1	V-s/rad
K_i	2.0	V/rad

5. K_p (V-s/rad),
6. K_i (V/rad), and
7. BTI length (s).

Use the same methods as [Lab Exercise 04](#) and [Lab Exercise 06](#) to bring the `Lab7.mat` file to Matlab.

Lab 07.4 Laboratory Procedure

1. Test and debug your program.
For debugging purposes, use the *base set* of controller parameters in [Table 07.1](#).
2. While the motor is at steady-state speed, *gently* apply a *steady* load torque to the motor shaft. What are the responses of the actual speed and control voltage? Explain.
3. Beginning with the base set of parameters, explore the effect of varying the proportional gain K_p on the transient response. Try small (0.05) and large (0.2) values of K_p . What are the effects on the oscillation frequency and on the damping? Explain in terms of the transfer function parameters.
4. Beginning with the base set of parameters, explore the effect of varying the integral gain K_i on the transient responses. Try small (1) and large (10) values of K_i . What are the effects on the oscillation frequency and on the damping? Explain in terms of the transfer function parameters.
5. Finally, using the base set of parameters, record the control torque and actual velocity responses for a step change in the reference velocity that starts from -200 rpm and goes to $+200$ rpm.

In Matlab, compare these experimental responses with the analytical responses for the continuous system approximation.

The theoretical responses can be calculated using the (appropriately scaled) Matlab `step` command. The analysis should be plotted over the experimental responses. Use the `subplot` command to place

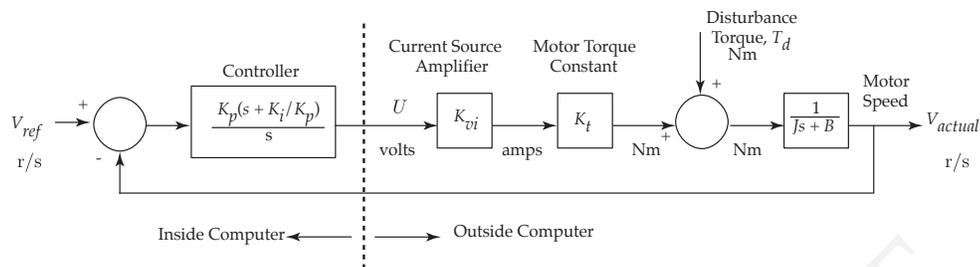


Figure 07.3: continuous version of the full control loop.

Table 07.2:

K_{vi}	Current Source Amplifier Gain	0.41	A/volt
K_t	Motor Torque Constant	0.11	N-m/A
J	Inertia in conventional units	3.8×10^{-4}	N-m-s ² /r
K_p	Proportional Gain	—	V-s/r
K_i	Integral Gain	—	V/r

both the control value and the measured velocity plots on the same page.

What do you conclude?

Lab 07.5 DC Motor Controller Model

The model in [Figure 07.3](#) is the continuous approximation of the actually discrete control of the DC motor.

For accuracy of the approximation, we need the following:

1. the sampling frequency is much larger than the natural frequency of the system and
2. time delays caused by computation are insignificant and
3. the control value does not saturate and
4. the mechanical damping B is small in comparison with the effects of the proportional term K_p in the controller.

Parameter values are shown in [Table 07.2](#).

The following transfer functions can be obtained from the block diagram:

$$\frac{V_{\text{act}}(s)}{V_{\text{ref}}(s)} = \frac{\tau s + 1}{\frac{s^2}{\omega_n^2} + \frac{2\zeta}{\omega_n} s + 1}, \quad (07.3)$$

$$\frac{V_{\text{act}}(s)}{T_d(s)} = \frac{sK_d}{\frac{s^2}{\omega_n^2} + \frac{2\zeta}{\omega_n} s + 1}, \text{ and} \quad (07.4)$$

$$\frac{U(s)}{V_{\text{ref}}(s)} = \frac{sK_u(\tau s + 1)}{\frac{s^2}{\omega_n^2} + \frac{2\zeta}{\omega_n} s + 1}. \quad (07.5)$$

Let $K = K_{vi}K_t$ N-m/V; then the following values can be used to model the system:

numerator values: $\tau = \frac{K_p}{K_i} \quad \text{s},$

$$K_d = \frac{1}{K_i K} \quad \text{rad/N-m},$$

$$K_u = \frac{1}{K} \quad \text{V-s}^2/\text{rad}$$

natural frequency: $\omega_n = \sqrt{\frac{K_i K}{J}} \quad \text{rad/s}$

damping ratio: $\zeta = \frac{K_p}{2} \sqrt{\frac{K}{JK_i}}.$