Brushed DC Motor Control: Parameter characterization, open loop and PI controller simulation

Dave Seaton
EE Student
ECE480 Design Team 4
Michigan State University

Introduction

During the Fall Semester of 2010, Texas Instruments collaborated with Michigan State University's Senior Capstone Design course to expand their upcoming line of DC Motor control solutions. While their C2000 family of microcontrollers already provided consumers with an easy way to develop DC machine systems, they wanted other available options to better suit the wide variety of customer needs. Thus, the low power MSP430 family was chosen for its versatility and cost.

Besides the students' task of designing the DIMM control card and accompanying software, the DC motor(s) used must be characterized in order to simulate, develop, test, and properly demonstrate the system. Not only is the characterization important to understanding how the motor operates, but it is essential for designing compensators that increase performance and reduce error. This application note will go over some of the methods for and other reasons behind motor characterization. It will also touch base upon motor simulation and how the two relate to each other. Both of these practices play a critical role in overall product design.

DC Motor Model

A DC Motor system can be represented by an electromechanical schematic that equates electrical losses and mechanical forces to finite, measureable values. The entire schematic is used to develop a system-level transfer function that characterizes the motor's behavior. This schematic is shown in Figure 1 below. Circuit analysis techniques are then used to obtain a single transfer function (Eq. 1) located on the next page.

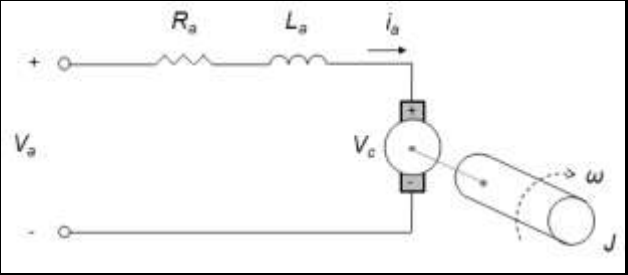


Figure 1. Representative DC Motor Schematic

$$\frac{\omega}{V_a} = \frac{K}{(J \cdot L_a)s^2 + (J \cdot R_a + b \cdot L)s + b \cdot R_a + K^2}$$

Equation 1. Equivalent Motor Transfer Function

State variable (unit)	Description
V_a (volts)	Supply voltage
R_a(Ohms)	Armature resistance
L_a(Henrys)	Armature inductance
i_a (Amperes)	Armature current
V_c (volts)	Back EMF of the motor
ω $(^{rad}/_{s})$	Speed of the rotor
J $(^{kg}/_{m^2})$	Inertia of the system
b (not pictured)	Friction coefficient

Table 1. State Variables and Descriptions

As one can see in the table above, each variable in the transfer function correlates to a real component of the system. Another important aspect of the function is that it equates input voltage (V) to the motor speed (ω) as an output. This is important to note because the software that was released with the MSP430 motor control suite interprets back-EMF to equate as rotor speed.

Procedure for Determining Component Values

After deriving the equations for a DC motor system, it is necessary to then quantify each term. Although there are instruments that may automate these tests, we will step through the procedure of manually determining each quantity. Staying true to the scientific method, it is recommended that enough data points are taken to average the numbers in order to ensure accuracy.

The first and arguably easiest parameter to obtain is the winding resistance (Ra) of the coils inside of the DC machine. This can be determined directly by using an Ohmmeter or indirectly by exciting the coils with a voltage and monitoring the current. By the nature of having a brush physically coming into contact with the commutator, the resistance value will vary greatly for both methods. For this reason, measurements derived from either method must be repeated for a number of rotor positions.

One would assume that by merely connecting an Ohmmeter to each terminal, the winding resistance can be found instantly. However, due to some capacitive and inductive properties behind the sensors within the meter, the displayed value will tend to "float" and constantly change value. As a result, the readout on the screen does not reflect the true resistance of the metal. To avoid this phenomenon, one can measure resistance without having to use a dedicated acquisition device.

