GEORGIA INSTITUTE OF TECHNOLOGY
SCHOOL of ELECTRICAL and COMPUTER ENGINEERING

ECE 4550 — Control System Design — Fall 2019

Lab #6: DC Motor Position Control

Contents

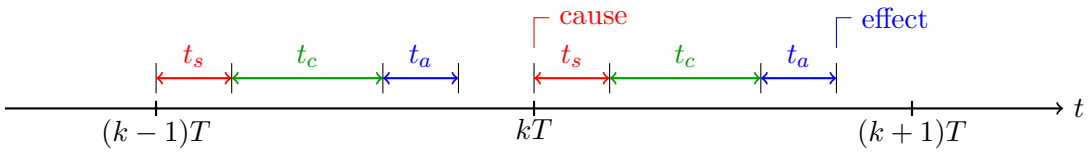
1 Background Material

1.1 Introductory Comments

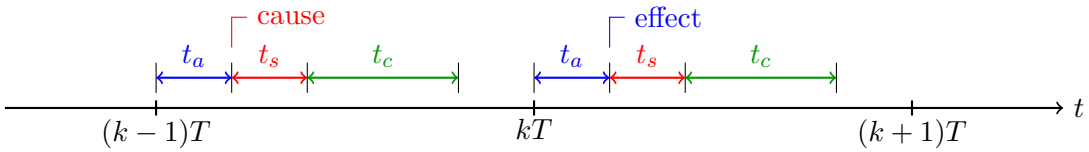
The objective of this lab is to implement a state-space integral control algorithm on a microcontroller, in order to perform position control. We will leverage the reduced-order physics-based model of the DC motor plant; the coefficient matrices of this model will be needed for controller design and implementation, so we will use approximate values inferred from measured input-output data. We will utilize plant model knowledge and the computational capability of the microcontroller to perform the numerical processing required by the state-space integral control algorithm, including the real-time approximate solution of differential equations governing the estimator and regulator subsystems of the controller; integrator anti-windup compensation will be used to avoid the poor transient response that can arise when actuator saturation occurs.

Two possible code flows are displayed in Figure 1a and Figure 1b. Feedback controllers need to read from the sensor (which takes time t_s), compute a corrective action (which takes time t_c), and write to the actuator (which takes time t_a). Typically, $t_s + t_a$ is extremely small (e.g. 100 ns), whereas t_c may be large compared to the timer period T (e.g. $0.5T$ or 500 μs). Since the actuator write depends on the sensor read, but is not available until after a lengthy calculation has taken place, the intuitive code flow has the disadvantage that sensor reads and actuator writes are not synchronized in time. Systems that change on the interior of the control interval are more difficult to model and simulate; moreover, the computation time may vary from control interval to control

interval, leading to non-regular timing in such systems. Thus, the modified code flow is preferred, since it has the advantage that sensor reads and actuator writes will be synchronized in time. This desirable time synchronization is achieved, however, by delaying the actuator write until the timer event that follows controller computations.



(a) Intuitive code flow (sense-compute-actuate) degrades synchronous timing.



(b) Modified code flow (actuate-sense-compute) enhances synchronous timing.

Figure 1: Two possible interrupt-based code flows for control systems ($t_s + t_a \ll t_c$).

1.2 Plant Modeling

We typically require two different models of the same DC motor plant. The *simulation model* has three state variables (position, speed and current) whereas the *design model* has only two state variables (position and speed). We use the higher accuracy simulation model to represent the plant for simulation purposes, but we use the lower complexity design model to represent the plant for controller design purposes. We have this option since the dynamics being neglected by the design model—the voltage to current transient response—is both stable and fast.

