Teaching Your PI Controller to Behave (Part VIII)

Dave (Wisconsin) Wilson Apr 15, 2013

Dave Wilson, Motion Products Evangelist, Texas Instruments

We are now on the eight part of a ten part series about teaching your PI controller to behave like you want it to. At the heart of this discussion is a proposed tuning technique which allows you to be more proactive with the tuning process and reduce the amount of empirical guesswork at the back-end of your design process. TI has already used this tuning technique on several motor control designs with good success! But as with any “parameter based” design process, the results are only as good as how well you know your parameters. In summary, for a permanent magnet synchronous machine, we need to know the following parameters to tune the PI loops:

Rs (motor stator resistance)

Ls (motor stator inductance)

P (# of motor poles)

λ_r (rotor flux)

J (system inertia)

If you’re lucky, the motor data sheet may have some of this information. But in many cases you’re on your own trying to figure out what they are. But don’t despair! TI has just released a new sensorless Field Oriented Control algorithm called InstaSPIN-FOC which can measure most of these parameters for you automatically! Figure 1 is a high level diagram of InstaSPIN-FOC, which shows its three distinct components:

1. Motor ID: This algorithm interrogates the motor to gather most of the parameters above. It is usually run just once during the commissioning process for a particular motor.

2. FAST: This is the observer which calculates the real-time motor parameters needed for Field Oriented Control. The neat part is that it doesn’t require a shaft angle or speed sensor to do it! F.A.S.T. stands for Flux, Angle, Speed and Torque, which represent the

main outputs of the FAST observer. A diagram of how the FAST observer can be used in a Field Oriented Controller is shown in figure 2.

3. PowerWarp: This is a special energy savings operating mode used to reduce the copper losses in AC Induction Machines, especially during light-load conditions.

