CIRCUIT CELLAR®
THE MAGAZINE FOR COMPUTER APPLICATIONS

FEATURE ARTICLE

Alan Kilian

Quick and Easy Motor Control

Implementing a PIC-SERVO Controller

Before you get your motor running and head out on the highway, you may want to listen to what Alan has to say about the PIC-SERVO from JR Kerr Engineering. If you need to control a DC motor with encoder feedback, this chip will fit the bill nicely.

There are many methods for controlling motors. You can use stepper motors, DC motors that use PWM and encoders for feedback, or a complete precision motion control system that uses an LM628 motion-control chip.

These systems require a lot of work building and programming microprocessors to get the motor started. This article will show you how to achieve the first move in an hour, for only about $40!

NEW PRODUCT

Recently, JR Kerr introduced a new product that is helpful for controlling a DC motor with encoder feedback. The PIC-SERVO is a pair of PIC microcontroller chips designed to implement a PID servo-control system.

The system is implemented as a two-chip set. One chip is used as a 16-bit quadrature encoder counter. The other is used to implement the PID control algorithms and communicate through an asynchronous serial port to a host computer or microprocessor. The set is available for $35 through several distributors. You only need to add a motor driver.

The JR Kerr web site provides excellent descriptions of the pinouts, theory of operation, example implementation, and sample code, so I won't expound on those. Follow the schematic in the documentation, hook it up to your serial port, and let's get going (see Figure 1).

MOTOR CONNECTIONS

First, you need to hook up the motor correctly. Attach the encoder inputs to the PIC-ENC chip, and attach the motor power leads to the motor driver. If you have more than one PIC-SERVO chipset, set the addresses of each chipset using the instructions at the JR Kerr site.

Next, use the PIC-SERVO PWM mode to confirm the motor is hooked up correctly. Enable the PWM mode with the `LOAD_TRAJECTORY` command, and set a PWM value to about 64 to get the motor moving.

You can determine if the position values are counting up or down by using the `READ_STATUS` command. They should be counting up. If they are counting down, reverse either the motor's power connections or swap the A and B encoder wires. Then, using the `LOAD_TRAJECTORY` command, reverse the direction of the motor and verify that the encoder counts down.

At this point, the motor and encoder are wired properly. However, if you want the positive direction of motion to be the other way, swap the motor power and encoder connections.

Go through this exercise for each motor/encoder/PIC-SERVO unit.

After you have your motors correctly wired and tested, let's get to the PID part.

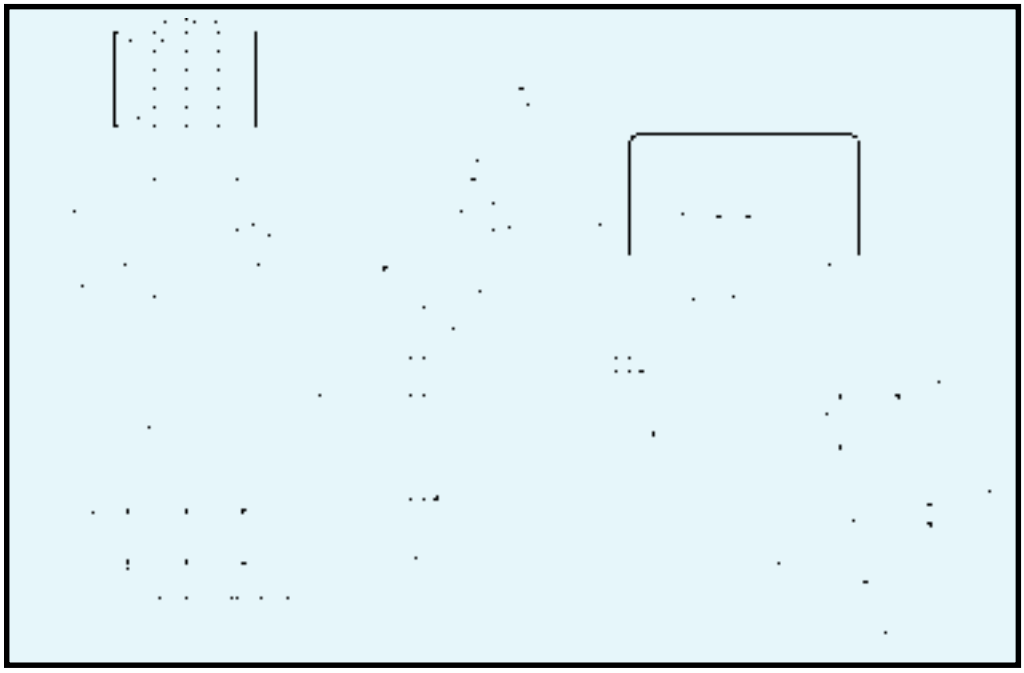

Figure 1—*This is a schematic for the minimal PID control system using the PIC-SERVO.*