You should review both the simulation model and the design model from posted lecture notes. The simulation model will appear explicitly in posted simulation code, whereas the design model will be used implicitly (gain calculations) and explicitly (estimator model) for controller implementation. In order to prepare for the controller design calculations that follow, recall that the design model may be expressed in the form

$$\dot{x}(t) = \begin{bmatrix} 0 & 1 \\ 0 & -\alpha \end{bmatrix} x(t) + \begin{bmatrix} 0 \\ \beta \end{bmatrix} (u(t) - w(t))$$
$$y(t) = \begin{bmatrix} 1 & 0 \end{bmatrix} x(t) + v(t)$$

where α and β are load-dependent constant parameters to be determined using a parameter identification procedure. The coefficient matrices A, B and C are the point of departure for design.

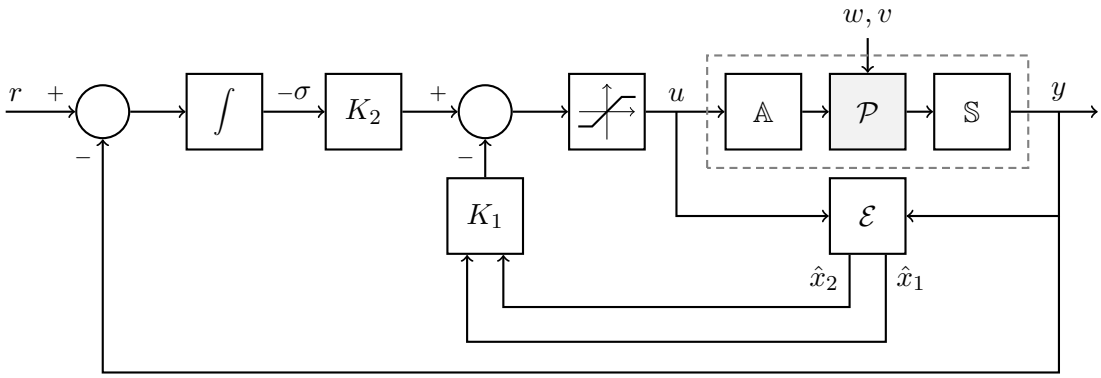
1.3 Controller Design

The general theory of state-space integral control has been summarized in some notes posted to canvas, so *you should have read through those notes prior to attempting this lab*. Critical steps in applying those notes to the control system design problem at hand are highlighted below. Following the design-by-emulation concept, often referred to as "indirect" digital design, we will first design a suitable *continuous-time* position controller, and then discretize that controller in a final step in order to obtain a corresponding *discrete-time* position controller.

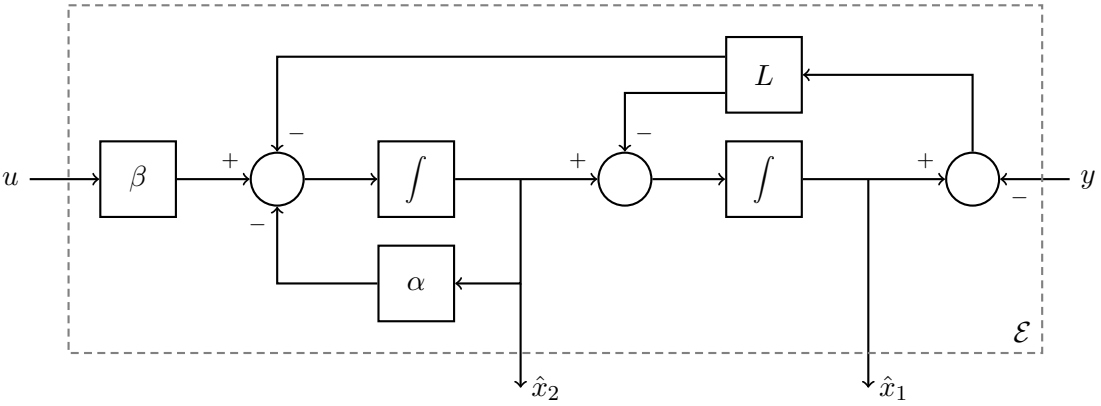
The controller is implemented in C with a GPIO enable signal (Labs 1–2), it operates periodically with period established by a timer interrupt (Lab 3), the input signal of the plant is a pulse-width modulated voltage and the output signal of the plant consists of quadrature encoder pulses (Lab 5). Although not used in this lab, analog-to-digital conversion (Lab 4) is often employed to enhance the quality of torque, speed or position controllers, by adding a current feedback loop. Hence, this lab integrates almost all that we have learned so far this semester.