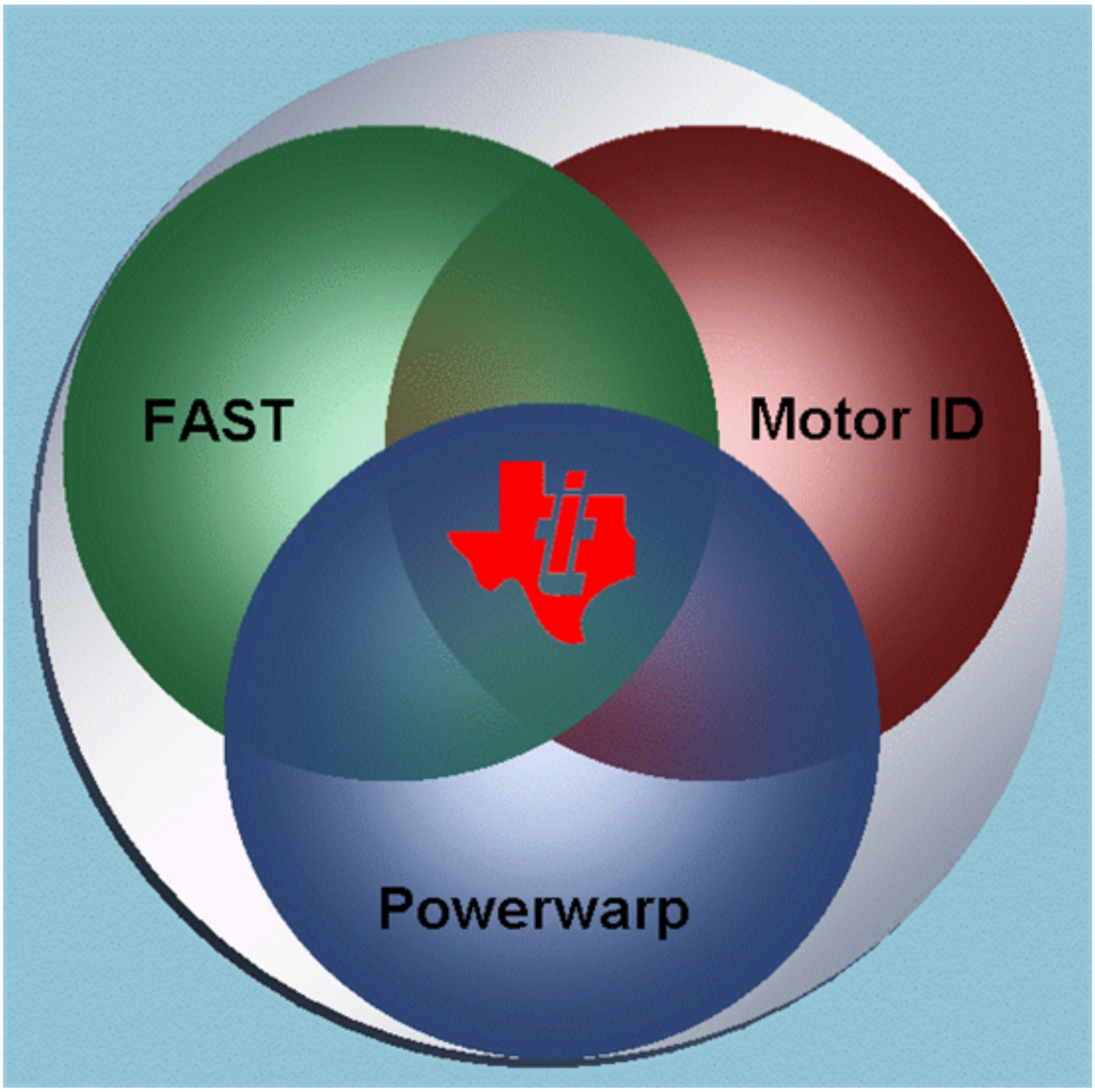

Figure 1. Components of InstaSPIN-FOC

There is actually a fourth component of InstaSPIN-FOC which is part of the motor ID function, and runs in real time to provide continuous estimates of the stator resistance. This is the parameter that typically changes the most as the motor heats up during operation. Monitoring it in real time significantly improves the performance of the FAST observer, and also allows you to dynamically change Kb in the current PI controllers to maintain pole/zero cancellation. I have a lot more to say about this feature, as well as the other components of InstaSPIN-FOC in an upcoming blog series.

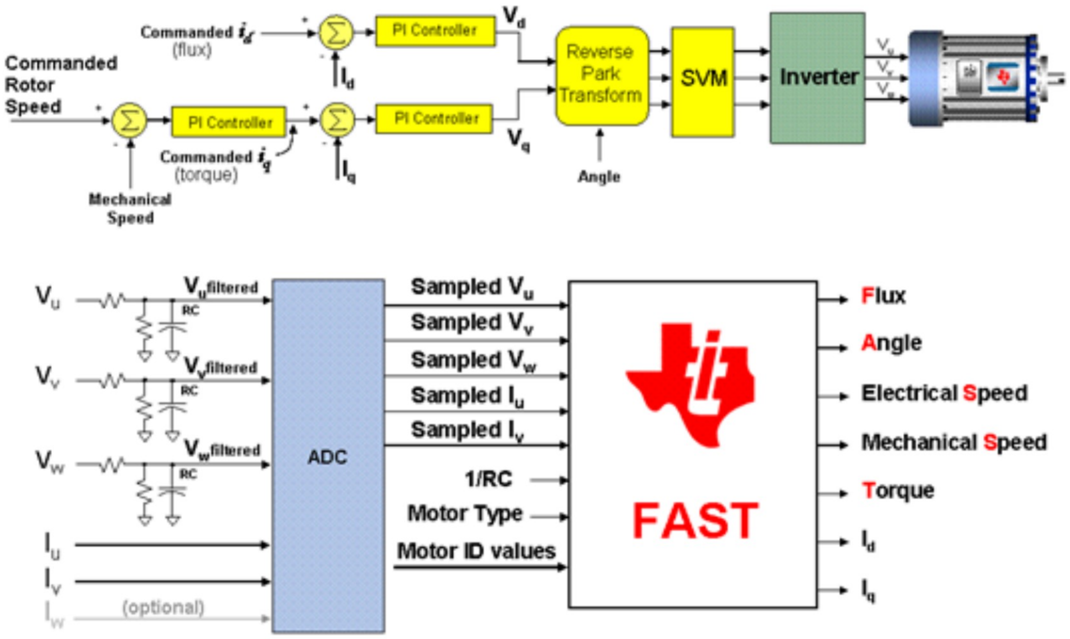

Figure 2. FAST Embedded in a Field Oriented Controller

So we see that InstaSPIN-FOC provides measurements for Rs, Ls, and the rotor flux. Fortunately, the number of motor poles is often listed on most data sheets, or can easily be determined empirically. This leaves only system inertia (J) that is unknown. But the entire velocity loop tuning procedure hangs on whether we accurately know what this parameter is. Even though InstaSPIN-FOC doesn't directly measure inertia for you, the following is a proposed procedure for how to use the outputs from FAST to measure inertia:

1. Design the current controller by setting Ka = Ls/τ, or π Ls/(10T_S), whichever is lower. Recall that τ equals the velocity feedback filter time constant, Ls = the stator winding inductance, and T_s = sampling period. Also set Kb = Rs/Ls.