This secondary method is reminiscent of a basic principle of power electronics: the short (locked rotor) and open circuit (synchronous speed) tests. In order to monitor current without any transients affecting the value, the rotor must be locked in a certain position. Paying close attention not to overload the motor, apply a small voltage to the terminals. Watch as the current rises and levels off at a value. Once recorded, allow the rotor to turn a small amount and repeat the current monitoring technique. It is also important to realize that exposure to high amounts of current will generate heat and affect the thermal properties of the materials (such as resistance). After a sufficient amount of data points have been recorded, use Ohm's Law (Eq. 2) to find each point's resistance and then average them. This result will be close to the actual winding resistance. Table 2 shows the characterization of an Anaheim Automation BDD-38-63-12.0V-14200 Brushed DC motor. Be observant of the particular motor's maximum ratings because the current will rise quickly.

Voltage (V)	Current (A)	Resistance (Ω)
1.00	0.72	1.139
1.00	0.84	1.190
1.00	0.78	1.282
…	…	…
2.00	1.65	1.212
2.00	1.48	1.351
2.00	1.64	1.219

Table 2. Sample Resistance Characterization

The next motor parameter, equally as important as resistance, is the inductance (La) of the coils. This property of the motor determines how current flows into the armature. There are two ways of measuring the inductance, the first being the LCR meter (used for the Anaheim Automation motor). This approach is fast and accurate, although one may not have an LCR meter available to them. This LCR method involves setting the meter to determine "series inductance" and attaching the motor terminals to the input of the instrument (no external voltage source is necessary). Just as in the resistance derivation, it is important to measure this value from different rotor positions. Table 3 contains the set of inductances for the same brushed DC motor.

Approx. Rotor Angle (deg)	Measured Inductance (μH)
~0	230.25
~70	320.11
~150	229.10
~210	232.94
~300	231.60

Table 3. Sample Inductance Measurement Data

Another way of determining inductance of coils is through the Time Constant method. Since series inductance/resistance (LR) or capacative/resistance (RC) circuits are governed by Eq. 3, a function generator and oscilloscope (with current measuring capabilities) can be used to find the circuit time constant. With the time constant and resistance already derived, the inductance or capacitance can be extracted by using Eq. 4.

(eq 3.)

$$\Delta Value = \left[Value_{final} - Value_{start}\right]\left(1 - \frac{1}{e^{t/\tau}}\right)$$

(eq. 4)

$$\tau = \frac{L}{R}$$

Equation 3. Universal Time Constant Formula
Equation 4. Series LR Time Constant

The goal is to let the function generator mimic a repeated step response by using the square wave function. Being sure to set the generator to a reasonable amplitude and frequency, attach the current clamp to the wires connected to the motor terminals. Once the oscilloscope is monitoring the current, the cursors can be used to determine levels at certain time intervals. Using these amperage-time value pairs, a relatively accurate inductance value can be calculated.

The third parameter that is essential in characterizing a DC motor is the Motor Constant (K). Deriving this value relies directly on having the no-load speed of the motor (in rad/s) and some indication of the Back EMF of the motor. Usually the manufacturer will provide a datasheet with the no-load speed and current. Regardless, it is suggested that the current is measured again due to part variations that may occur. Back EMF can be obtained by monitoring motor current under no-load and then multiplying it by the winding resistance in order to calculate the voltage drop. Once this is determined, Eq. 5 relates the input voltage to the speed, much like the original transfer function (Eq. 1) of the motor. This is scaled by using the quantity K just as one would Gain (A) of an opamp circuit. In order to solve for K, simply divide the summated voltage by the rotor speed.

$$\omega = \frac{V_a - V_{BEMF}}{K} = \frac{V_a - I_a \cdot R_a}{K}$$

Equation 5. Speed-Voltage Equation

Finally, there are two other mechanical forces that govern how a DC motor system behaves. Since the system inertia (J) and friction constant (b) are typically a result of material properties and load characteristics of the system, how their values are determined will not be summarized in this application note. The motor itself will have a very small (if not negligible) inertia and friction constant. For this reason, their values are approximated using standard models of similarly-sized 12 Volt DC Motors. Since these values are incredibly small, the exact effect on the transfer function are relatively small compared to other system properties. This concludes the basic characterization of a DC motor.

