

Chapter 8: Proportional-Integral-Derivative Control

PID is an acronym for Proportional-Integral-Derivative Control. In this chapter, we will explore each of these control methods and how they work together to efficiently control a system.

One objective of a process control method is to hold a system constant. In the previous chapter we used various means of cycling the heater of our incubator to maintain a desired temperature. Using differential gap, we created an allowable band, or range of temperatures, in which the heater would cycle, causing the temperature to cycle on and off, above and below the desired setpoint. In Chapter 6, we saw how we could use pulse-width modulation (PWM) to add variable amounts of energy to our system in duty cycles between 0% (fully off) and 100% (fully on). While these types of control have their advantages and disadvantages, PID control allows greater control of a system but can be more difficult to implement and tune (or adjust) for optimum performance. Holding a process constant involves continually adding energy that exactly equals the system energy losses. If the system losses were constant, then process control would be as simple as applying one steady state level of drive. However, the factors that affect a process do change. They change in unpredictable magnitudes and at unpredictable rates. Compounding this problem is that a system has reaction delays that must be understood.

An instant change in energy losses due to a disturbance is not felt immediately. Furthermore, an instant change in drive does not create an instant output response. Process control can be as much an art form as it is a science. The first step to understanding PID control is to realize that every system has both gains and losses of energy.

$$E_{\text{System}} = E_{\text{In}} - E_{\text{Out}}$$

A system is said to be in equilibrium when the energy gained equals the energy lost.

$$\text{Equilibrium: } E_{\text{In}} = E_{\text{Out}}$$

In equilibrium, our incubator would maintain a constant temperature. But this is seldom, if ever, the case. Depending on the heater drive and conditions surrounding the incubator, temperature will either be increasing or decreasing. Conditions of the system will change the energy loss, such as room temperature changing, air movement changes around the

tube, sunlight falling on the tube, or the resistor aging. The amount of heat added and removed is seldom constant over long periods of time.

In an oil-flow system, the drive element, the pump, is controlled to maintain a desired oil flow rate. A sudden change in the system may be a valve closing, blocking one path for oil flow. A slow change in the system may be a filter gradually clogging or the oil temperature changing affecting the viscosity or 'thickness'. The pump needs to adjust in order to compensate for these changes.

One other system to consider is an automobile. Typically we want the car to maintain a constant speed on the highway. The engine makes up for friction losses from the tires on the pavement and the air blowing over the car. When the car is maintaining a constant speed, the system is in equilibrium and our foot keeps the accelerator in a constant position. When conditions change, such as the car climbing a hill, the car's velocity changes. The forces increase on the car, removing more energy than the engine is supplying, and the car begins to slow. Without depressing the accelerator more, will the car eventually come to a stop on the hill? No, it will slow to a lower constant speed where once again energy losses equal energy input, and a new equilibrium is reached.

Throttle position directly relates to the power demanded from the engine. This power moves the mass of the car and overcomes friction, wind resistance, gravity due to road incline, etc. If all of these conditions (disturbances) were constant, our car would stabilize at some constant speed. When you want to go 55 mph on level highway, this would correlate to a specific throttle position. If all conditions stay the same, lock in this throttle position and your speed will not change.

Now, think carefully about how you might react in an effort to keep the car going at a constant speed. Consider how you would respond in pressing the gas pedal relative to continually watching the speedometer. The speedometer is the measuring device (sensor) and it is providing you feedback as to how your process is going. We will relate your "natural" reaction to the individual principles of a PID controller as you attempt to drive a steady 55 miles per hour (mph).

First, let's consider your simple response to noticing that you are going 5 mph slower than you desire. You respond by pressing the gas pedal a certain amount. If you had been going 10 mph slower than desired, your reaction would naturally be more forceful. And, likewise if you had been only slightly below the target speed your tendency would

be to only change the pedal position a small amount. Your control reaction is proportional to the absolute magnitude of the mph error.

Next, consider that your car is going up a long continuous hill and your speed drops to 45 mph. At 10 mph below your desired setpoint, the proportional reaction above was enough to overcome some of the effects of the incline but, due to the long hill, your car stabilizes at 50 mph. What would you do? Understanding your objective is to go 55 mph, you would naturally press down a little harder. And, if that brought the speed up to 52 mph, would you stop pressing? No, you would press a little more and a little more until finally you had integrated exactly enough control action to compensate for the steepness of the hill and be at 55 mph. As a smart controller, you would not allow long-term continuous error to persist in your process.

Finally, consider how you might react to driving up a steep hill versus the last long continuous hill. Remember that in our example you are reacting to what you see the speedometer doing – it is providing you process feedback information. If all of a sudden you notice your speed is dropping quickly, what do you assume? Right, you have just gotten onto a very steep hill. What will happen if you do not respond quickly and forcefully to this rapid change in your speed? Things may only get worse, right? Having an element of control in which you respond relative to the rate at which your process is changing can help buffer the affects of rapidly occurring, high magnitude disturbances. Responding to the rate of change at which an error is occurring is the function of derivative control. When you work with derivatives in mathematics you are usually analyzing the slope at points along a changing curve. Derivative control action responds solely to the slope of the error, or the speed at which the value is changing.

Hopefully, as you think of yourself as an intelligent PID controller in our driving example, you can see how a blended response to the absolute magnitude of error (P), the duration that an error persists (I), and the rate that error is occurring (D) can provide very efficient and effective process control. Determining the appropriate blend of the individual control modes must take into account the dynamics of the entire system. As with all continuous process control scenarios, keeping your car at a steady speed is dynamic. There is a dynamic to the magnitude, duration, and rate at which road conditions change. The time required for individual drivers to notice, evaluate, and react to speedometer changes is dynamic. And, the weight, engine torque, and transmission ratio are only a few of the dynamic variables of your car that defines its response to a throttle change. As a “smart controller” behind the wheel you will find yourself quickly adapting the blend of your PID response based on the characteristics of your vehicle as

well as the driving conditions. Keep this example in mind as you continue through this chapter.

Conditions (or disturbances on our system) can change very rapidly; such as when the car suddenly encounters a hill. Or may be very slow; such as tire wear reducing efficiency. PID control can measure and take action on:

1. How far from the setpoint a system is, or the magnitude of the error.
2. The duration for which an error remains.
3. How quickly an error occurs in the system, or the "rate" of change.

The sum of these three evaluations comprises the output drive in an attempt to maintain a system in equilibrium. Figure 8-1 illustrates the evaluation and control of a system for PID control.

The classic PID formula for calculating the controller output is as follows:

$$Co_{PID} = Bias + K_p E + K_I \int E dt + K_D \frac{dE}{dt}$$

or

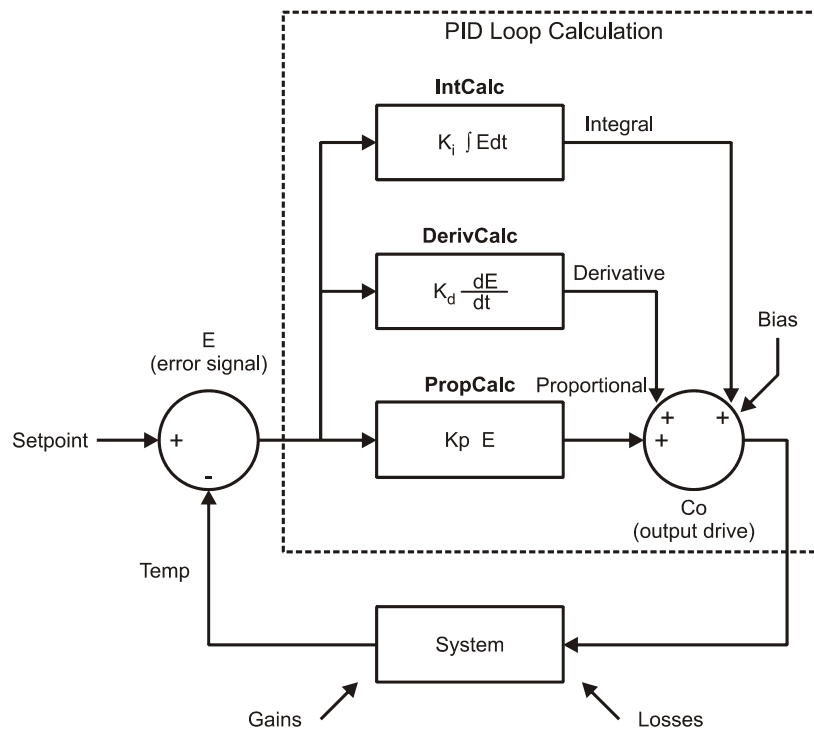
$$DRIVE_{TOTAL} = DRIVE_{BIAS} + DRIVE_{PROPORTIONAL} + DRIVE_{INTEGRAL} + DRIVE_{DERIVATIVE}$$

Where:

Co = controller output or drive
 Bias = User defined drive
 E = Amount of Error*
 t = Time
 K_p = proportional gain
 K_I = integral gain
 K_D = derivative gain
 dE = Change in error
 dt = Change in time

*(Earlier E was used for Energy. Symbols have multiple uses).

Figure 8-1 Evaluation and Control of PID



Breaking down the control in English:

- The controller will deliver a level of drive based on the calculations to try and maintain the system at the desired setpoint.
- **Bias Drive** is the amount of signal needed to drive the system at the setpoint under normal conditions. In Chapter 6, the PWM drive to the heater was manually adjusted until the incubator's temperature was in the operating band. Under constant conditions, approximately 30% of output drive was necessary to keep the system in the band. But, as we saw, when air was blown over the incubator, conditions changed, the drive remained constant and the system dropped below our defined operating limit.

- **Error (E)** is the measurement from the desired setpoint to the actual system condition, such as a difference between the desired temperature and the actual temperature.
- **Proportional Drive** continuously evaluates error and adds or subtracts output drive in an attempt to drive the system back to the setpoint. Too low? More drive. Too high? Less drive. The gain is used to adjust how much action is taken based on the error.
- **Integral Drive** measures the duration and magnitude of the error and adds or subtracts to eliminate the long lasting error. The longer the error and the greater the amount of error, the greater the integral drive will be. Gain is used to adjust the amount of action based on the duration and magnitude of error.
- **Derivative Drive** measures how quickly error changes in respect to time. The faster the error changes, the more action that will be taken to oppose the change to bring the system back under control. The gain once again is used to adjust the amount of action based on this calculation.