SAFETY

The motor-control values depend on your robot's configuration. Completely assemble the robot before tuning the motion controls. If you add equipment later, you will probably have to tune the controls again.

You are going to instigate strange motions for the motors and mechanics, so be prepared with an emergency stop button. Something can catch in a gear train easily, or the mechanics may fly apart during tuning. You can hook up a switch in series with the motor DC power supply to stop the motors when things get out of hand.

PROPORTIONAL TERM

Here are some abbreviations that I'll use for the article. The proportional gain is called Kp. The derivative gain is Kd, and the integral gain is Ki. The desired position, velocity, and acceleration are P, V, and A. The position error, where you want to be minus where you are, is Ep. And, Ev is the velocity error, how fast you want to go minus how fast you are going.

Let's review the PID coefficients. Prop your robot off the ground so that the wheels can turn. Send two packets to the PIC-SERVO to get things set up. Use the `LOAD_TRAJECTORY` command to set P to zero, V and A to a large value like 0x1FFFFFFF, and enable the servo loop. Then, use the `SET_GAIN` command to set the position-error limit to 0x3FFF, and the Kp, Kd, and Ki terms to zero.

Kp indicates how hard the motor should work to remain in its current position. Set Kp to 1 and enable the servo loop; the motor will stay where it was when you enabled the servo.

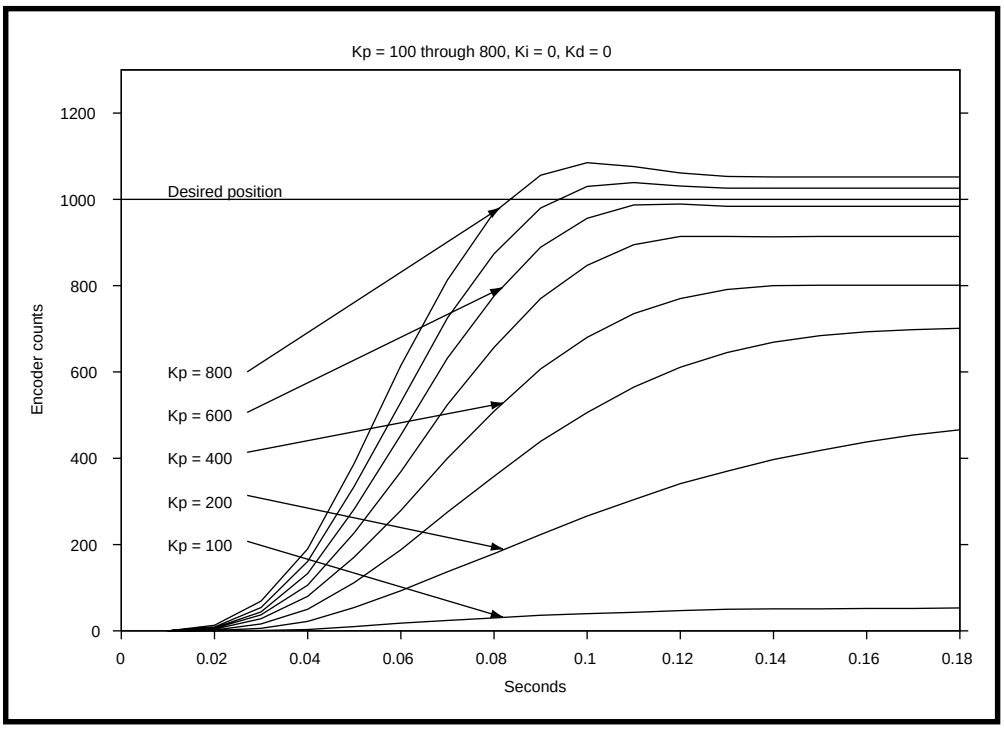

Figure 2—*This shows the motion trajectories when using only Kp.*

If you move the motor shaft by hand, the motor driver will try moving the motor back to where it started. It can't achieve starting location, but you should be able to see a voltage across the motor with a voltmeter, or a 'scope. If you turn the motor in the other direction, you should see the motor-direction bit change. If these things happen, the controller is trying to servo the motor back to its desired location. The value of the PWM signal is:

$$PWM = \frac{Kp \times Ep}{256}$$

where PWM = 0 is fully off, and PWM = 255 is fully on. You can see this if you have a

