Control Technology Corporation

Control TechNotes

Creating a PID Loop Using Quickstep

Technical Note No. 22, February 9, 2000

Why Create a PID Outside of Servo Control?

In certain applications, you may want to create a PID loop for control of a process variable outside the realm of servo control. Using an analog input as feedback from your process variable you can create a PID control loop in the Quickstep Programming Language.

The example that follows assumes that there are additional tasks that need to be done and shows multitasking. The program was designed to contain some extra programming to insure that the PID loop executes on a (nearly) regular interval.

Why Execute a PID Program at Regular Intervals?

Why do you have to execute a PID program on regular intervals? It is because the integral (I) and differential (D) portions of a PID loop are time dependent functions. For example, the integral function is derived from a cumulative error, which is determined by adding the value of each error (setpoint - feedback = error) during each scan of the program (store Cumulative_error + error to Cumulative_error). If the time between each sample were changed then the integral gain of the system would also change.

25 South Street, Hopkinton, MA 01748

Sample Program

```
[1] Initialize
    ;;; This program creates a PID loop to be used for control
    ;;; loops whose update times can be slow such as temperature
    ;;; or liquid level control loops. A (nearly) constant scan
    ;;; time is established using the millisecond clock and by
    ;;; making the PID task a supertask.
    ;;;
    ;;; In this step we are setting some initial values for
    ;;; our program so that it will work the first time it is
    ;;; used. Once final values for these and other adjustable
    ;;; register values have been arrived upon this step should
    ;;; be edited to reflect the changes or deleted all together.
    ------------------------------------------------------------------------
    <NO CHANGE IN DIGITAL OUTPUTS>
    ------------------------------------------------------------------------
    store 1000 to Scale_Output
    store 50 to Scantime
    store 1000 to Feedback_Span
    store 5000 to Setpoint
    goto Next
[2] Multitask
    ;;; This step launches the PID tasks and other tasks to be
    ;;; used in a given application.
    ------------------------------------------------------------------------
    <NO CHANGE IN DIGITAL OUTPUTS>
    ------------------------------------------------------------------------
    do (PID Multitask_1 Multitask_2 Multitask_3) goto Initialize
[3] PID
    ;;; We set this task as a “supertask” to help maintain a constant
    ;;; scan time for this task.
    ;;;
    ;;; Normally in a multitasking situation the program would scan
    ;;; the next appropriate step in each task in the order that they
    ;;; are entered in the do loop. In this program, for example,
    ;;; the order would be:
    ;;;
    ;;; PID->Multitask_1->Multikask_2->Multitask_3->PID-> etc.
    ;;;
    ;;; With the PID task set as the supertask the order of execution is
    ;;; as follows:
    ;;;
    ;;; PID->Multitask_1->PID->Multitask_2->PID->Multitask_3->PID-> etc.
    ;;;
    ;;; Note: You should use this only if you are running more than
    ;;;       2 tasks.
    ------------------------------------------------------------------------
    <NO CHANGE IN DIGITAL OUTPUTS>
    ------------------------------------------------------------------------
    store 1000 to super_task
    goto Next
```