With each element in PID, the control action is based on the error, either current, accumulated over time, or change with respect to time. Mathematically, integrals and derivatives are based on an infinite number of points in the time continuum. Consider Figure 8-2 showing an actual reading oscillating around a setpoint. Data is measured and action taken at every possible point. The integral evaluation provides the area under the curve. As an example, if the error resulted in a certain amount of power being expended, the integral of the error would provide the total power used over time. The mathematical result would provide a very precise value for the area under the curve. Note that if a value above the setpoint were considered a positive error, data below the setpoint would be a negative error. If the areas A and B were equal and these two areas were integrated, the result would be 0.

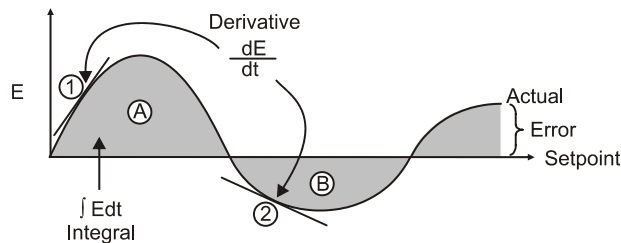


Figure 8-2
Continuous
Measurement for
PID

The derivative evaluation provides a measurement of the change in value with respect to time (the faster the error changes, or the slope at that instant, the higher the magnitude of the error.) At point 1 of Figure 8-2, the value is changing very rapidly with a positive slope (rising). At point 2, the value is changing slowly with a negative slope (decreasing).

Analog circuits using op-amps can provide true proportional, integral and derivative drive where the waveform is processed on a continuous basis. At any given instant, the error is measured and processing of that error is made. In microcontrollers, instantaneous measurement is not possible. Samples are taken at intervals, and approximations are made to provide PID control. Before we test the microcontroller performing PID control, let's first look at a system controlled with analog PID.

Electronic Systems Technologies students, at Southern Illinois University Carbondale, regularly construct and test the floating-ball project as shown in Figure 8-3. An infrared beam is used to detect the position of the magnetic ball. Drive to the electromagnet is used to control the position of the ball. The setpoint is the desired voltage representing the light falling on the detector based on the ball partially in the beam. The actual ball position is detected based on the amount of light striking the phototransistor and producing an output at the collector. The error is the difference between the actual and desired voltages. As the ball begins to drop (light level increases), drive to the magnet must increase, and as the ball gets pulled up (light level decreases), the drive to the magnet must be decreased. When properly adjusted, or tuned, the ball will maintain a relatively stable position, "floating" in the beam.

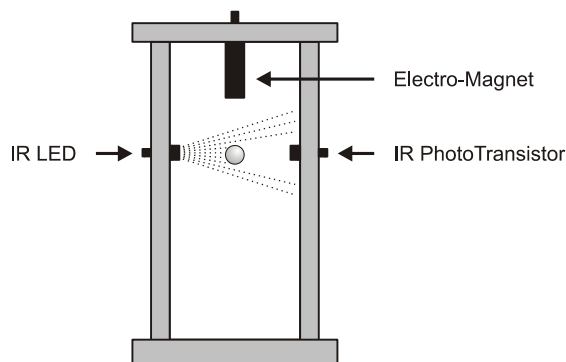
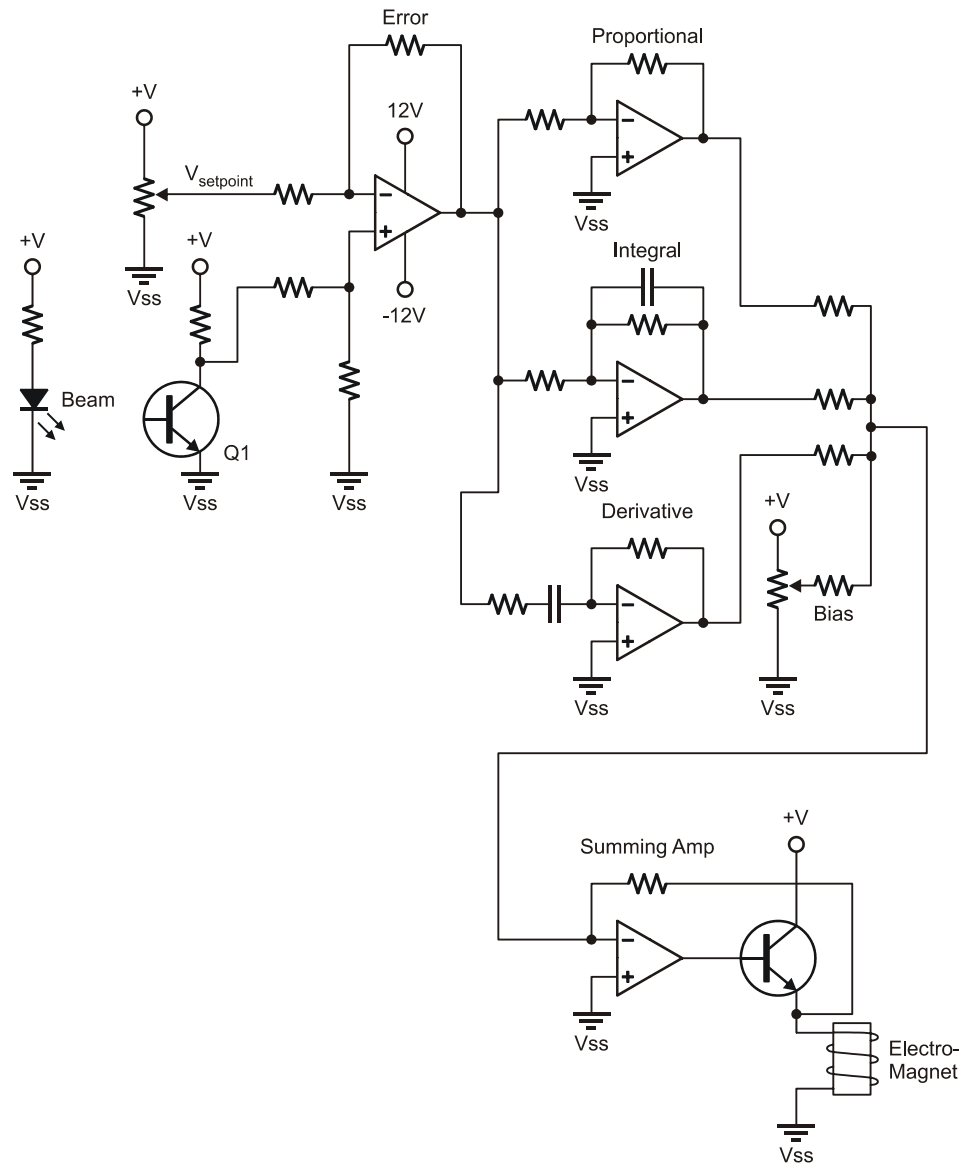


Figure 8-3
Floating Ball
Project

Figure 8-4 Typical Op-Amp PID Control Circuit for the Floating Ball – *DO NOT BUILD*



By using a circuit similar to the one in Figure 8-4 to perform the measurements, calculations, and drive, the position of the ball is continuously evaluated and drive is updated nearly instantaneously. The difference between setpoint and actual voltages is determined (Error) and this signal is processed with three op-amp configurations to amplify the error (proportional), measure the error over time (integral) and the change in error in respect to time (derivative). The three signals and a bias voltage are added together (summing) to provide a drive voltage. The error instantaneously causes a change in the output. By adjusting the resistances for each op-amp stage, the gain, and consequently the contribution to the total output voltage, of the individual P, I and D blocks may be adjusted to achieve stable operation.

While newer advances in digital technology, such as Digital Signal Processors (DSP) can come very close to continuous measurement and control of the analog control circuit in Figure 8-4, programmable microcontrollers such as the BASIC Stamp are limited in the speed in which the value may be measured, calculations made, and action taken.

Consider the flowchart in Figure 8-5. Code must be processed at each step, which takes time. Sampling can only be performed at discrete intervals of time instead of continuously. The speed of the sampling and evaluation needed is very dependent on the dynamics of the system under control. Keeping a ball floating requires relatively fast sampling and control (around 30 updates per second). Other systems, such as our incubator, have lower demands.

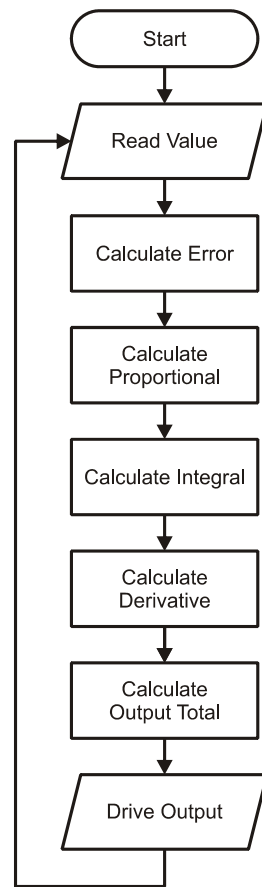


Figure 8-5
Programmatic PID
Processing Flowchart

Because of the discrete sampling, the actual error and calculations will need to be performed as shown in Figure 8-6. The value is measured at time intervals and P, I and D calculations are based on those error samples. Since the signal value is only measured at discrete intervals, the proportional drive is only updated at each sample.

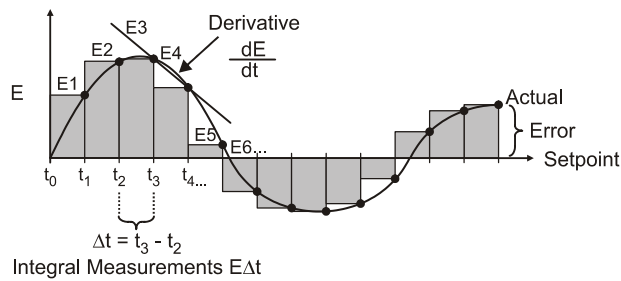


Figure 8-6
Discrete PID
measurements

Derivative drive is based on the rate of change of the error – how quickly the error changed between samples:

$$\text{DERIVATIVE} = (E_4 - E_3) / (t_4 - t_3) = \Delta E / \Delta t$$

Using discrete PID measurements, the PID formula becomes

$$C_{PID} = \text{Bias} + K_P E + K_I \sum E \Delta t + K_D \Delta E / \Delta t$$

Since we are on the subject of using light levels for PID control as used in the Floating Ball project, let's experiment with PID evaluation using light. In the next activity, we'll build a light-sensing system and use it to experiment with the effect of proportional (P), integral (I) and derivative (D) control, both singularly and in combination. The system consists of a photoresistor to sense the amount of light, and your hand, to either cast shade or allow more light onto the sensor. Your hand, then, creates disturbances to the system, and the photoresistor senses the system variable, in this case, light intensity. This system will demonstrate the PID evaluations of a system, but it is not actually representative of the response of a closed-loop system, as there is no electro-mechanical system in place to control the light level based on the PID evaluations.