1.3.1 Controller Structure

The architecture of the controller we're using is shown in Figure 2. Shaded block $\mathcal{P}$ represents DC motor plant *hardware*, which consists of an actuator (power converter), a motion system (rotor and load) and a sensor (optical encoder). The disturbance w (external torque disturbance) and noise v (position sensing noise) act directly on the hardware level. All unshaded blocks represent embedded *software*. Block $\mathbb{A}$ represents the actuator interface, which receives voltage signal u and generates corresponding actuator signals via the PWM peripheral. Block $\mathbb{S}$ represents the sensor interface, which receives sensor signals and generates corresponding position signal y via the QEP peripheral. The ensemble $(\mathbb{A}, \mathcal{P}, \mathbb{S})$ constitutes a generalized plant with SI unit connection ports.



(a) Overall system emphasizing the interaction of the plant and regulator subsystem.



(b) Estimator subsystem responsible for generating estimates of plant state variables.

Figure 2: State-space integral control architecture for position control applications.

The saturation block accounts for the limits on applied motor voltage inherited as a consequence of the fixed power supply voltage. In the inner loop of the controller, the estimator—the block labeled $\mathcal{E}$—generates the scalar signals $\hat{x}_1$ and $\hat{x}_2$, which are estimates of the measured position y

and the unmeasured speed $\dot{y}$, respectively. The estimator uses the pre-saturated command voltage as a proxy for the average voltage actually applied to the motor, in order to eliminate the need for any direct measurement of motor voltage. In the outer loop of the controller, the scalar signal σ is generated by integrating the error between the measured position y and its commanded value r. The gain matrices of the estimator (L) and the regulator (K_1 and K_2) are selected so as to ensure stabilization of the overall system, and the error integrator guarantees effectively zero steady-state error between the measured position and its commanded value.

1.3.2 Analog Controller Design

From the design model shown above, it is clear that

$$A = \begin{bmatrix} 0 & 1 \\ 0 & -\alpha \end{bmatrix}, \quad B = \begin{bmatrix} 0 \\ \beta \end{bmatrix}, \quad C = \begin{bmatrix} 1 & 0 \end{bmatrix}.$$

The stability of the estimator subsystem is determined by the matrix

$$A - LC = \begin{bmatrix} 0 & 1 \\ 0 & -\alpha \end{bmatrix} - \begin{bmatrix} L_1 \\ L_2 \end{bmatrix} \begin{bmatrix} 1 & 0 \end{bmatrix} = \begin{bmatrix} -L_1 & 1 \\ -L_2 & -\alpha \end{bmatrix}.$$

If we want the actual characteristic polynomial

$$\det\left(sI - (A - LC)\right) = s^2 + (L_1 + \alpha)\, s + (L_1\alpha + L_2)$$

to match the desired characteristic polynomial

$$(s + \lambda_e)^2 = s^2 + 2\lambda_e s + \lambda_e^2$$

having two roots at $s = -\lambda_e$, then the estimator gains must be

$$L_1 = 2\lambda_e - \alpha, \quad L_2 = \lambda_e^2 - 2\alpha\lambda_e + \alpha^2.$$

It is possible to solve the estimator eigenvalue assignment problem in this case because our sensor choice results in an *observable* system. As expected, L_1 and L_2 depend on plant parameters (α, β) and design parameter λ_e. A large λ_e would lead to a fast estimator, but also to large L_1 and L_2 which could result in problems relating to sensor noise amplification; therefore, selection of λ_e involves a trade-off between rate of response and signal/noise ratio.