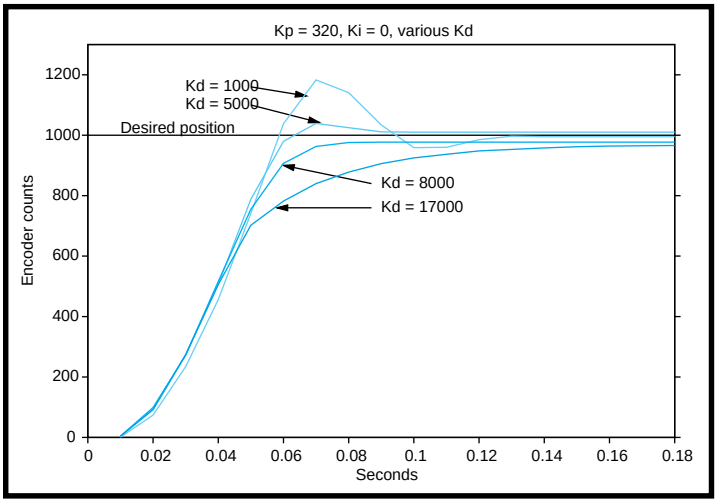

Figure 3—*These are the motion trajectories when using Kp and Kd.*

400-count-per-revolution encoder. With 1 Kp, you need to turn the motor 80 revolutions to get the PWM signal half-way on.

$$PWM = 1 \times \frac{(400 \times 80)}{256} = 125$$

If you choose a Kp of 160, the PWM signal can turn fully on in one revolution:

$$\frac{160 \times (400 \times 1)}{256} = 250$$

Because you only set the Kp term to 1, it's like you have a weak spring trying to move the motor back to the starting location. The farther you move it from its set point, the harder the motor turns on. You need to turn it far to get more motor torque. Increasing Kp means the controller works harder to hold the motor at the original position.

Now, try setting Kp to higher values. A value of 2000 will probably be like a strong spring. If you move the encoder 32 encoder counts (29°), the motor will turn on fully, trying to get the encoder back to zero.

The motor will vibrate back and forth across the set point like a spring. In fact, you probably should sneak up on the larger values. Jumping to a large value might cause your motor to vibrate quickly and overheat.

Testing with only Kp gives poor performance. First, Kp is set too low, and the motor doesn't get close to the desired position. As you increase Kp, you get closer to the target location; then it begins to overshoot the target and backs up. Even greater Kp values make the motor overshoot and then oscillate around the desired stopping point.

Figure 2 shows the result of moving a 1000-encoder count using Kp = 100 through Kp = 800.

THE DERIVATIVE TERM

The Kd term controls the desired velocity, like the Kp term controls the desired position. The "d" stands for derivative, and the derivative of position is velocity, so let's call it velocity.

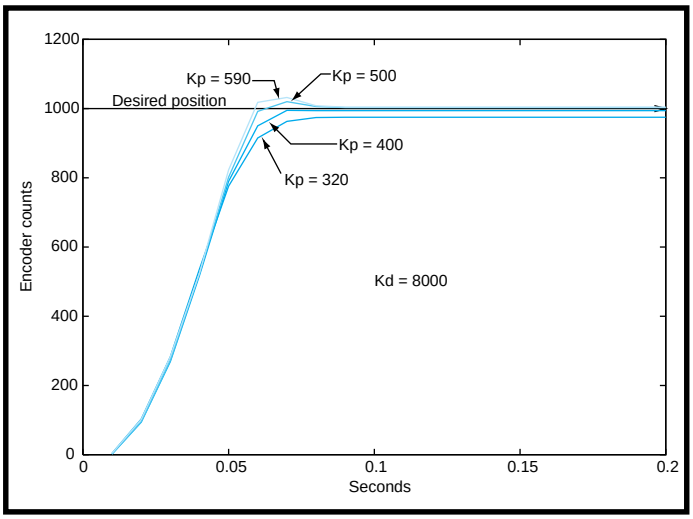

Figure 4—*These motion trajectories are the results you get from using a large Kd value and increasing Kp values.*

You've probably increased Kp to get the motor running, and now it's oscillating around the desired set point, never stopping at the correct point. The servo loop works perfectly. The problem is that you didn't program the motor to stop at the set point, you programmed it to head towards the set point if it's not there.