Consider, though, if the light level was influenced by the amount of light coming in through a window, and we had a motor connected to window blinds. As the light level increased during the day, the motor would close the blinds in an attempt to maintain light level at the setpoint. In such a system, the output drive values would directly control the motor. For a Parallax Standard Servo motor and the BS2, reasonable drive values would range 500 to 1000. However, for this activity, we are going to simply output values that are in the same range as those of the light sensor, about 0 to 2000 or so. Our output values will not be values that are needed for actual control.

Our main focus will be on watching how changes in the sensor cause changes in the output values, even though the output values are not in themselves meaningful in the physical world. Furthermore, the activity is set up so that as the photoresistor reading increases, the output drive will also increase. This is the opposite of our incubator model – as the temperature readings increase, the heater output must decrease to keep the temperature at the setpoint.

The materials needed for this activity are listed below.

ACTIVITY #1: BIAS DRIVE

Parts required:

- (1) 10 k Ω Photoresistor
- (1) 0.1 μ F Capacitor
- (1) 220 Ω Resistor

- ✓ Add the light sensor circuit in Figure 8-7 to your board, leaving all the other components in place. The wiring diagram shows the recommended parts placement. We will be using the full heater circuit again shortly.

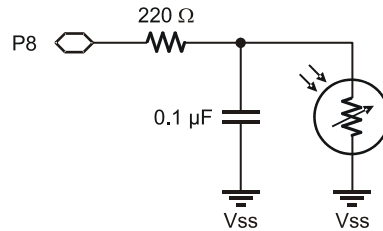
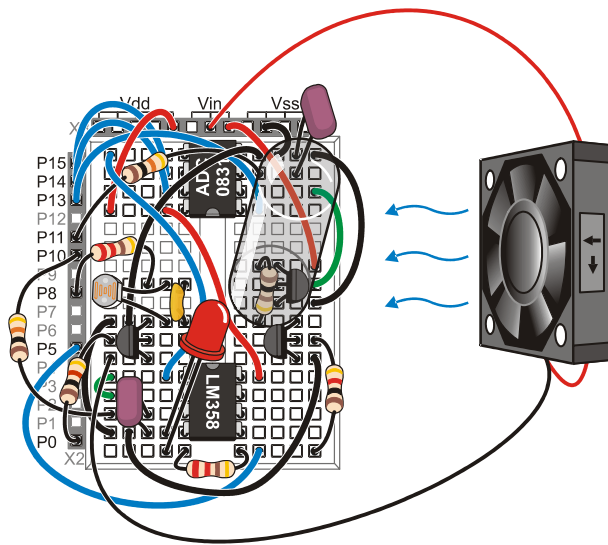


Figure 8-7
Light Sensor
Schematic (left) and
Wiring Diagram
(below)

*Leave the other parts
on the board.*



8

✓ Enter, save and run the program PIDEval.bs2 and close the Debug Terminal.

```
' -----[ Title ]-----
' Process Control - PIDEval.BS2
' Demonstrated P, I and D evaluations with StampPlot
' {$STAMP BS2}
' {$PBASIC 2.5}

' -----[ Declarations ]-----
SetPoint    VAR    Word    ' Setpoint value
Photoval    VAR    Word    ' Value of photo RC
Error       VAR    Word    ' Calculated Error
Prop        VAR    Word    ' Calculated proportional output
IntegSample VAR    Word    ' Sample Integral value (Edt)
Integ       VAR    Word    ' Calculated total integral value (SumEdt)
LastError   VAR    Word    ' Last error for deriv calcs
Deriv       VAR    Word    ' Calculated derivative output (dE/dt)
ErrorChange VAR    Word    ' Change in error for deriv calcs (dE)
Total       VAR    Word    ' Total output

SignBit     VAR    Bit     ' Holds sign for math
Counter     VAR    Byte    ' Count of samples
Samples     VAR    Byte    ' Number of samples per evaluation

Delay       CON    100     ' Delay to obtain approx times between evaluations
```

```

Photo PIN 8          ' Photo RC Network

' -----[ Initialization ]-----
' ***** Reads values from StampPlot on reset
PAUSE 1000          ' Allow connection to stabilize

' Read time between evaluations (seconds * 4)
DEBUG CR,"!READ [(drpTime),*,4]",CR
DEBUGIN DEC Samples
PAUSE 50

' Read setpoint
DEBUG CR,"!READ (txtSetP)",CR
DEBUGIN DEC SetPoint
PAUSE 50

' -----[ Main Routine ]-----
DO
  ' Adjust delay to approximate real time between evaluations
  ' Take samples and plot
  IF Samples > 2 THEN
    FOR Counter = 1 TO Samples
      GOSUB ReadPhoto
      GOSUB PlotData
      IF Samples > 4 THEN PAUSE Delay
    NEXT
  ELSE
    GOSUB ReadPhoto
    GOSUB PlotData
  ENDIF

  ' Evaluate and plot
  GOSUB CalcError
  GOSUB CalcProp
  GOSUB MarkProp
  ' GOSUB CalcInteg
  ' GOSUB MarkInteg
  ' GOSUB CalcDeriv
  ' GOSUB MarkDeriv
  GOSUB CalcTotal
  GOSUB MarkTotal
LOOP

' -----[ Subroutines ]-----

ReadPhoto:
  HIGH Photo
  PAUSE 10
  RCTIME Photo, 1, PhotoVal
RETURN
  ' Charge photoresistor's RC network Cap
  ' Allow 10 milliseconds to charge fully
  ' Measure discharge time thru photoresistor

```

```

CalcError:                                ' Calculate error
    Error = PhotoVal - SetPoint
RETURN

CalcProp:                                  ' Calculate proportional output
    Prop = Error
RETURN

CalcInteg:                                 ' Calculate integral output
    IntegSample = Error * Samples           ' Sample is error x time (samples) Edt
    SignBit = IntegSample.BIT15             ' save sign for math
    IntegSample = ABS(IntegSample)/4         ' approx 1/4 sec per sample -
                                            ' convert to seconds

    IF SignBit = 1 THEN IntegSample = IntegSample * -1 ' re-apply sign

    Integ = Integ + IntegSample              ' Sum sample to current value
    SignBit = Integ.BIT15                   ' Save sign of integ, 1 = negative
    Integ = ABS(Integ) MAX 25000             ' Limit maximum to 25000
    IF SignBit = 1 THEN Integ = Integ * -1   ' Re-apply sign
RETURN

CalcDeriv:                                 ' Calculate derivative output
    DEBUG "!O txtLE=", SDEC LastError,CR    ' Update SP with last error
    ErrorChange = Error - LastError         ' Calculate dE
    SignBit = ErrorChange.BIT15             ' Save sign, 1 = negative
    Deriv = ABS(ErrorChange)/Samples * 4     ' Divide by dt
    IF SignBit = 1 THEN Deriv = Deriv * -1   ' Re-apply sign
    LastError = Error                       ' Save Last error for this eval
RETURN

CalcTotal:                                 ' Calculate total output
    Total = Prop + Integ + Deriv             ' Sum all for total
    SignBit = Total.BIT15                   ' Save sign of total, 1 = negative
    Total = ABS(Total) MAX 25000            ' Limit maximum to 30000
    IF SignBit = 1 THEN Total = Total * -1   ' Re-apply sign
RETURN

MarkProp:                                  ' Mark propotional evaluation as line from setpoint to actual
    DEBUG "!DMOD 9",CR,
        "!LINE (PTIME), (AINVAL0), (PTIME), (AINVAL1), (Green)",CR,
        "!DMOD 13",CR
RETURN

MarkInteg:                                 ' Mark integral evaluation as rectangles
    DEBUG "!DMOD 9",CR,
        "!FREC (DATAVAL1), (AINVAL0), (PTIME), (AINVAL1), (Yellow)",CR,
        "!RECT , , , (Green)",CR,
        "!DMOD 13",CR
RETURN

```

```

MarkDeriv:      ' Mark derivative evaluations as circles and lines
DEBUG  "!FCIR (PTIME), (AINVAL1), 0.3A, (Blue)", CR,
        "!DMOD 9", CR,
        "^DWITH 3", CR,
        "!LINE (DATAVAL1), (DATAVAL4), (PTIME), (AINVAL1), (Green)", CR,
        "!SETD 4, (AINVAL1)", CR,
        "!DMOD 13", CR
RETURN

MarkTotal:      ' Plot outputs and update
DEBUG  "!O txtE=", SDEC Error, CR,
        "!O txtActual=", DEC PhotoVal, CR,
        "!O txtProp=", SDEC Prop, CR,
        "!O txtEdt=", SDEC IntegSample, CR,
        "!O txtsumEdt=", SDEC Integ, CR,
        "!O txtdE=", SDEC ErrorChange, CR,
        "!O txtdEdt=", SDEC Deriv, CR,
        "!O txtTotal=", SDEC Total, CR,
        "!O plot2.draw=line (DATAVAL1), 0, (PTIME), 0, (RED)", CR,
        "!O Plot2.Draw=LINE (DATAVAL1), (DATAVAL2), (DATAVAL1), ", SDEC
Total, ", (Black)", CR,
        "!O Plot2.Draw=LINE (DATAVAL1), ", SDEC Total, ", (PTIME), ", SDEC
Total, ", (Black)", CR,
        "!SETD 1, (PTIME)", CR,
        "!SETD 2, ", SDEC Total, CR,
        "!O butLog.Run", CR
RETURN

PlotData:      ' Plot photoresistor value
DEBUG  "^AWTH 2", CR,
        "!ACHN 0, ", DEC Setpoint, ", (RED)", CR,
        "!ACHN 1, ", DEC PhotoVal, ", (BLUE)", CR,
        "!STAT Actual=", DEC PhotoVal, CR
RETURN

```

✓ Run StampPlot macro `sic_pc_pid_eval.spm`.