The regulator subsystem is modeled by the matrices

$$\mathcal{A} = \begin{bmatrix} 0 & 1 & 0 \\ 0 & -\alpha & 0 \\ 1 & 0 & 0 \end{bmatrix}, \quad \mathcal{B} = \begin{bmatrix} 0 \\ \beta \\ 0 \end{bmatrix}, \quad \mathcal{K} = \begin{bmatrix} K_{11} & K_{12} & K_2 \end{bmatrix}$$

and its stability is determined by the matrix

$$\mathcal{A} - \mathcal{B}\mathcal{K} = \begin{bmatrix} 0 & 1 & 0 \\ 0 & -\alpha & 0 \\ 1 & 0 & 0 \end{bmatrix} - \begin{bmatrix} 0 \\ \beta \\ 0 \end{bmatrix} \begin{bmatrix} K_{11} & K_{12} & K_2 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 \\ -\beta K_{11} & -\alpha - \beta K_{12} & -\beta K_2 \\ 1 & 0 & 0 \end{bmatrix}.$$

If we want the actual characteristic polynomial

$$\det\left(sI - (\mathcal{A} - \mathcal{B}\mathcal{K})\right) = s^3 + (\alpha + \beta K_{12})\, s^2 + (\beta K_{11})\, s + (\beta K_2)$$

to match the desired characteristic polynomial

$$(s+\lambda_r)^3 = s^3 + 3\lambda_r s^2 + 3\lambda_r^2 s + \lambda_r^3$$

having three roots at $s = -\lambda_r$, then the regulator gains must be

$$K_{11} = \frac{1}{\beta}3\lambda_r^2, \quad K_{12} = \frac{1}{\beta}(3\lambda_r - \alpha), \quad K_2 = \frac{1}{\beta}\lambda_r^3.$$

It is possible to solve the regulator eigenvalue assignment problem in this case because our actuator choice results in a *controllable* system. As expected, K_{11}, K_{12} and K_2 depend on plant parameters (α, β) and design parameter λ_r. A large λ_r would lead to a fast regulator, but also to large K_{11}, K_{12} and K_2 which could result in problems relating to actuator saturation limits; therefore, selection of λ_r involves a trade-off between rate of response and control effort.[1]

Once the gain matrices have been determined, all that would remain for analog control implementation would be to construct an op-amp circuit capable of implementing the feedback law based on state-space integral control; in vector form this feedback law is

$$\begin{aligned} u(t) &= -K_1\hat{x}(t) - K_2\sigma(t) \\ \dot{\hat{x}}(t) &= A\hat{x}(t) + Bu(t) - L\left(C\hat{x}(t) - y(t)\right) \\ \dot{\sigma}(t) &= y(t) - r(t), \end{aligned}$$

whereas substitution of (A, B, C) and (L, K_1, K_2) results in the scalar form

$$\begin{aligned} u(t) &= -K_{11}\hat{x}_1(t) - K_{12}\hat{x}_2(t) - K_2\sigma(t) \\ \dot{\hat{x}}_1(t) &= \hat{x}_2(t) - L_1\left(\hat{x}_1(t) - y(t)\right) \\ \dot{\hat{x}}_2(t) &= -\alpha\hat{x}_2(t) + \beta u(t) - L_2\left(\hat{x}_1(t) - y(t)\right) \\ \dot{\sigma}(t) &= y(t) - r(t). \end{aligned}$$

For many reasons, we prefer to approximate this control algorithm on a microcontroller.

1.3.3 Digital Controller Design

By definition, analog controllers are implemented by analog circuits and digital controllers are implemented by digital circuits. Analog controllers provide corrective actions on a continuous-time basis whereas digital controllers provide corrective actions on a discrete-time basis. If we want to implement an existing analog controller on an embedded microcontroller chip, the analog controller must first be discretized to generate a corresponding digital controller. It is not possible to perform *controller discretization* in an exact way, since a discrete-time system response cannot match a continuous-time system response on a continuous-time basis.[2]

For this lab, we will be content to employ the simplest of all discretization schemes, the forward Euler method. This simple method relies on the approximation

$$\dot{x}(kT) \approx \frac{x(kT+T) - x(kT)}{T}$$

[1] A large λ_r could also cause instability, since our controller design neglects current as a state variable.