If you increase Kd, you're telling the servo to follow a desired velocity in addition to a desired position. The PIC-SERVO will internally generate both a desired position and a desired velocity 2000 times per second during the move. It compares these values to the actual position and velocity and generate two error signals, Ep and Ev.

Now, I'll explain how Ev and Kd make it speed up or slow down. The PIC-SERVO wants the motor to stop when it is at the end of a move, so the desired velocity is zero, but the actual velocity is the motor's speed as it passes the set point. Because the motor is going too fast, the control loop reduces the PWM duty cycle to slow the motor as it nears the end of its move. The larger the Kd value, the more the system tries to make the motor follow the desired velocity and stop at the end of the move.

If Kd is too great, the slightest motor motion causes the motor to turn on fully in the opposite direction, which makes the motor move in that direction, which, in turn, makes the controller turn the motor on in the other direction. You can experiment with this value by setting Kp to zero, Kd to 100, and giving the motor a fast, sharp tap that causes a significant velocity error and determines if Kd is too great.

You should hear a quiet buzz, but you might get a wildly oscillating motor that tries to break your gear train, so be ready with the off switch.

FIRST STEPS

Try to make a 1000-count step. First, set the velocity and acceleration values to high values (100,000 for both). Set Kd

to the highest value that didn't cause things to break, and set Kp to 100. Then, set the PIC-SERVO to servo mode by sending the LOAD_TRAJECTORY command followed by the position, velocity, and acceleration values.

Issue a START_MOTION command and watch what happens. The motor moved, but probably not the entire 1000 counts. It should move in the correct direction at least. Move the motor by hand, watching the encoder values and the status until it reaches 1000 counts. After the status confirms it's complete, begin increasing the Kp value, making one-rotation moves until the motor makes a move of almost 1000 counts.

If the motor moves more than 1000 counts, turns around, goes back past the set point, reverses again, and oscillates forever, there's too much Kp.

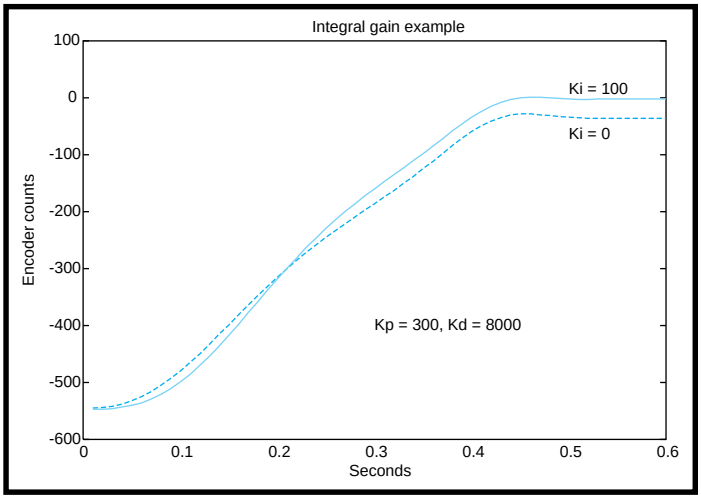

Figure 5—*Notice the motion trajectories that result when using different Ki values.*

TEST DRIVE

Because I planned to test drive the controller, I wanted a simple way to see what a move looks like. I built a motion control system to drive an Etch-a-Sketch (check out www .etch-a-sketch.com/).

Each knob was replaced with a nylon gear and two motors and 96 line encoders were mounted to a base plate. Because each line on a quadrature encoder produces four counts at the encoder counter, this gave me 384 counts per revolution at the motor, and with a 122:24 gear ratio, a total of:

$$384 \times \left(\frac{122}{24}\right) = 1950$$

counts per revolution at the knob.

Using the PIC-SERVO, I drove the horizontal knob at a constant velocity, and commanded the vertical motor to do a step motion. The high gear ratio makes it difficult to see the ringing by looking at the screen.