Before connecting, we'll discuss some of the features of this StampPlot interface, shown in Figure 8-8. The uppermost plot shows the values read from the photoresistor, plotted with a blue line. The plot below shows the output drive, plotted in black.

The photoresistor plot has these characteristics:

- The default range is 0 to 2000.
- The Setpoint is plotted in red.

- Vertical green lines show the point at which data is evaluated for PID calculations.

The Output Drive Plot has these characteristics:

- The default range is from -2000 to 2000, making the scale half that of the upper plot.
 - A red line shows the 0 value, not the setpoint.
- ✓ Shade the sensor with your hand to a comfortable amount where you can quickly light or darken the sensor with hand movements.
- ✓ If there is considerable difference between the setpoint (red line) and your value, disconnect on StampPlot, change the Setpoint value on the plot and reset your BASIC Stamp.

8

Testing Proportional Evaluation

- ✓ In the “Evaluate and plot” section of the Main Routine, ensure that the integral and derivative subroutines are commented out as in the following:

```
' Evaluate and plot
GOSUB CalcError
GOSUB CalcProp
GOSUB MarkProp
' GOSUB CalcInteg
' GOSUB MarkInteg
' GOSUB CalcDeriv
' GOSUB MarkDeriv
GOSUB CalcTotal
GOSUB MarkTotal
```

- ✓ Ensure that "Evaluation Time" is selected to 4.
- ✓ Press Reset on the Board of Education and reset the plot.
- ✓ Lighten and darken the sensor and note how often the data is sampled for calculations (green vertical lines).
- ✓ Note that the total drive is updated below.
- ✓ Note that green vertical lines in the top plot denote when calculated samples are taken. At each sample, the output, due to proportional control only, is updated on the lower plot.
- ✓ After about 40 seconds of data, change the sample time to 0.5 seconds and reset the BASIC Stamp (not the plot).

Figure 8-8 Proportional Only Evaluation of Light Level

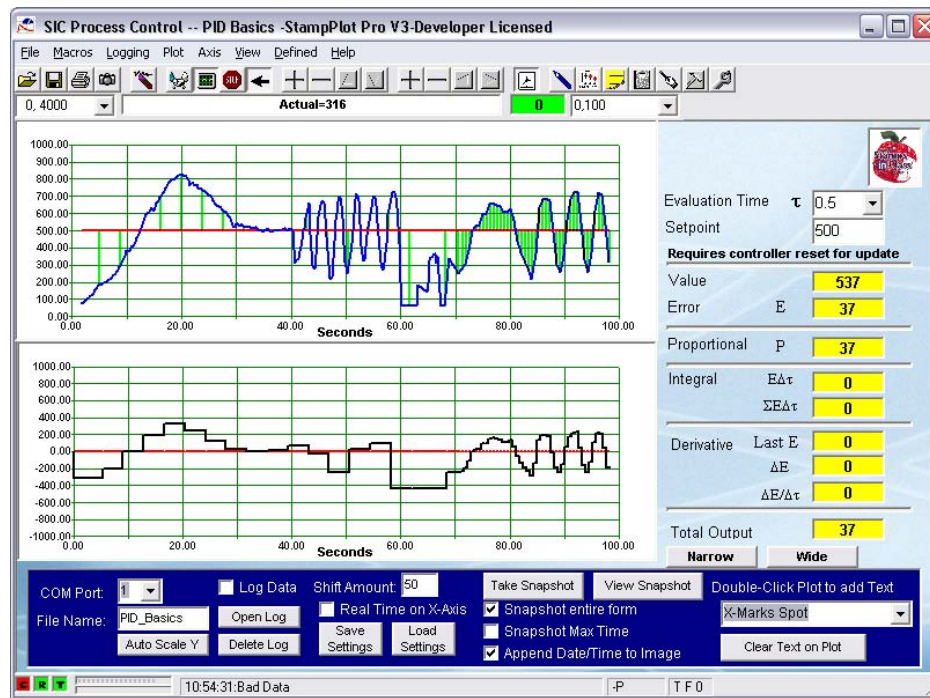


Figure 8-8 is a sample of the plot. Note that data is sampled and plotted much more often than error is calculated and total output is calculated and plotted. Time between samples is approximate. At each sample time, the error is calculated, and the proportional amount is plotted on the bottom plot. This program and macro have been designed so that the error is the actual value minus the setpoint. The greater the error, the greater the output value being plotted. In this case, output is proportional to the error.

Note that with slow sampling, the output is only a rough approximation of the actual curve. The rapid changes in signal are barely noticeable in the output. As the sampling rate increases, the output matches the error much closer.

The code reveals how the error is calculated and the proportional output calculated:

```
CalcError:
    Error = PhotoVal - SetPoint
RETURN

CalcProp:
    Prop = Error
RETURN
```



Error in Error? In the classical PID formula, as illustrated in Figure 8-1, error is calculated as:

Error = Setpoint – MeasuredValue ' Error per Figure 8-1

However, for this program we are using the expression

Error = Photoval – Setpoint ' Error used in PIDEval.bs2

In effect, errors that would be negative in the classical formula, are positive in our PIDEval program. This was done only for this activity, in order to show the output drive increasing as the photoresistor readings increase. That is, for photoresistor values greater than the setpoint, we would like to see a positive output drive, for which we need a positive error.

In the next activity, using the incubator model, we will again use the classical formula, which will produce a negative error when the temperature is greater than the setpoint.

8

The total output is the sum of the P, I, and D outputs, but in this case, Integral and Derivative will be zero since those calculations are not being performed. Since the limit of the BASIC Stamp calculations is $\pm 32,000$, code has been added to ensure the maximum does not exceed 25,000. The BASIC Stamp can't perform all math operations on negative numbers and provide results we would recognize. When working with division, MIN, MAX, and some other operations, they need to be performed on positive values only. To do this, the sign of the value is saved (Bit 15 = 1 if negative), the absolute value is used, and the sign re-applied.

```
CalcTotal:
    Total = Prop + Integ + Deriv
    SignBit = Total.BIT15
    Total = ABS(Total) MAX 25000
    IF signBit = 1 THEN Total = Total * -1
RETURN
```

Currently, the proportional control is simply equal to the error: $P = E$. In actuality, the error is multiplied by some constant, gain (K_p), to adjust how much drive is applied for a set amount of error: $P = K_p E$

- ✓ In the **CalcProp** subroutine, multiply the Error by 4 and retest.
- ✓ Compare the output plot to your previous for the same amounts of error. Notice the four-fold increase in output action taken for the same error as before.

In this activity, though there is an output based on error, there exists no feedback that would control the window blinds in an attempt to maintain the light level. But you can imagine that based on the amount of error, and therefore the proportional output, in an actual system, the higher the magnitude of the error, the more control action that would be taken.

Testing Integral Evaluation $\Sigma E\Delta t$

In this section we will test the integral control based on error over time. Since the BASIC Stamp does not have continuous internal timers, all times for samples and evaluations are approximations.

- ✓ In the code, comment out the proportional subroutines and uncomment out the integral ones:

```
' GOSUB CalcProp
' GOSUB MarkProp
GOSUB CalcInteg
GOSUB MarkInteg
```

- ✓ Download the code and close the Debug Terminal.
- ✓ Set the time between samples to 4 seconds.
- ✓ Set the control plot range to +/- 32,000 by clicking the "Widen" button several times.
- ✓ Increase the time of the plot to 200 seconds.
- ✓ Connect on StampPlot.

Set 1

- ✓ Reset the plot, reset the BASIC Stamp, and slowly darken and lighten the sensor over 30 seconds or so.
- ✓ Partially shade the sensor to a light level equal to the setpoint, and hold for 10 seconds.

Set 2

- ✓ Allow full light to fall on the sensor for 30 seconds.

Set 3

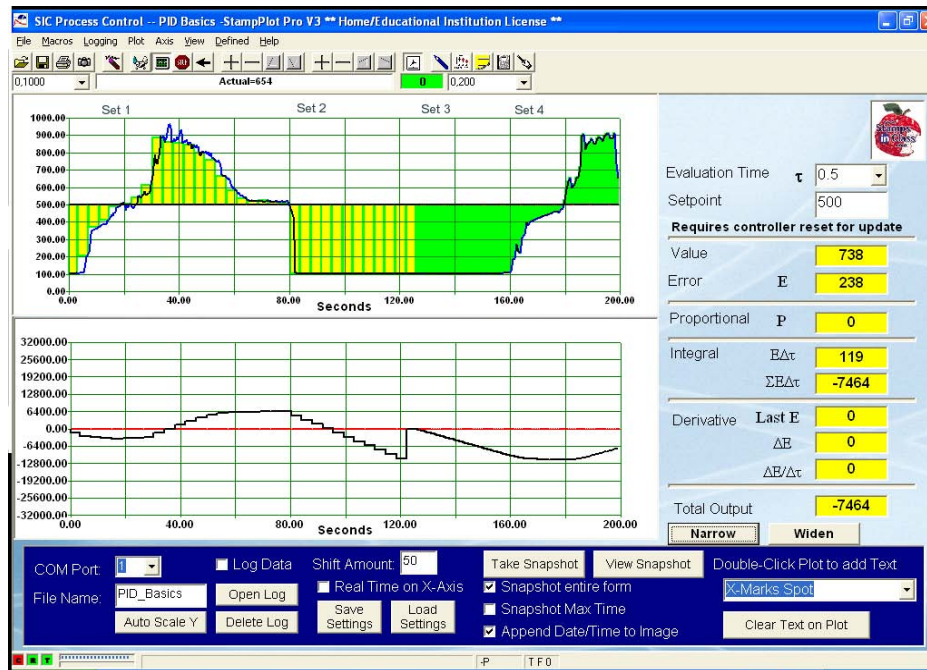
- ✓ Change the time between samples to 1 second and reset the BASIC Stamp.
NOTE: A reset will cause the integral value to be reset to zero. This is expected.
- ✓ Continue to allow full light to fall on the sensor.

Set 4

- ✓ Gradually increase and decrease the light levels.
- ✓ Stop plotting (F6).

Figure 8-9 is a plot of our results for discussion.