2. Disable the velocity loop and use only the current loop. Set Id =0, and apply a fixed value of q-axis current which is sufficient to cause the motor and load to accelerate through a good portion of the rated speed range.

3. As the motor is ramping up to speed, periodically sample and record the values for motor mechanical speed (ω(n) in rad/sec) and also the motor torque (T_1(n) in Newton-meters). Both of these measurements are outputs from the FAST observer. The motor torque should be pretty consistent since Iq is constant, and the motor speed should monotonically increase for a well behaved load. For example, figure 3 shows the simulated samples of the speed output from the FAST observer for a permanent magnet motor driving a fan load with fixed inertia. The sampling frequency is 200 mS over a 4 second span, resulting in a total of 20 samples.

4. Now enable the velocity loop and set the PI coefficients to very low values that will barely allow you to spin the motor up to speed. Since we don't know inertia yet, we can't accurately specify the velocity loop coefficients anyway. Don't worry about the dynamic response of the system at this point, since we are only interested in steady state performance.

5. Command the motor to go to each of the recorded speeds from step 3 by monitoring the speed output of the FAST observer. After the system response has completely settled out for each commanded velocity, again record the torque values from FAST ($T_2(n)$ in Newton-meters). This torque will correspond to the static load torque apart from any acceleration artifacts.

6. We now have all the data required to solve for inertia as a function of time (J(n) in kg-meters^2) by using the following difference equation:

$$\hat{J}(n)=\frac{2T_{spd}\left(T_1(n-1)-T_2(n-1)\right)}{\omega(n)-\omega(n-2)}$$

Equ. 1

where: J(n) is the inertia at sample time "n" (kg-m^2)

T_{spd} is the periodic sampling period of the speed loop (sec)

T_1 is the total torque consisting of static friction and acceleration torque (Newton-meters) from the FAST observer

T_2 is the static friction torque only (Newton-meters) from the FAST observer

ω is the mechanical speed measurement (rad/sec) from the FAST observer

.