[2] It is possible to perform *plant discretization* in an exact way, since a continuous-time system response can match a discrete-time system response at sampling instants. This concept is exploited in so-called "direct" digital design.

where x denotes an arbitrary signal, T denotes a fixed sampling period and k represents a discrete-time index. Using this method to approximate derivatives by differences, the previously designed analog controller gets replaced by the corresponding digital controller[3]

$$\begin{aligned} u[k] &= -K_1\hat{x}[k] - K_2\sigma[k] \\ \hat{x}[k+1] &= \hat{x}[k] + T\left(A\hat{x}[k] + Bu[k] - L\left(C\hat{x}[k] - y[k]\right)\right) \\ \sigma[k+1] &= \sigma[k] + T\left(y[k] - r[k]\right). \end{aligned}$$

This digital controller is not equivalent to the analog controller from which it was derived, for the reasons stated above. However, this digital controller will do a good job of emulating the desired analog controller behavior if T is sufficiently small.

To complete the design of our digital controller, we must substitute our plant model matrices (A, B, C) and our selected gain matrices (L, K_1, K_2) into the above equations, we must add the controller saturation effect, and we must exhibit the one-cycle delay introduced by our ISR programming philosophy. The result of these final steps is the computed controller output signal

$$u^*[k] = -K_{11}\hat{x}_1[k-1] - K_{12}\hat{x}_2[k-1] - K_2\sigma[k-1],$$

the saturated controller output signal

$$u[k] = \begin{cases} +U_{\max} & \text{, if } u^*[k] > +U_{\max} \\ -U_{\max} & \text{, if } u^*[k] < -U_{\max} \\ u^*[k] & \text{, otherwise} \end{cases}$$

and the controller state variable update equations

$$\begin{aligned} \hat{x}_1[k+1] &= \hat{x}_1[k] + T\hat{x}_2[k] - TL_1\left(\hat{x}_1[k] - y[k]\right) \\ \hat{x}_2[k+1] &= \hat{x}_2[k] - T\alpha\hat{x}_2[k] + T\beta u[k] - TL_2\left(\hat{x}_1[k] - y[k]\right) \\ \sigma[k+1] &= \sigma[k] + T\left(y[k] - r[k]\right). \end{aligned}$$

These equations have been presented in scalar form to simplify their programming in C.

1.3.4 Anti-Windup Compensation

The implementation of state-space integral control as previously described would not perform well if actuator saturation were to occur. The problem, known as integrator windup, has been described in some notes posted to canvas; generally speaking, integration of output error is not appropriate during time intervals on which the actuator is saturated, since that would lead to an increasing integrator output without resulting in a larger corrective influence.

One approach to anti-windup compensation is to use conditional integration. With this approach, pure integration is performed whenever saturation is not occurring whereas no integration at all is performed whenever saturation is occurring. Assuming that the control signal will be pre-saturated in code so as to avoid triggering saturation within the actuator itself, the synthesis of the control signal involves three steps: (i) compute the desired control signal $u^*(t)$ in the usual way; (ii) pre-saturate $u^*(t)$ to obtain a value for $u(t)$ that is compatible with actuator saturation limits; (iii) compute the integrator output $\sigma(t)$ by assigning either a zero value or the output error

[3]The square bracket notation is used to distinguish discrete-time signals from continuous-time signals; the index inside the brackets indicates the control cycle during which the specified signal value is valid.

value to the integrator input according to the presence or absence of saturation. Mathematically, this three-step procedure is summarized by

$$u^*(t) = -K_{11}\hat{x}_1(t) - K_{12}\hat{x}_2(t) - K_2\sigma(t)$$
$$u(t) = \begin{cases} +U_{\max} & , \text{ if } u^*(t) > +U_{\max} \\ -U_{\max} & , \text{ if } u^*(t) < -U_{\max} \\ u^*(t) & , \text{ otherwise} \end{cases}$$

where

$$\dot{\sigma}(t) = \begin{cases} 0 & , \text{ if } u^*(t) > +U_{\max} \\ 0 & , \text{ if } u^*(t) < -U_{\max} \\ y(t) - r(t) & , \text{ otherwise} \end{cases} .$$