Figure 8-9 Integral Only Evaluation of Light Level



Set 1

At reset, the amount of integral control is zero because nothing has been added yet. With each sample, the amount of error and change in time is calculated and added to the total. Large error with respect to time causes large incremental changes in output. As long as the error is negative, the output becomes increasingly negative. It is not until the error goes positive that the output will be reduced. With sufficient positive error, the output will become zero, and then become increasingly positive. Note what area is actually used at each sample. Sometimes it is very representative of the curve, and sometimes not, based on our sampling rate.

Set 2

With the value at the setpoint, the integral value remains the same since nothing is added or subtracted.

Sets 3 and 4

In Set 3, full light was allowed to fall for a while with 4-second control sampling. Note the slope of the integral control building. In Set 4, when the time was changed to 1 second, the slope of the output was relatively the same but the output change was smoother. Integral is $E\Delta t$. Over all, the error and the total time stayed the same, and the integral value grew at the same rate. The only difference was the total time was calculated and added much more frequently in Set 4.

For example, with 4-second evaluation, with an error of -400 , each integral sample added -1600 or $(-400)(4 \text{ seconds})$. Over 12 seconds, the total would be $(3 \text{ samples})(-1600) = -4800$. With 1-second evaluations, each sample would be $(-400)(1 \text{ second}) = -400$. Over 12 seconds, the total or sum would be $(-400)(12) = -4800$.

Looking at the code for the integral control:

```
CalcInteg:                                ' Calculate integral output
  IntegSample = Error * Samples             ' Sample is error x time (samples) Edt
  SignBit = IntegSample.BIT15              ' save sign for math
  IntegSample = ABS(IntegSample)/4          ' approx 1/4 sec per sample -
                                          ' convert to seconds

  IF SignBit = 1 THEN IntegSample = IntegSample * -1 ' re-apply sign

  Integ = Integ + IntegSample              ' Sum sample to current value
  SignBit = Integ.BIT15                   ' Save sign of integ, 1 = negative
  Integ = ABS(Integ) MAX 25000             ' Limit maximum to 25000
  IF SignBit = 1 THEN Integ = Integ * -1   ' Re-apply sign
RETURN
```

Since we want to see data between evaluations, 1 second's worth of data collection takes 4 samples (allowing us to see the data between evaluations). So that the integral value for the sample is 4 times too large and must be divided. Again, signed math is undesirable, so the sign is saved and re-applied. The sample value is then added to the integral total while ensuring it does not become too large.

Again, just as in proportional control, the output would be driving the system, but in this case, it is based on the magnitude and duration of the error. The longer that error exists, the greater the output will become. Large amounts of error for extended periods of time can cause integral values to reach very high values. This 'windup', problems associated with it, and how integral control can be used in the operation of the system will be explored more when the incubator is used.

As you can see, integral output control can build very quickly and easily masks other control action. The amount of integral output may be reduced by applying a very small gain (K_i) such as 0.1 to the calculation.

- ✓ In the BASIC Stamp code, divide the Integral Sample by 10 for a gain of 0.1 and retest.

```
IntegSample = ABS(IntegSample)/4 / 10
```

Testing Derivative Evaluation: $\Delta E/\Delta t$

In this section we will test the derivative evaluation based on change in error with respect to time.

- ✓ In the code, comment out the Integral subroutines and uncomment the Derivative ones:

```
' GOSUB CalcInteg
' GOSUB MarkInteg
GOSUB CalcDeriv
GOSUB MarkDeriv
```

- ✓ Download the code and close the Debug Terminal.
- ✓ Set the time between samples to 2 seconds.
- ✓ Set the control plot (lower) range to +/- 1000 through the use of the "Narrow" button.
- ✓ Connect on StampPlot.

Set 1

- ✓ Reset the plot, reset the BASIC Stamp, and slowly darken and lighten the sensor over 30 seconds or so.

Set 2

- ✓ Change the light level fairly rapidly for about 10 seconds.

Set 3

- ✓ Hold the light level constant above the setpoint.

Set 4

- ✓ Change the evaluation sample time to 0.5 seconds and reset the BASIC Stamp (not the plot).
- ✓ Slowly darken and lighten the sensor over 20 seconds.

Set 5

- ✓ Quickly lighten and darken the sensor over about 10 seconds.
- ✓ Close the connection (F6).

Figure 8-10 is a plot of our results for discussion.

Set 1

As the signal gradually rises, the slope of the error (line connecting dots) is positive, and the output is positive. As the signal gradually falls, the slope of the error is negative and so is the output. It does not matter if the actual value is above or below the setpoint, it is simply a matter of the direction of change.

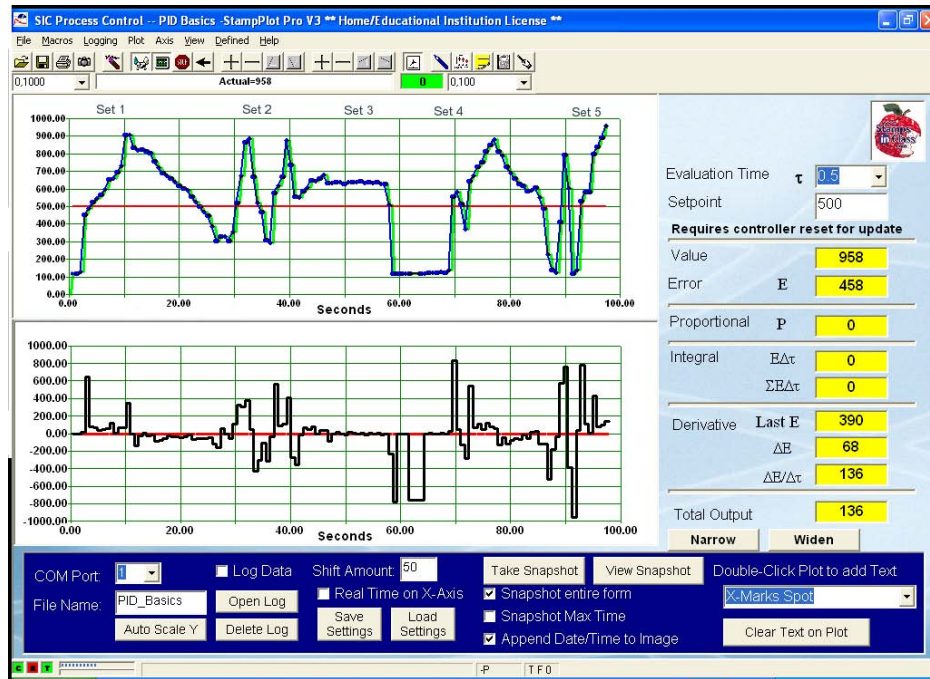
Set2

As the signal changes faster, the output increases in magnitude. The error changed more over the same sample time though the level of the signal was relatively the same.

Set 3

With the signal relatively constant, though well above the setpoint, the output is roughly zero (as well as we could hold our hand stable). Even though there existed an error, the error was not changing, therefore no output.

Figure 8-10 Derivative Evaluation Using Light



8

Set 4

With a decrease in time between evaluation sampling, roughly the same curves increased the output. With closer sampling, the maximum change of the signal is better evaluated. Note the negative output spike at 60 seconds. When the controller was reset, the value of the last error was reset to zero. When a reading was performed, it was calculated as a sudden change in error.

Set 5

With the shorter time and higher rates of error change, the output was significantly higher.

Note the measured readings: With a current error of 458, and a last error of 390, this produced a change in error (ΔE) of 68. With a change in time (Δt) of 0.5 seconds, this resulted in an output ($\Delta E/\Delta t$) of $68/0.5$ or 136.

In an actual system, just as with the automobile example, the derivative output changes in a direction and magnitude to counter the error. Acting on the window blinds, as the sensor suddenly lightened, the output would quickly move the motor with a motion relative to the change.

```

CalcDeriv:                                ' Calculate derivative output
DEBUG "!O txtLE=", SDEC LastError,CR      ' Update SP with last error
ErrorChange= Error - LastError            ' Calculate dE
SignBit = ErrorChange.BIT15               ' Save sign, 1 = negative
Deriv = ABS(ErrorChange)/Samples * 4      ' Divide by dt
IF SignBit = 1 THEN Deriv = Deriv * -1    ' Re-apply sign
LastError = Error                         ' Save Lat error for this eval
RETURN

```

Just as with proportional and integral, we can adjust the amount of force applied by derivative by setting the gain (K_D). Too much gain can cause erratic control of a system, though.

- ✓ Modify the program to multiply `Deriv` by a gain of 10, and set a sample time of 1.0 second.
- ✓ Try to hold your hand so that the output does not exceed a value of ± 400 for 20 seconds.
- ✓ Even leaving the sensor totally uncovered, the measurement fluctuates due to noise. Note the amount of output change simply based on the noise.

Adding Bias

In the examples up until now, the output swung both positive and negative for control of the system. In most cases, the output will only be in one direction. Consider the car example. On a level highway, the pedal will be in a position to supply sufficient energy to keep the car at the setpoint speed. The pedal is adjusted up or down from this center position to control the speed. In the floating ball example, the electro-magnet will have a set amount of drive (preferable 50%) when the ball is correctly in the beam. As it rises, drive will cut back; as it falls, drive will increase.

Bias is used to provide a midpoint drive that, with good engineering, should maintain a stable system at the setpoint under normal conditions. We will test adding bias to our calculations and controlling the system with that as the midpoint.

- ✓ Add a bias to your output by adding 500 to the total drive calculated in the **CalcTotal** subroutine.

```
Total = 500 + Prop + Integ + Deriv
```

Modify the code so that only proportional calculations are made and plotted.

- ✓ Download the program, use 1.0 seconds for evaluations.
- ✓ Hold the level at the setpoint.
- ✓ Cycle the light level up and down several times while trying not to exceed the range of 0 to 1000 on the output. Note that when operating at the setpoint, the output is 500. Since there is no error, the only output is based on bias.
- ✓ Have the BASIC Stamp calculate all 3 evaluations, but not plot mark any on the upper plot. Leave integral with a gain of 0.1 (/10), but remove the gains from proportional and derivative (no gain essentially means a unity gain, or gain of 1).

```
GOSUB CalcProp
' GOSUB MarkProp
GOSUB CalcInteg
' GOSUB MarkInteg
GOSUB CalcDeriv
' GOSUB MarkDeriv
```

- ✓ Experiment with light levels to try and keep the output at or near 500.
- ✓ Identify the different types of control action illustrated in the output plot.