Sample Program

```
[4] PID_Loop
    ;;; Here we create a time-based loop where this step is
    ;;; executed at regular intervals. We must do so because the integral
    ;;; ( v*dt ) and differential( dv/dt ) factors are both time
    ;;; dependent. The first store command zeros the millisecond clock
    ;;; at the beginning of the step. At the end of the step, we check
    ;;; for if the passage of time has exceeded the desired SCANTIME,
    ;;; starting the loop again if we have. So long as the SCANTIME is
    ;;; longer than the time it takes to execute all the statements,
    ;;; this step should run at regular intervals.
    ;;;
    ;;;                          Feedback Calibration
    ;;;
    ;;; Store statements 2 through 4 are used to scale the feedback variable
    ;;; for use by the loop. Your feedback should always be scaled for the
    ;;; same range as your Setpoint. For example, let's say you want your
    ;;; Setpoint to be 0 for a minimum and 10000 for a maximum. Now let's
    ;;; assume that your feedback device provides 200 for a minimum and 8000
    ;;; for a maximum. You would set Feedback_Zero to -200 and Feedback_Span
    ;;; to 10000/(8000-200)*1000 or 1282 to give you a 0 to 10000 range for
    ;;; your feedback.
    ;;;
    ;;;                       Error and Proportional Gain
    ;;;
    ;;; Store statement 5 determines the instantaneous error between the
    ;;; Setpoint and the feedback. This error is then multiplied by the
    ;;; proportional gain to provide value of P_out. Note that P_gain = 1000
    ;;; represents a gain of 1 when the value of Scale_Output is equal to
    ;;; 1000.
    ;;;
    ;;;                   Cumulative Error and Integral Gain
    ;;;
    ;;; Store statement 7 is used to generate the integral factor of our PID
    ;;; loop. The Cumulative_error is the sum of all errors that occurred
    ;;; during each SCANTIME interval. The Cumulative_error is then
    ;;; multiplied by the value of I_gain and stored in I_out. Note that
    ;;; I_gain = 1000 represents a gain of 1 when the value of Scale_Output
    ;;; is equal to 1000.
    ;;;
    ;;;               Differential Error and Differential Gain
    ;;;
    ;;; Store statement 9 uses the present error and the error in the
    ;;; previous scan (Previous_error) to determine the differential error.
    ;;; The differential error is then multiplied by D_gain and stored to
    ;;; D_out in Statement.  Note that D_gain = 1000 represents a gain of 1
    ;;; when the value of Scale_Output is equal to 1000. Next, we take the
    ;;; present error and store it to Previous_error for use in the next
    ;;; scan of the Task.
    ;;;
    ;;;                Summing the Outputs and Setting Limits
    ;;;
    ;;; Store statements 11 and 12 sum the outputs of the P, I, and D
    ;;; functions and scale that sum in statement 13. The scaling of the
    ;;; output allows us to have gain values as low as .001 for a given
    ;;;
    ;;; Scale_Output value of 1000. We next check the summed output against
    ;;; limits we have set for the output. The reason this is done is that
    ;;; the device we send the signal to may require a signal lower than
    ;;; +/- 10 Volts.
```

Sample Program

```
    ------------------------------------------------------------------------
    <NO CHANGE IN DIGITAL OUTPUTS>
    ------------------------------------------------------------------------
    store 0 to millisecond_timer
    store Feedback + Feedback_Zero to Scaled_fdbk1
    store Scaled_fdbk1 * Feedback_Span to Scaled_fdbk2
    store Scaled_fdbk2 / 1000 to Scaled_fdbk3
    store Setpoint - Scaled_fdbk3 to error
    store error * P_gain to P_out
    store Cumulative_error * I_gain to I_out
    store error - Previous_Error to Diff_error
    store Diff_error * D_gain to D_out
    store error to Previous_Error
    store P_out + I_out to Output2
    store Output2 + D_out to Output1
    store Output1 / Scale_Output to Output
    if Output > Limit_Plus goto plus_limit
    if Output < Limit_Minus goto minus_limit
    store Output to loop_out1
    store Cumulative_error + error to Cumulative_error
    if millisecond_timer >=Scantime goto PID_Loop
[5] plus_limit
    ;;; Should the output of the PID loop exceed the set limit
    ;;; this step clamps the output at that limit.
    ------------------------------------------------------------------------
    <NO CHANGE IN DIGITAL OUTPUTS>
    ------------------------------------------------------------------------
    store Limit_Plus to loop_out1
    if millisecond_timer >=Scantime goto PID_Loop
[6] minus_limit
    ;;; Should the output of the PID loop exceed the set limit
    ;;; this step clamps the output at that limit.
    ------------------------------------------------------------------------
    <NO CHANGE IN DIGITAL OUTPUTS>
    ------------------------------------------------------------------------
    store Limit_Minus to loop_out1
    if millisecond_timer >=Scantime goto PID_Loop
[7] Multitask_1
    ------------------------------------------------------------------------
    <NO CHANGE IN DIGITAL OUTPUTS>
    ------------------------------------------------------------------------
    goto Multitask_1
[8] Multitask_2
    ------------------------------------------------------------------------
    <NO CHANGE IN DIGITAL OUTPUTS>
    ------------------------------------------------------------------------
    goto Multitask_2
[9] Multitask_3
    ------------------------------------------------------------------------
    <NO CHANGE IN DIGITAL OUTPUTS>
    ------------------------------------------------------------------------
    goto Multitask_3
```