The recommended digital implementation, including the one-cycle delay, is defined by

$$u^*[k] = -K_{11}\hat{x}_1[k-1] - K_{12}\hat{x}_2[k-1] - K_2\sigma[k-1]$$
$$u[k] = \begin{cases} +U_{\max} & , \text{ if } u^*[k] > +U_{\max} \\ -U_{\max} & , \text{ if } u^*[k] < -U_{\max} \\ u^*[k] & , \text{ otherwise} \end{cases}$$

where

$$\sigma[k+1] = \begin{cases} \sigma[k] & , \text{ if } u^*[k] > +U_{\max} \\ \sigma[k] & , \text{ if } u^*[k] < -U_{\max} \\ \sigma[k] + T\left(y[k] - r[k]\right) & , \text{ otherwise} \end{cases} .$$

For a more complete discussion of integrator windup and its compensation, including a detailed example with illustrative simulations, please consult the notes posted to canvas.

1.4 Simulation of the Positioning System

Control engineers follow a systematic process when working on a development project. The process begins with an understanding of the plant physics, including the derivation of its time-domain dynamic model. Selection of an actuator and sensor come next, where the options under consideration should be limited to those that guarantee existence of feedback gains providing internal stability; such guarantees are established from controllability and observability analyses. The next step is to obtain numerical values for all actuator-plant-sensor parameters appearing in the design model, either by deriving these values from physical principles, by finding these values on data sheets, or by evaluating these parameters using experimental measurements. Once the design model parameters are known, the next step is to compute feedback gains. These feedback gains depend on eigenvalue locations typically selected with transient response specifications in mind, but over-ambitious design choices may lead to large feedback gains and consequent problems due to actuator saturation and sensor noise. Hence, *simulation* is an important next step in the design process, especially for microcontroller-based implementations; such implementations contain both continuous-time (plant) and discrete-time (controller) subsystems, so the simple `for` loop used for continuous-time simulation gets augmented with an `if` conditional to account for the update rate of microcontroller code. Simulation provides a convenient way to explore the influence of unmodeled dynamics, parameter uncertainty, eigenvalue/bandwidth specification, controller update rate, and actuator-sensor quantization. We are now at the simulation stage in this project.

1.4.1 Actuator-Sensor Quantization

Before considering the task of simulating the complete control system, let's first learn how to simulate the effect of signal quantization. Our actuator supplies a voltage using 2000 duty cycles over a ± 24 V range. Our sensor measures a position using 1000 counts per mechanical revolution. The resulting discrepancies between the actuator input signal and actuator output signal, and between the sensor input signal and sensor output signal, are computed in the following Matlab code; both input-output relations are graphically displayed in Figure 3.

```
% actuator quantization (2000 duty cycles for +/- 24 V)
Qa = 48/2000;
Xa = 0.4;
xa = Xa*(-1:1/1000:1);
ya = Qa*round(xa/Qa);
subplot(121), plot([-Xa Xa],[0 0],'k',[0 0],[-Xa Xa],'k')
hold on, stairs(xa,ya,'LineWidth',1.5), hold off
axis([-Xa Xa -Xa Xa]), axis square
xlabel('desired voltage [V]'), ylabel('actual voltage [V]')
title('actuator quantization')

% sensor quantization (1000 counts per revolution)
Qs = 2*pi/1000;
Xs = 0.1;
xs = Xs*(-1:1/1000:1);
ys = Qs*round(xs/Qs);
subplot(122), plot([-Xs Xs],[0 0],'k',[0 0],[-Xs Xs],'k')
hold on, stairs(xs,ys,'LineWidth',1.5), hold off
axis([-Xs Xs -Xs Xs]), axis square
xlabel('actual position [rad]'), ylabel('sensed position [rad]')
title('sensor quantization')
```