Challenge 8-1: Predict PID Outputs**Part A**

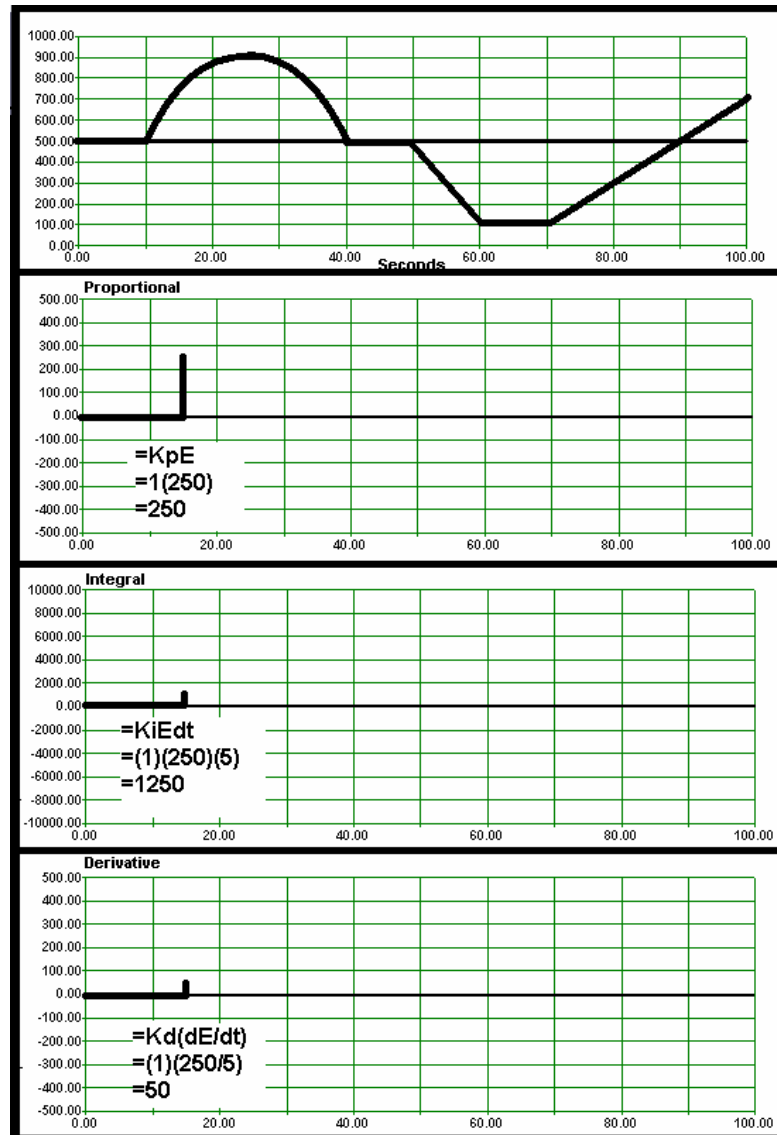
Figure 8-11 on page 273 is a plot of light levels and 3 plots for the P, I and D outputs. With a bias of 0, predict the output for each evaluation individually. Assume gains of 1 for all, and a 5 second evaluation time. The first set is done for you.

Part B

Below is data representing the systems conditions and data for PID evaluation. Calculate the output for each evaluation. The 1st 3 lines are done for you.

Table 8-1: PID Evaluation Activity Given $K_p = 2$ $K_i = 0.01$ $K_d = 4$ Setpoint = 500							
Actual	Δt	E	ΔE	$P(K_p E)$	$E \Delta t$	$I(K_i \Sigma E \Delta t)$	$D(K_d \Delta E / \Delta t)$
500	1.0	0	0	0	0	0	0
510	1.0	10	10	20	10	.1	40
490	0.5	-10	-20	-20	-5	0.05	-160
480	0.5						
470	2.0						
520	1.0						
500	0.5						
500	1.0						

Figure 8-11 PID Evaluation Activity



ACTIVITY #2: BIAS AND SYSTEM RESPONSE

Activity #1 was good to demonstrate the PID evaluations of a system, but it is not representative of the response of a closed-loop system. There was no electro-mechanical system in place to control the light level based on the PID evaluations. Consider though if we had a servo connected to blinds. As the light level increased, the servo would position the blinds in an attempt to maintain light level at the setpoint, perhaps in growing some rare plant that required a certain amount of light to flourish.

The standard hobby servo sold by Parallax requires values from the BS2 from 500 to 1000 for control. Sending the servo a pulse width value between 500 and 1000, the servo is controlled to a position between roughly 0 and 90 degrees. If the servo was coupled to a light shade, the shades could be adjusted to be fully open or fully closed. Biasing would be set so that the light level coming through the blinds was at a medium level under normal conditions (midday sun).

Now consider the value of light error in our experiments. An error of 200, 500 or a 1000 in some cases was very likely. If it got lighter outside (early morning sun shining through), the error and thus the control output would result in the servo adjusting the blinds to shut some. Conversely, darkening (cloudy day) would cause the blinds to open some.

Our output was hundreds or thousands, positive and negative. The servo requires a range of 500 to 1000 for full control. There is a disparity in the values we output and the values needed for actual control. Of course, with a little math we can span and offset the output value to match the controls input value. But it is much easier to discuss systems in terms of percentage so that no matter the ranges involved, the system is measuring, calculating, and driving in the common units of percent.

Under normal conditions, the servo will be controlled over a range of 0% for fully open to 100% fully closed. The light level will be from 0% for darkest to 100% for lightest. Under normal condition, the desired light level of 50% is achieved when the servo is driven at 50%. If the light level increases, say 10%, the servo will have its output increased by 10% to darken the area. If the light level reaches 100%, the servo will be at 100% to be fully closed.

We begin discussing the entire system in terms of percentages, and it makes it much easier to discuss in generalities and in actual system response. %Error, %Total Drive,

%Drive Bias, %Drive Proportional and so on. The new equation for the systems operation becomes:

$$\%DRIVE_{TOTAL} = \%DRIVE_{BIAS} + \%DRIVE_{PROP} + \%DRIVE_{INT} + \%DRIVE_{DERIV}$$

As discussed, a system in equilibrium is where the energy gains in that system equals the energy losses. When a system is designed, the engineers will have anticipated typical losses on the system. The bias drive is the drive used to compensate for normal losses. In terms of control, you've already performed this in earlier chapters. The amount of PWM drive was adjusted until the incubator was at or near the setpoint of 101.5 °F. This would be the bias drive for the incubator.

Bias drive provides a best-case starting point for control of a system. When a system, such as an incubator, is engineered, it is possible to design it so that a 50% drive to the heating element will control it very near the setpoint under normal conditions. This design provides for up-to 50% of the drive to be removed or up-to an additional 50% to be added for control of the system. Consider an outdoor incubator. If 50% drive on the heaters is good for a nice spring day, an additional 50% drive can help warm it on very cold winter days, or cutting back the drive by 50% on very hot summer days will help keep it at the setpoint.

For the eggs in our incubator to hatch healthy chicks, 50% drive on our heater would provide sufficient energy to maintain temperature in the incubator near 101.5 °F under our average laboratory conditions. Unfortunately, our system is not well engineered. Being a non-insulated polystyrene test tube with a small resistor for a heater, chances are that exactly 50% drive will not provide the exact amount of energy needed to meet our desired setpoint.

Note that this bias temperature may fluctuate due to room conditions, and that depending on conditions, your results may vary and 50% may be well above 101.5 °F.

$$\begin{aligned} C_{PID} &= B \\ \%Drive_{TOTAL} &= \%Drive_{BIAS} \end{aligned}$$

Parts required:

Same as Activity #1

√ Enter, save and run IncubatorPID-SP.bs2.

```

' -----[ Title ]-----
' Process control - IncubatorPID-SP.bs2
' Controls an incubator using PID control.
' StampPlot is used to perform calculations
' {$STAMP BS2}
' {$PBASIC 2.5}

' -----[ Declarations ]-----

ADC_DataIn  VAR Byte      ' Analog to Digital Converter data
TempF       VAR Word

LED         PIN 0         ' LED output pin
ADC_CS      PIN 13        ' ADC Chip Select pin
ADC_Clk     PIN 14        ' ADC Clock pin
ADC_Dout    PIN 15        ' ADC Data output
ADC_Vminus  PIN 11        ' ADC Offset Value
ADC_Vref    PIN 10        ' ADC Span reference

Heater      PIN 5         ' Incubator Heater
Fan         PIN 0         ' Incubator Fan

DriveTime   VAR Byte      ' number of seconds (x4 for PWM) to drive
PWMVal      VAR Byte      ' PWM Value to drive heater
x           VAR Byte      ' Working variable
Offset      VAR Byte      ' ADC Offset value (tenths)
Span        VAR Byte      ' ADC SPan value (tenths)

' -----[ Initialization ]-----
LOW Fan
PAUSE 1000          ' Connection Stabilization
GOSUB ReadSP        ' Read values from StampPlot

' -----[ Main Routine ]-----
DO

  GOSUB ReadADC
  GOSUB UpdateSP
  GOSUB GetSPDrive
  GOSUB ControlIncubator
LOOP

' -----[ Subroutines ]-----

' **** Set ADC Span and offset and read ADC value
READADC:
  PWM ADC_Vminus,Offset * 255/500,100
  PWM ADC_Vref,Span * 255/500,100
  LOW ADC_CS
  SHIFTIN ADC_Dout,ADC_Clk, MSBPOST,[ADC_DataIn\9]

```



```

HIGH ADC_CS
RETURN

' **** Read last total drive reading from StampPlot
'      and condition of cooling checkbox
'      Call macro routine to mark plot
GetSPDrive:
  DEBUG "!READ [(txtDtot),*,255],/,100]",CR
  DEBUGIN DEC PWMVal
  PAUSE 50
  DEBUG "!READ (chkCool)",CR
  DEBUGIN DEC Fan
  PAUSE 50
  DEBUG "!MACR .PlotData",CR
RETURN

' **** Send new value to StampPlot.
UpdateSP:
  DEBUG CR,IBIN Fan,CR
  DEBUG DEC ADC_DataIn,CR
RETURN

' **** Drive PWM for amount of drive time in 250mSec intervals
'      Drive only if fan is off
ControlIncubator:
  IF FAN = 1 THEN PWMVal = 0
  FOR x = 1 TO DriveTime * 4
    PWM Heater,PWMVal,250
  NEXT
RETURN

' **** Read Configuration information from StampPlot
ReadSP:
  PAUSE 50
  DEBUG CR,"!Read (txtLower)",CR          ' Read Lower Temp
  DEBUGIN DEC Offset
  PAUSE 50
  DEBUG "!Read [(txtUpper),-, (txtLower)]",CR ' Read Span
  DEBUGIN DEC Span
  PAUSE 50
  DEBUG "!Read (txtTime)",CR              ' time x 4 for PWM
  DEBUGIN DEC DriveTime
  PAUSE 50
RETURN

```

✓ Open StampPlot macro `sic_pc_pid_sp_calc.spm`.