Simulation using MATLAB and Simulink

Once the DC motor has been characterized, it is time to utilize those quantities in simulation models. These simulations not only better our understanding of the motor, but they provide us with critical predictions of how the system will respond. This forecasted result can be used to engineer more robust designs earlier in the project (as opposed to finding problems in a finalized revision of the project). Because the MSP430 Control Card software utilizes native PI compensation, it is essential to debug our software knowing that our Proportional and Integral gains are properly set. There are many more reasons why running simulations are important that are not discussed in the scope of this note. This guide will introduce open loop models in MATLAB and provide closed loop, compensated models in Simulink (which is a subset of MATLAB) in order to show the multiple ways to arrive at a satisfactory simulation.

Open Loop Simulation in MATLAB

When a motor is simply plugged into a voltage source with no type of speed control other than the voltage level, this is referred to as "open loop" control. When there is a feedback loop that monitors a motor characteristic, the system turns into "closed loop" control, which will be detailed in later sections. This open loop can be simulated in MATLAB by initializing the state variables, creating a transfer function with the constituent parts of the fraction, and then applying a `step()` or emulated `ramp()`

input to that transfer function. MATLAB will then automatically plot a graph from the results of the `step()` function. It is important to note that these simulations were run in the **R2008a** version of MATLAB. Previous versions may lack some of the functionality required to run the following procedure. These next steps will show one how to execute the simulation:

1. Open “MATLAB R2008a” (or equiv.), the status bar located at the bottom of the window will display “Ready” when the program is finished loading.
2. In the “Command Window”, enter the code for declaring and initializing the state variables:

```
  K = 0.007384;
  R = 1.2284;
  L = 0.000230081;
  J = 0.0009;
  B = 0.00724;
```

3. After these variables appear in the “Workspace” column to the left (if not available, typing a variable name into the command prompt and pressing ENTER will output a confirmation that the correct value was assigned to that variable), it is now time to create the transfer function with the code:

```
  num = K;
  den = [(J*L) ((J*R)+(L*b))
((b*R)+K^2)];
  dcmotor=tf(num,den)
```

4. Because there is no semicolon after the last statement, the finalized transfer function should output to the “Command Window”.
5. Finally, it is time subject the transfer function to the unit response input by entering:

```
  step(dcmotor,0:0.01:2);
```

6. MATLAB’s status window will display “Busy” while it is calculating the plot. After a few moments, MATLAB will open a new window “Figure 1” with the simulation data (the title of the graph can be modified by using the statement `title(‘string’)`).

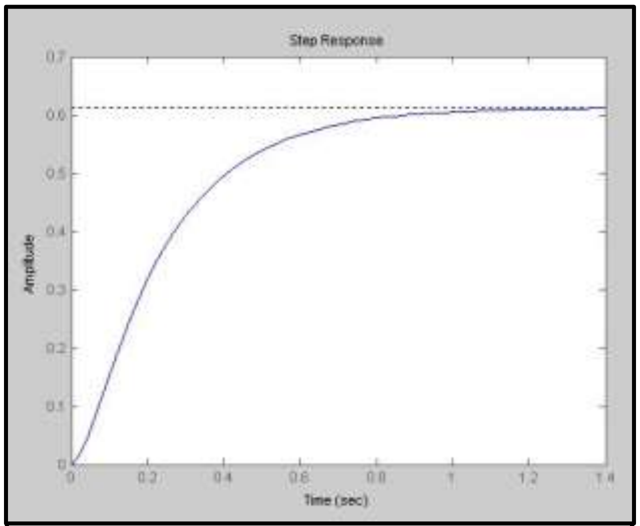
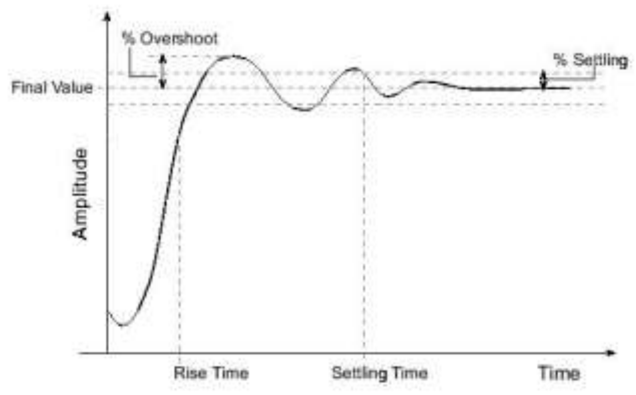