Figure 2 shows that I need 600 Kp before I get close to the desired set point. Kp values greater than 600 overshoot the desired set point.

Figure 3 shows what happens when

I increase Kd from 1000 to 8000. As Kd increases, performance gets better until it reaches 8000, then the performance worsens and never reaches the set point.

Figure 4 shows what happens when I increase Kp using 8000 Kd. Again, performance increases to a point, and then gets worse.

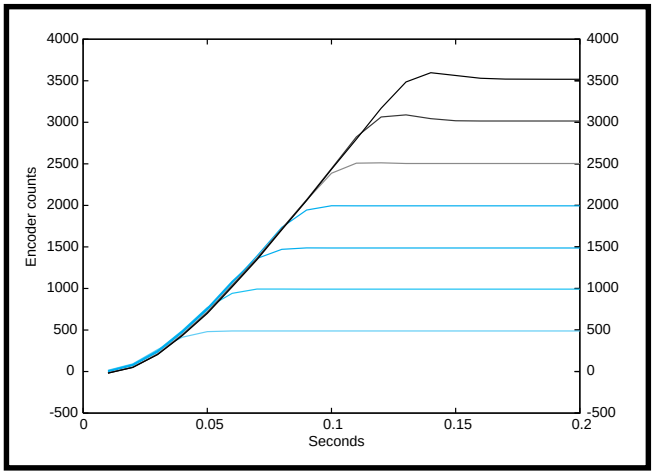

Figure 6—*These are the motion trajectories for various move lengths.*

WHAT'S THE I TERM?

If you're going to control a simple robot that won't try to stop on a hill, or need to be precise in its location, you don't need an I term.

To demonstrate the I term's value, I added a stick and ball of clay to the Etch-a-Sketch. If I try to position the motor so the stick is horizontal, the clay will force the motor off position. This simulates a robot stopping on a hill. The `READ_STATUS` command moved the position and velocity and plotted the results using a graphing program.

With an upward move, the motor stops too soon and never reaches the desired location (see Figure 3). If you add more Kp to get it to stop at the correct location, you will overshoot the end point and ring. The motor won't stop at the set point. If you look at what's happening, you'll understand why.

Imagine that the motor did stop at the set point. What would happen? The position and velocity errors would be zero, so Kp × Ep + Kd × Ed = PWM = 0. The motor is turned off and it falls backwards due to the weight. Then Ep, Ev, and PWM increase, getting closer to the endpoint again.

If you could use the constant position error to increase the PWM, you could pinpoint the set point. If you could leave that PWM value on when you get to the set point, you would be able to stop at the set point even if there is a force trying to move it away from the set point.

That's what the I term does. It allows the error value to add up over time, and makes the motor move slowly to the final set point. Be careful because too much Ki or integral limit can make things unstable quickly. Figure 5 shows what happens with a 0 and 100 Ki.

MOVE LENGTHS

Figure 6 shows the performance when you try to do different lengths of moves. Performance is successful to a move length of 2000 encoder counts, then it overshoots. This means that you may need different parameters depending on the move length. There is no penalty for looking at the length, and then setting the coefficients for that move differently from the other lengths.

GET MOVING

Getting a PID control loop to perform is complex. By trying different values, you might get the desired performance. Or, you may reach the goal by starting with one parameter, changing it until you get good performance, and then working on the others systematically.

You'll learn a lot by taking the system for a spin, and you'll get a great motion control system, too.

Alan Kilian is a lead visualization programmer at the University of Minnesota's Computational Biology Centers. He's worked as a programmer for 17 years at Cray Research Inc. and CyberOptics Inc. and is a founding member of the Twin Cities Robotics club (www.tcrobots.org). You may reach him at kilian@pobox.com.

SOFTWARE

The parts list and software is available on the *Circuit Cellar* web site.

SOURCES

PIC-SERVO
J R Kerr Automation Engineering
www.jrkerr.com

Distributors
Jameco
(800) 831-4242
(800) 237-6948 Fax
www.jameco.com

HdB Electronics
(650) 368-1388
(800) 2 TRY HDB
Fax: (650) 368-1347
www.hdbelectronics.com

Circuit Cellar, the Magazine for Computer Applications. Reprinted by permission. For subscription information, call (860) 875-2199, subscribe@circuitcellar.com or www.circuitcellar.com/subscribe.htm.