Before connecting, we will discuss some features of this interface, shown in Figure 8-12.

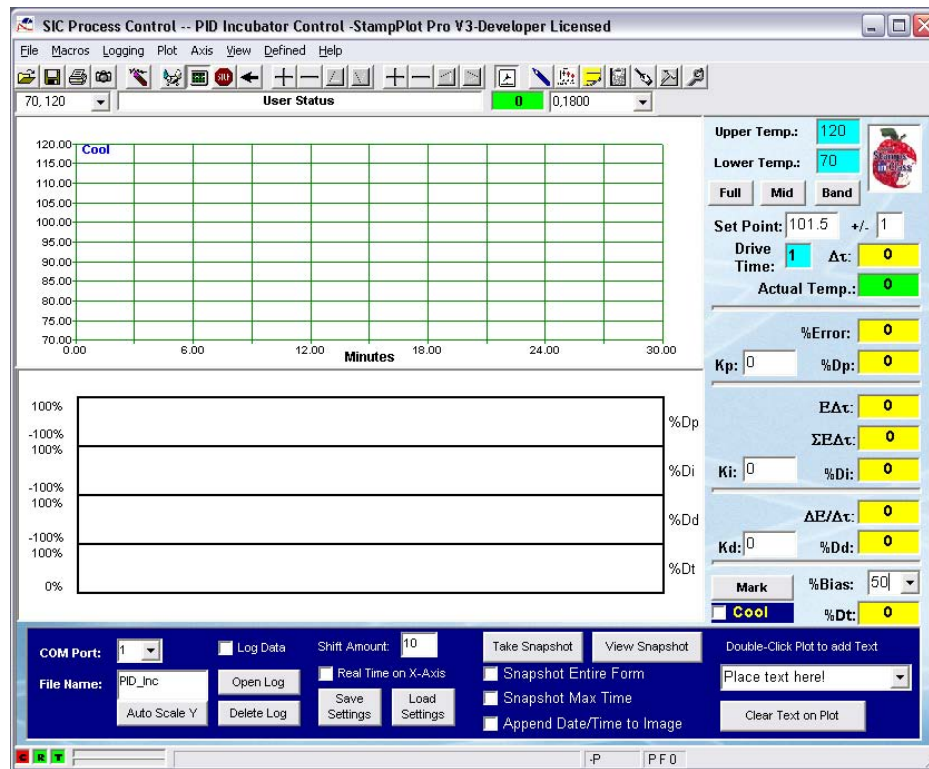
- Upper and Lower Temp.: Sets the upper and lower temperature for reading by the ADC. If this setting changed, you must reset your BASIC Stamp.
- Full, Mid, Band: Sets the span of the plot to the temperature limits, mid range for the band selected, or the control band only.
- Setpoint: Sets the temperature setpoint of the system.
- +/- : Sets the control band, such as a setpoint of 100 °F +/- 1 = 99 to 101.
- * Drive Time: Sets the seconds to apply heating drive continuously before updating.
- Δt : Actual time between samples.
- Actual Temp: Displays the current temperature of the system.
- %Error: Displays the %Error between the setpoint and actual temperatures. 200% is the maximum the StampPlot macro will calculate.
- Kp, Ki, Kd: Sets the gain constants for control.
- $E\Delta t$, $\Sigma E\Delta t$ and $\Delta E/\Delta t$: Displays values used for integral and derivative controls.
- %Dp, %Di, %Dd: Displays the %Drive from each of the control actions.
- %Bias: Sets the bias drive of the system.
- %Dt: Percent of total drive, 0 – 100.
- Cool: Energizes the fan, de-energizes the heater.
- Mark: Marks the current control settings on the plot.
- The lower graphs will plot the %Drives for P, I, D and total drive.

When plotting begins, the Message window should open, listing the current incubator settings and conditions. Data will also be logged to message and data files when logging is enabled. Let's try it now.

- ✓ Connect on StampPlot and plot.
- ✓ Verify that the Drive Control Setting match the following:

Upper Temp:	120
Lower Temp:	70
Setpoint:	101.5
Band (+/-):	1
Time:	1
Kp, Ki, Kd:	0
%Bias:	50

Figure 8-12 StampPlot PID Interface



8

- ✓ Allow the temperature to stabilize.
- ✓ Record your 50% bias temperature: _____
- ✓ Once the Temperature is stable, adjust the %Bias setting to bring the temperature closer to the setpoint.
- ✓ Click "Mark". This will mark the current drive settings in blue at the top of the plot and the current values of drive on the lower plots.
- ✓ Create a disturbance by checking "Cool". This will energize the fan and turn off the heater. For uniformity in disturbances, the cooling should be turned off once the temperature drops 5 °F.
- ✓ Allow the temperature to return to a stable temperature.

Analyzing our run in Figure 8-13, we see that it required about 5 minutes to stabilize at the 50% bias temperature of around 98 °F. Changing the %Bias to 60% and then to 55%, the temperature slowly drifted and stabilized closer to the setpoint.

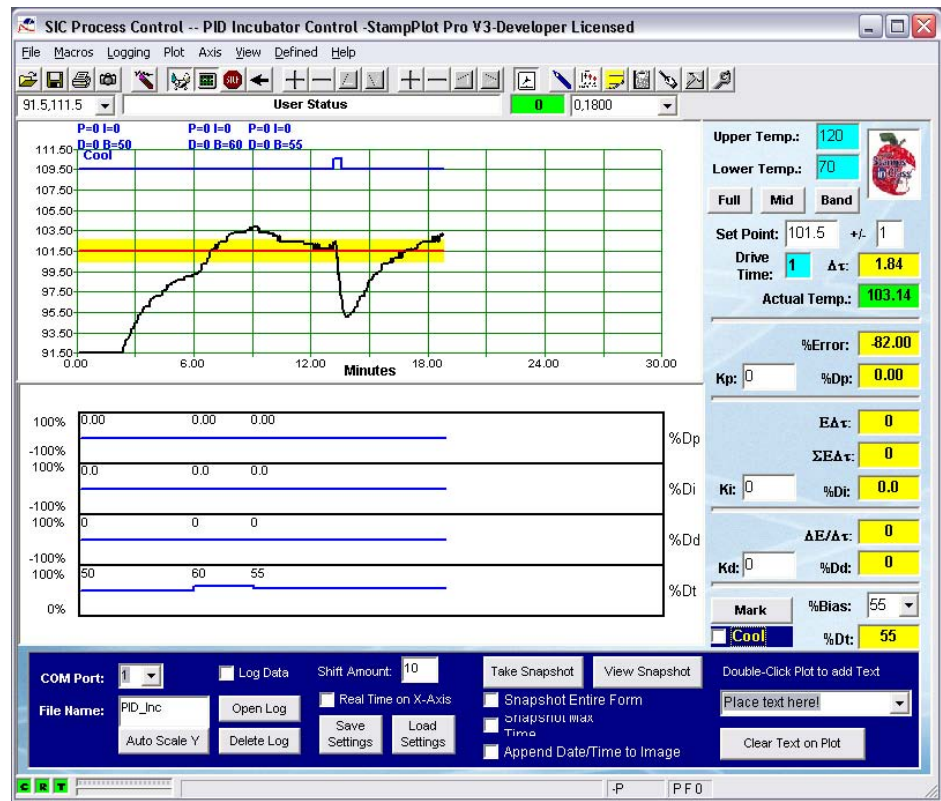
Is this system acting as a closed-loop? Not really. Just as in Chapter 6, we are manually manipulating the drive to control the system. As seen from the bottom graphs, the only time that the %Dt (Drive Total) changes is when we adjust the bias. Following a disturbance at 13 minutes, the system required approximately 7 minutes to recover and stabilize once again at the setpoint, though slightly higher - the room's winter heater may have raised temperature in the room, slightly changing conditions.

At 55% drive, the energy added by the heater equals the amount of energy lost at the setpoint to maintain the system stable. But, if the air around the incubator becomes cooler, greater energy escapes resulting in a lower stabilized temperature.

Program Discussion

The program IncubatorPID-SP.bs2 and macro sic_pc_pid_sp_calc.spm work together to control the drive to the heater. Upon BASIC Stamp reset, the temperature limits for monitoring and the drive time are read. With each loop, the raw data is sent to StampPlot and the PID calculations and plotting are performed. The BASIC Stamp reads the total drive percent and drives the heater based upon that value. StampPlot can perform floating point math and handle very large numbers. All integral and derivative calculations are based on the real time that they arrived. One drawback is that it takes time to send data from the BASIC Stamp, allow StampPlot to perform calculations, and accept the returning data. With a drive time of 1 second, typical times of Δt were 1.75 to 2.10 seconds, and may be slower depending on the speed of your computer. Also, the incubator will not work unless StampPlot is running.

Figure 8-13 Adjusting Bias Drive



8

ACTIVITY #3: PROPORTIONAL CONTROL AT BIAS TEMPERATURE**Parts required:**

Same as Activity #1

$$C_{PID} = B + (K_p E)$$

$$\%Drive_{TOTAL} = \%Drive_{BIAS} + \% Drive_{PROP}$$

The next evaluation in the PID equation is proportional control of the system. The amount of drive from proportional control is a direct relationship to how much error exists in the system. The greater the error, the greater the proportional drive. Error is the difference between the value we want the system to be and its measured value.

$$Error = Setpoint - Actual$$

Note that when the temperature is greater than the setpoint, it produces a negative error. This may seem arbitrary, but it's an important fact. As temperature increases, the drive must decrease to bring the system back to the setpoint. A temperature above the setpoint will produce a negative error. This is opposite of how the light sensor testing was performed. This is an example of the drive being inversely proportional. The output drive is proportional to the error, but in an opposite direction.