Figure 2. Speed-Voltage Equation

The graph for this simulation is located above in Figure 2. By using the menu in the figure, basic properties of the plot can be displayed to the user. There is another MATLAB function called by entering stepinfo(numerator, denominator) into the command window. This will display the numeric results for the rise time, settling time, overshoot, undershoot, and peak for that specific transfer function. This concludes the simulation of the motor in open loop configuration.

PID Compensation

PID compensation is the practice in which gain, differential, and integral terms literally compensate for system dynamics that effect performance and error. A unit called a controller is placed in the control loop to condition the signal sent to the end device. The PID controller calculates an error term based on nth order equations determined by a desired set-point and the actual state of the system. The controller attempts to make up for this error and tries to modulate the signal sent to the device, in this case: a DC motor. PID controllers are used to eliminate unwanted system characteristics such as slow response time, overshoot, and steady state error. Each term and its exact effect on the response can be referenced in Figure 3. It is important to note that adjusting the gain on a certain term to fix one attribute may lead to the worsening of another.



Gain Increased	Rise Time	Overshoot	Settling Time	Steady-State Error
K_p	Decrease	Increase	None	Decrease
K_i	Decrease	Increase	Increase	Decrease
K_d	Small Decrease	Small Decrease	Small Decrease	None

Figure 3. PID Gain Effect on Step Responses

The MSP430 Control Card software utilizes a slightly less complex version of the PID controller called a PI compensator. This is simply the PID without the differential gain. Without these values set properly, the system could send an input to the motor that would result in an over-current scenario. This would damage the internals if sustained over an extended period of time. Luckily in software, a ramp control module is used to lessen the stress put on the system by an instant step input. This drastically reduces the likelihood of an excessive overshoot event. In order to estimate the PI values as a starting point, root locus techniques can be used. The `rlocfind()` command in MATLAB can be used on a transfer function in order to find its proper gain coefficient. Once this derived, that gain coefficient can be used for the proportional gain in the compensator. Then, manually adjusting the integral gain will yield a satisfactory response through trial and error.

PI (Closed Loop) Simulation in Simulink

After calculating the appropriate estimates for the proportional and integral gains, it is now time to implement them in a Simulink model. Simulink is a subset of MATLAB and is an extremely useful tool for signal processing and system analysis. The graphical convention that this program uses is easy to learn and pleasantly easy to follow. The next portion will instruct the user how to assemble a signal model, specifically the PI control used for the MSP430 DC Motor Control software.

1. Open MATLAB R2008a and navigate to the "**Start**" menu located in the lower-left corner of the window.
2. Find and click the menu item "**Simulink → Library Browser**".