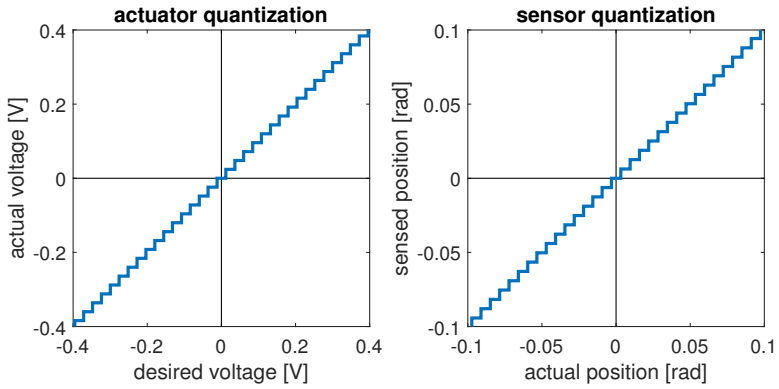


Figure 3: Visualization of PWM actuator quantization and QEP sensor quantization.

1.4.2 Description of Simulation Code

Now let's consider our main task, which is to simulate the overall system consisting of our continuous-time DC motor plant, with quantized actuator and quantized sensor, under the influence

of a discrete-time state-space integral controller commanded by a reference signal, with (or without) anti-windup compensation. The supplied Matlab code is intended to assist with this task.

First you must enter values for the plant parameters; inertia `J`, friction `F`, inductance `L`, resistance `R` and magnetic coefficient `K`. Parametric uncertainty is captured by assuming that parameters `J` and `F` represent the true plant, but that `Jhat` and `Fhat` are the parameters actually used for controller implementation; vary the `*1` factors to explore robustness to *parameter mismatch*. Dynamic uncertainty exists if the plant has more state variables than its design model, e.g. if the voltage-to-current transient is not fast enough and/or if the driven load exhibits non-rigid components. The simulation code accounts for this phenomenon by adopting a third-order simulation model and a second-order design model; vary `L` to explore robustness to *neglected dynamics*.

Next you must enter controller hardware parameters; sensor quantization `Qy`, actuator quantization `Qu` and supply voltage `Vdc`. After that you must enter controller software parameters; timer period `T`, regulator bandwidth `lambda_r` and estimator bandwidth `lambda_e`. Typically, we make the estimator eigenvalues faster than the regulator eigenvalues by a factor of four, but you can change the `*4` factor if desired. Finally, you must enter values for `revs` and `anti`, which determine the amplitude of the reference command and the presence or absence of anti-windup logic.

Additional features of this code are as follows: default parameters conform with the specifications in §2.2; software-generated actuator saturation limits are set to 99% of supply voltage to avoid current sensing problems (a feature not considered here); the external disturbance torque is assumed to be zero; the overall system is initialized at the zero equilibrium, with external inputs and state variables all equal to zero; the feedback gain matrices `L`, `K1` and `K2` are computed according to the procedure shown earlier; the forward Euler approximation is used to discretize the controller with period `T` and to simulate the plant with time-step `h`; a one-cycle delay in application of `u` is assumed, consistent with our standard interrupt-based code flow choice made previously; and supply current is determined from motor current and motor voltage using

$$i_{\mathrm{dc}}(t) = \frac{i(t)v(t)}{V_{\mathrm{dc}}}$$

which assumes a lossless power converter.

2 Lab Assignment

2.1 Pre-Lab Preparation

Each individual student must work through the pre-lab activity and prepare a pre-lab deliverable to be submitted *by the beginning of the lab session*. The pre-lab deliverable consists of a brief typed statement, no longer than two pages, in response to the following pre-lab activity specification:

1. Read through this entire document, and describe the overall purpose of this week's project.
2. Simulate the control system using the supplied code, in order to provide informed responses to the following questions; use words to convey your observations (don't submit plots).
   (a) Verify the default values for `R`, `K` and `F` that appear in the supplied code (which is based exclusively on SI units), using information provided on the motor data sheet.[4]
   (b) What happens when `Jhat` and `Fhat` are not equal to `J` and `F`?