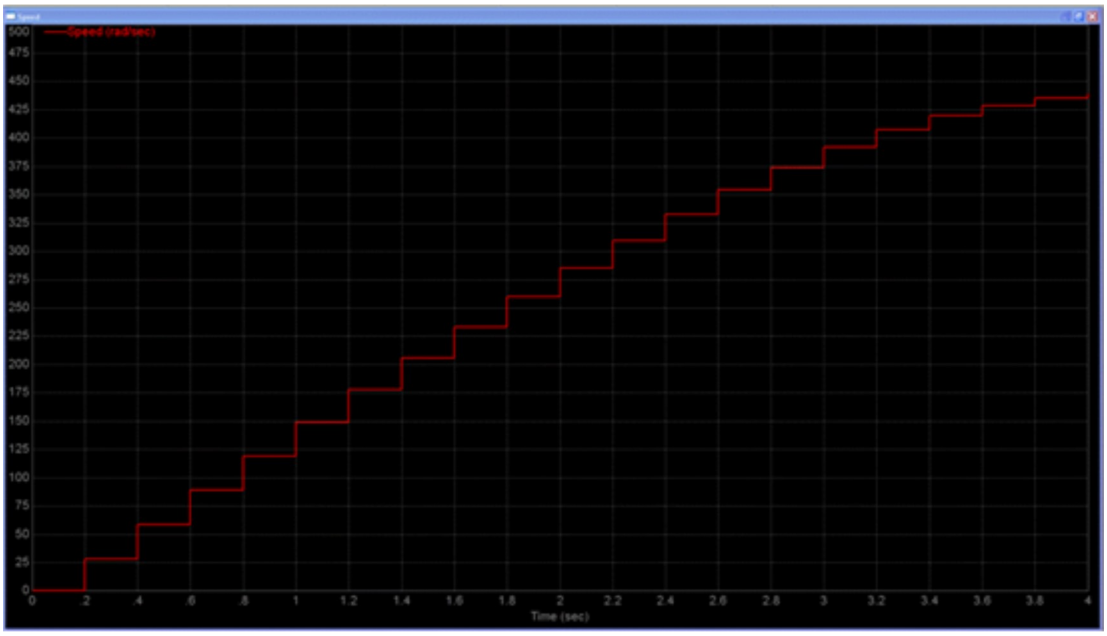

Figure 3. Simulated Speed Samples of a Fan Controller with Fixed Inertia

Figure 4 shows an example inertia plot using equation 1 applied to the data from figure 3. The estimated values are plotted in red at each time step along with the actual load inertia in green. In this particular scenario, the inertia is constant and the torque ripple is zero, making it the best condition to estimate inertia. As you can see, the results are most accurate for the range in figure 3 where the acceleration is fairly constant. In this case you would simply time-average those inertia estimates to come up with a single value for system inertia. But what if you have excessive torque ripple, such as a compressor load? In that case, you can think of the torque ripple as unwanted noise in the system. To maximize the signal-to-noise ratio, you should set the q-axis current as high as possible to increase the ratio between the T_1 and T_2 torque readings. It would also help to do a time average of the steady-state torque readings at each velocity point.

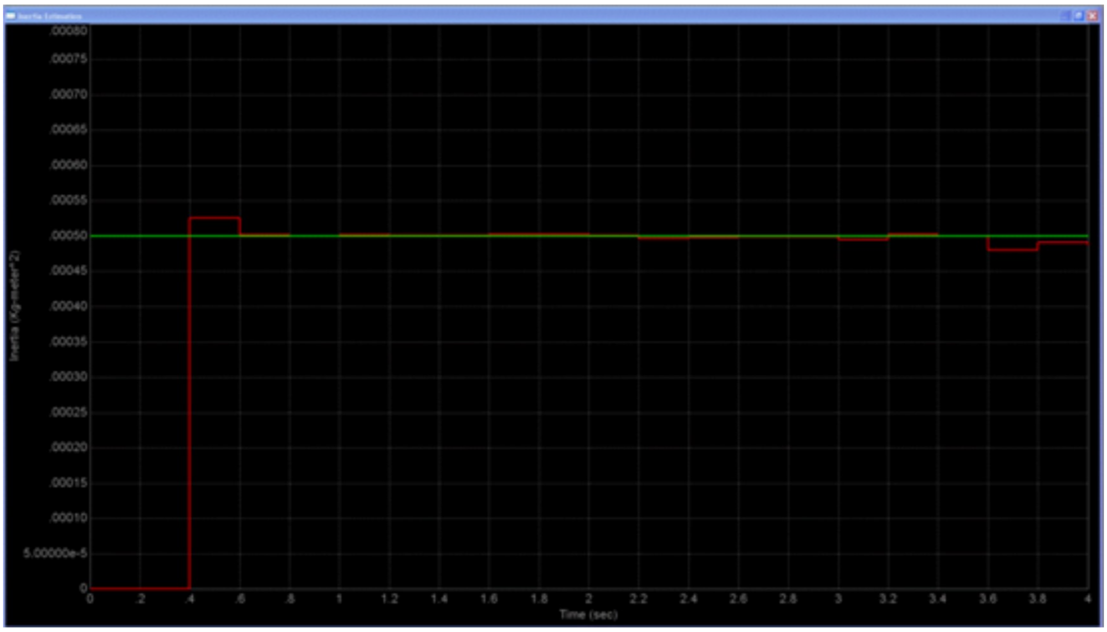

Figure 4. Estimated vs. Actual Values of System Inertia as a Function of Time

For applications where inertia is a function of velocity (like a centrifuge load), this technique can provide a distinct advantage over other techniques. Since equation 1 calculates inertia as a function of time, and you also have data for motor speed as a function of time from step 3, you can combine the two sets of data to correlate inertia as a function of speed. Having this data in a Look Up Table in memory allows you to dynamically update the Kc term in the velocity PI controller as the speed changes to optimize the response at any speed.

So far, we have discussed PI tuning in generic terms which are mostly independent of the control topology. In my next blog, I want to focus specifically on some of the more subtle points to consider when designing PI controllers for Field Oriented Control systems. Until then…

Keep Those Motors Spinning,

Dave

www.ti.com/motorblog

1 comment 0 members are here

Mayank Jain70 *over 7 years ago*

Hi

Your blog has really been helpful in pi tuning in continuous domain in simulink in Matlab. Currently what i am doing is that after Vds* and Vqs* (modulating signals) are generated, i am converting it back to alpha-beta and then to abc frame, which would then be sent for pwm generation for inverter. The controllers are working absolutely fine till here and motor works fine. The problem comes when i am using SVPWM block provided by texas library. When i am replacing (alpha-beta to abc) with alpha-beta to SVPWM block (using data type convertsion block before svpwm block) and then generating PWM signals, the motor response shoots up and down wildly and settles to zero speed. Kindly suggest the steps for the same.

Thanks

Mayank