3. After the library window appears, go to "**File → New → Model**" to start a new signal model.
4. Navigate to the menu item "**Simulation → Configuration Parameters**" and under "**Solver → Solver Options**" change the "Type" to **Fixed-step [ode4 (Range-Kutta)]** with a step size of **0.01**.
5. Now the DC Motor system will be assembled by placing all of the necessary subsystems on the model and then connecting them. Go to the "**Simulink**" directory in the "Libraries" window, click and drag:
   (1) Sources → **Repeating Sequence**
   (1) Math Operations → **Sum**
   (1) Ports & Subsystems → **Subsystem**
   (1) Continuous → **Transfer Fcn**
   (1) Signal Routing → **Mux**
   (1) Sinks → **Scope**
6. Connect the "triangular-shaped" output ports to the corresponding "arrow-shaped" input ports by clicking and dragging the line. The signal model should be assembled exactly as in **Appendix A**. Holding down "Ctrl" before clicking a line will create a new branch.
7. Once the model is assembled we must now construct the Subsystem and set other properties.
8. Double left-click the **Repeating Sequence** node. Under "Time Values" enter **[0 .125 2]** and under "Output Values" enter **[0 2 2]**. Click OK.
9. Double left-click the **Sum** node. Under "List of Signs", make the last "**+**" sign a "**–**". Click OK.
10. Double left-click the **Transfer Fcn** block. Type the coefficients derived from the "Open Loop Simulation in MATLAB" section of this application note into the corresponding numerator and denominator windows. Click OK.
11. Double left-click the subsystem to access its model. The input and output nodes should already be in place, but remove the line connecting them. Just as before, go to the Simulink Library Browser and click and drag:
    (2) Math Operations → **Gain**
    (1) Math Operations → **Add**
    (1) Continuous → **Integrator**
12. Connect the subsystem exactly as in **Appendix A** using the same method as the entire signal model.
13. Double left-click the proportional or integral gain nodes and enter the appropriate value. Click OK.
14. Exit out of the subsystem window and the simulation should now be ready to execute. Click the "**Play**" button or go to "**Simulation → Start**".
15. Double left-click the **Scope** node and the signals should be displayed in the output window. Right-clicking anywhere on the graph and selecting "**Autoscale**" can help fit the plot to the window. Also in the scope window, clicking the "**Parameters**" icon then "**Data History**" tab allows you to modify the number of data entries the plot displays. Uncheck the "**Limit data points to last**" option to obtain the entire simulation.

This concludes the Simulink PI simulation section of the application notes. All signal models and output plots are located in the Appendix A.