[4]Since the motor data sheet does not provide sufficient information to determine the default values for `L` and `J` that appear in the supplied code, these two default values were obtained by measurement.

(c) What happens when `lambda_r` and `lambda_e` are varied?

(d) What happens when `T` is varied?

(e) What happens when `r` is varied?

(f) How big is the steady-state positioning error?

Please note that it is not essential to write application code prior to the lab session; the point of the pre-lab preparation is for you to arrive at the lab session with a clear understanding of how the plant should respond to the controller we will implement during the lab session.

2.2 Specification of the Assigned Tasks

2.2.1 Parameter Identification

A preliminary task for model-based controller design is to approximate plant parameters, in this case α and β, from measured input-output data recorded using open-loop excitation (see posted notes on parameter identification). Using the supplied parameter identification program and input-output data produced as specified in Lab 5 Task 2, approximate the numerical values of α and β. Choose $T = 1$ ms to match the timer interrupt frequency used during data collection. Choose $\lambda = 50$ rad/s, or perhaps a bit higher, to (i) encompass the controller bandwidth assigned below for the position control task and to (ii) remove the influence of high-frequency noise.

Instructor Verification (separate page)

2.2.2 Position Control

Develop code that implements the state-space integral controller of §1.3.3, including the refinement of §1.3.4; incorporate the possibility to disable or enable the anti-windup compensation, in order to easily demonstrate the benefits of such compensation. A squarewave reference command should be generated in code to request periodic transitions between two programmable positions; for context, imagine that your motor is connected to a belt-pulley apparatus that will locate a robotic pick-and-place tool on a linear motion axis in order to automate the assembly of a printed circuit board. Set the supply voltage to 24 V and the current limit to 3.5 A. The PWM and QEP modules should be configured as in Lab 5.

Use a 1 kHz timer interrupt frequency, a 50 rad/s regulator bandwidth and a 200 rad/s estimator bandwidth. Since the overall bandwidth has been specified to be 50 rad/s, the corresponding time constant will be 20 ms and the corresponding settling time will be approximately 100 ms. Since the controller update period has been specified to be 1 ms, there will be approximately 100 controller updates per transient response, which adequately emulates continuous-time corrective actions.

Perform data logging of u (in V) and y (in rad) to record two complete forward-reverse cycles. Display u and y in graph windows to demonstrate the system's behavior, with and without anti-windup compensation. Generate response plots in Matlab that compare the experimental results with the simulation results. Consider two cases to see what happens when small and large motions are attempted using step-like reference commands:

(a) Set the programmable positions to 0 rad and 2π rad, and request transitions between these positions once per half second; one complete forward-reverse cycle will take 1 second.

(b) Set the programmable positions to 0 rad and 10π rad, and request transitions between these positions once per second; one complete forward-reverse cycle will take 2 seconds.

Instructor Verification (separate page)

GEORGIA INSTITUTE OF TECHNOLOGY
SCHOOL of ELECTRICAL and COMPUTER ENGINEERING

ECE 4550 — Control System Design — Fall 2019

Lab #6: DC Motor Position Control

Instructor Verification Page

Lab Section	Begin Date	End Date
L01, L02	October 8	October 22
L03, L04	October 10	October 24

To be eligible for full credit, do the following:

1. Submissions required by each student (one per student)
   (a) Upload your pre-lab deliverable to canvas before lab session begins on begin date.
   (b) Upload your `main.c` file for §2.2.2 to canvas before lab session ends on end date.
2. Submissions required by each group (one per group)
   (a) Submit a hard-copy of this verification page before lab session ends on end date.
   (b) Attach to this page the hard-copy plots requested in §2.2.2.

Name #1: _____

Name #2: _____

Checkpoint: Verify completion of the task assigned in §2.2.1.

Verified: _____ **Date/Time:** _____

Checkpoint: Verify completion of the task assigned in §2.2.2.

Verified: _____ **Date/Time:** _____