Appendix A

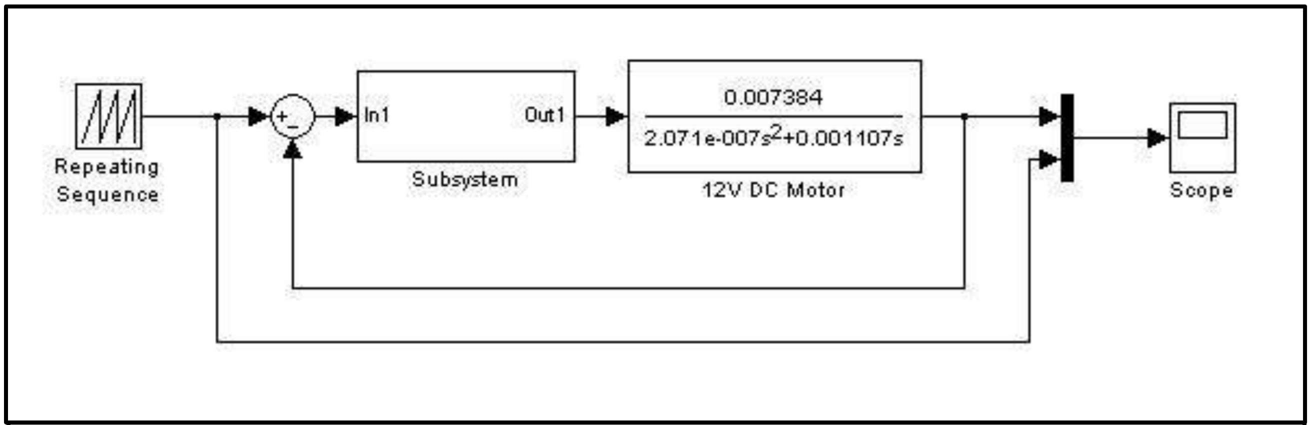


Figure 4. Simulink Model for Closed Loop PID Motor Control

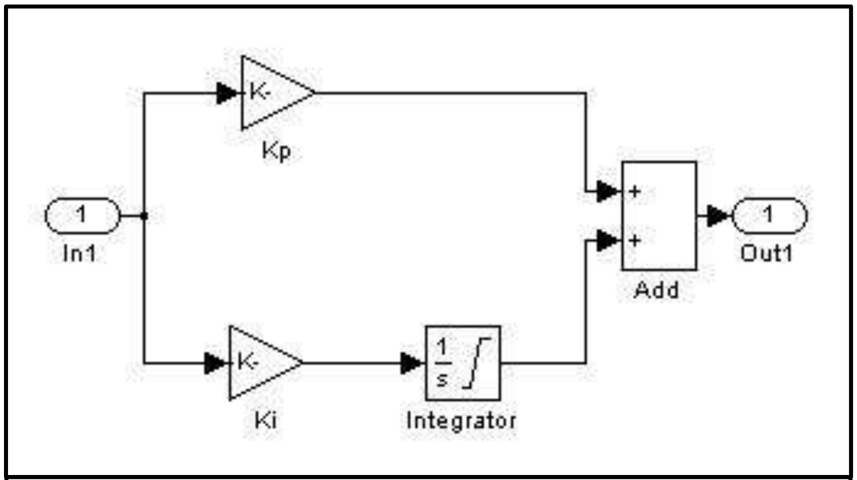


Figure 5. Subsystem Model from Figure 4

Appendix A
(continued)

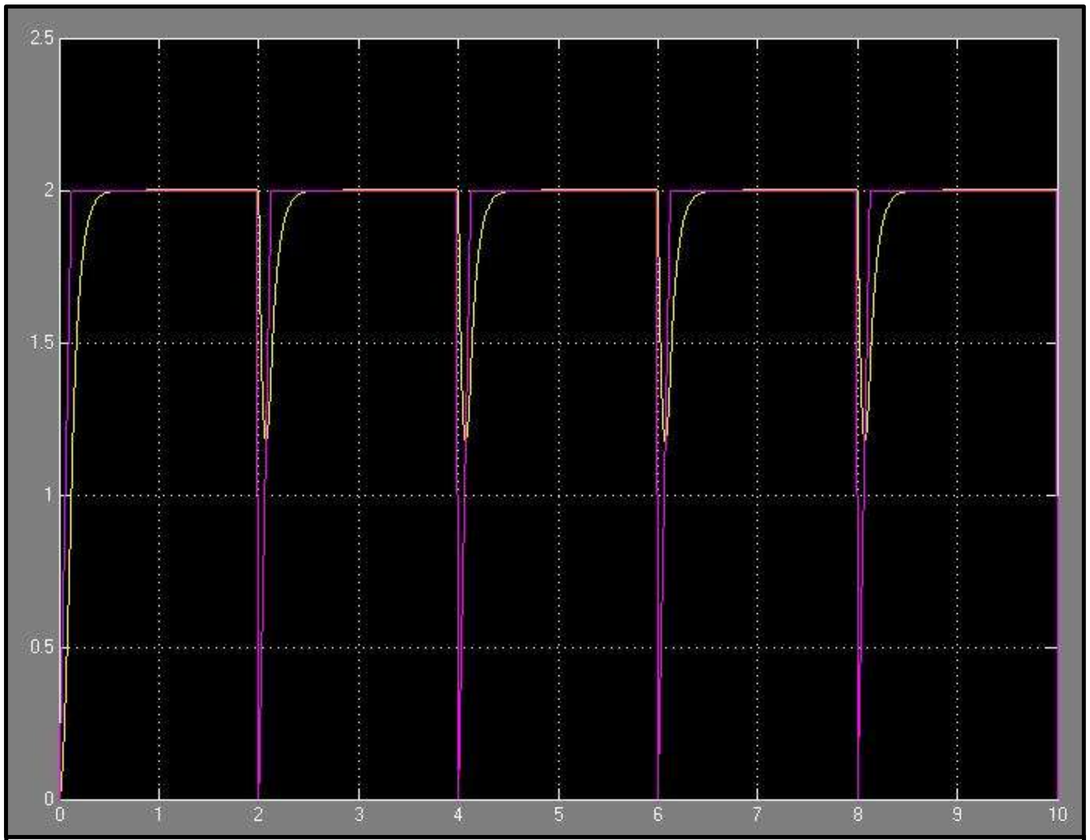


Figure 6. Scope Output from the Simulink DC Motor Simulation