As mentioned, we will work in percentages, so let's review the math needed. First, we need to define a band of temperature control. How about 100 °F +/- 0.5 °F? This defines an allowable temperature band of 1.0 °F.

For a temperature error of +0.3 °F:

$$\%Error = E / 1.0 \text{ °F} \times 100\% = 0.3 \text{ °F} / 1 \text{ °F} \times 100\% = 30\%$$

For -0.8 °F?

$$\%Error = E / 1.0 \text{ °F} \times 100\% = -0.8 \text{ °F} / 1 \text{ °F} \times 100\% = -80\%$$

Quite simply, the %Drive due to proportional control is the proportional gain (K_p) times the %Error. A gain of 1 with a percent error of 30 yields a drive change of 30%!

$$\%Drive_{PROP} = K_P \%E = 1 \times 30\% = 30\%$$

With a gain of 3, and an error of 30%:

$$\%Drive_{PROP} = K_P \%E = 3 \times 30\% = 90\%$$

Note that the greater the error, the greater the drive due to proportional control. Consider what this means to our incubator. Let's assume a bias of 50% only got the temperature to 99.7F. This would produce an error of +30% for a total drive of 80% (with a K_P of 1).

$$\%Drive_{TOTAL} = \%Drive_{BIAS} + \% Drive_{PROP} = 50\% + 30\% = 80\%$$

With a PWM duty cycle of 80%, what will the temperature in the incubator do? Increase. As the actual temperature approaches the setpoint, what happens to the proportional drive? It becomes less and less, backing off on total drive. Consider when temperature is at 99.9 °F. The error is +10%. What does this do for total drive?

$$\%Drive_{TOTAL} = \%Drive_{BIAS} + \% Drive_{PROP} = 50\% + 10\% = 60\%.$$

Notice that as the temperature approaches the setpoint, the drive is less. This may result in the temperature stabilizing below the setpoint, which is called steady-state error.

Proportional Band

Let's consider one more term: Proportional Band. Actually, we've already discussed all there is to know. It's simply another way of looking at the gain. Proportional band defines the percentage of the control band over which the system will take proportional control action.

Consider Figure 8-14a, which has a proportional band of 100%, and a K_P of 1. It shows that 100% of the drive is controlled over 100% of the control band. This is known as a 100% Proportional Band of Control. Gain is 1 because it is:

$$K_P = \Delta\text{Output}/\Delta\text{Input} = 100\%/100\% = 1$$

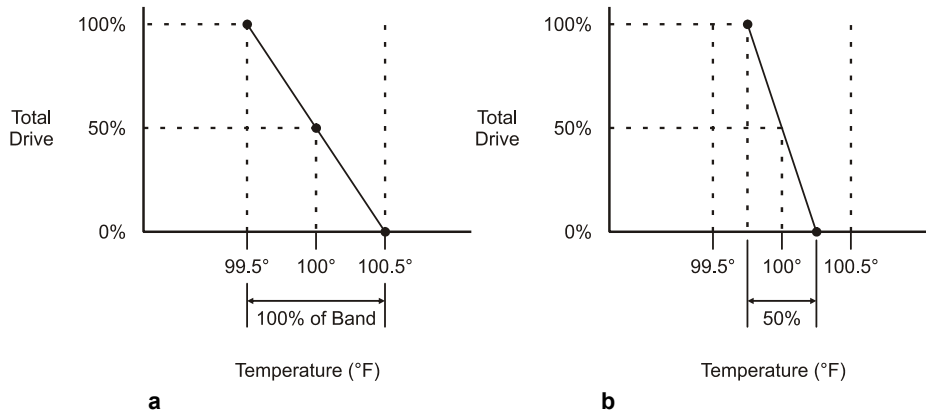


Figure 8-14: 100% Proportional Band (a) and 50% Proportional Band (b)

Now consider Figure 8-14b. 100% of the drive is controlled over only 50% of the control band. This is termed a 50% Proportional Band and has a gain of:

$$K_P = \Delta\text{Output}/\Delta\text{Input} = 100\%/50\% = 2$$

Gain and Proportional Band are actually inverses of one another. They are two different ways of discussing the same concept.

Applying Proportional Drive

It's time to apply the theory to practice and see the response of the system. For this run we want to be at the temperature that relates to a 50% bias drive.

- √ Connect and plot with the following setting to find the 50% bias temperature:

Upper Temp:	120
Lower Temp:	70
Setpoint:	101.5
Band (+/-):	2
Time:	1
Kp, Ki, Kd:	0
%Bias:	50
- √ Set the system setpoint to the current temperature, rounding to the nearest whole temperature ($96.9 = 97$).
- √ Click the "Mid" button to scale to plot.

Run #1: Kp = 0.5

- √ Set Kp = 0.5
- √ NOTE: Text box values are not updated until you press "Enter", or move to another control.
- √ Turn on fan and run until the temperature drops 5 °F, then turn off.
- √ Monitor the control action until fairly stable.

8

Run #2: Kp = 1.0

- √ Set Kp = 1.0
- √ Enable cooling for a 5 °F disturbance.
- √ Monitor the control action until fairly stable.

Run #3: Kp = 2.0

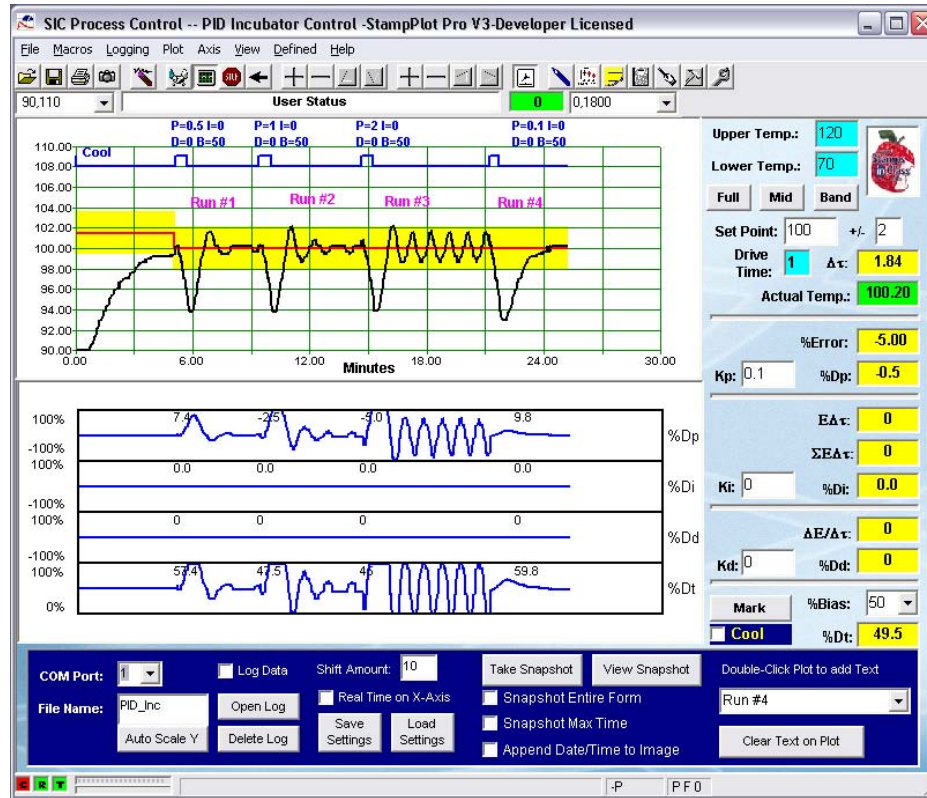
- √ Set Kp = 2.0
- √ Enable cooling for a 5 °F disturbance.
- √ Monitor the control action until fairly stable.

Run #4: Kp = 0.1

- √ Set Kp = 0.1
- √ Set Band = 5
- √ Enable cooling for a 5 °F disturbance.
- √ Monitor the control action until fairly stable.

Figure 8-15 is a plot of our results.

Figure 8-15 Proportional Control Testing



Analysis

Prior to Run #1 we can see the temperature drifting slowly up due only to a 50% drive from bias.

Run #1: $K_p = 0.5$

Note that following the disturbance, the temperature rose quickly to the setpoint of 100 °F. Looking at %Dp and %Dt in the lower plots, we can see how the amount of

proportional drive was inversely proportional to the error and was added to total drive. The temperature oscillates briefly around the setpoint prior to stabilizing. This response is known as "Critically Damped" because the value quickly settles into the setpoint with few oscillations.

Run #2: $K_p = 1.0$

For a similar disturbance, the temperature rose and overshoot by approximately the same amount as Run #1, but required more oscillations before settling into the setpoint. Compare the amount of %Dp for the same error as Run #1. With a higher gain, there is more forceful action. Due to the high number of oscillations, this system is "Under Damped". With a gain of 1, the system is operating with a 100% proportional band. That is, full control of the output occurs over full range of the control band. Note that at about time 11, the signal is at the top of the control band (100% of band). The output is at the bottom of the drive output (0% due to 50% - 50% proportional). Similarly, when temperature is at the bottom of the band (about time 10.5), the output is at maximum (100% due to 50% bias + 50% proportional).

8

Compare this to the drive in Run #1. The output was 100% when the temperature was at 96 °F, twice the width of the control band. Run #1 has a proportional band of 200%.

Run #3: $K_p = 2$

With a gain of 2.0, more forceful action is taken based on the same error. The temperature continually oscillates around the setpoint. This system is termed "Unstable". If gain is increased much more, the system would be operating in an on-off mode cycling between 0% and 100% as small errors drive the output one way and then the other.

Run #4: $K_p = 0.1$

With a very low gain, the temperature slowly drifts up and settles into the setpoint because of the very low drive added due to error. This system is "Over Damped".

Challenge 8-3: Proportional Calculations

1. Draw a 200% proportional band curve for a system with a setpoint of 110 °F and a band of ± 2 °F.
2. Calculate the proportional gain.
3. Based on the graph, what would the output be at 111 °F, 106 °F, and 115 °F?

ACTIVITY #4: PROPORTIONAL CONTROL NOT AT BIAS TEMPERATURE

Parts required:

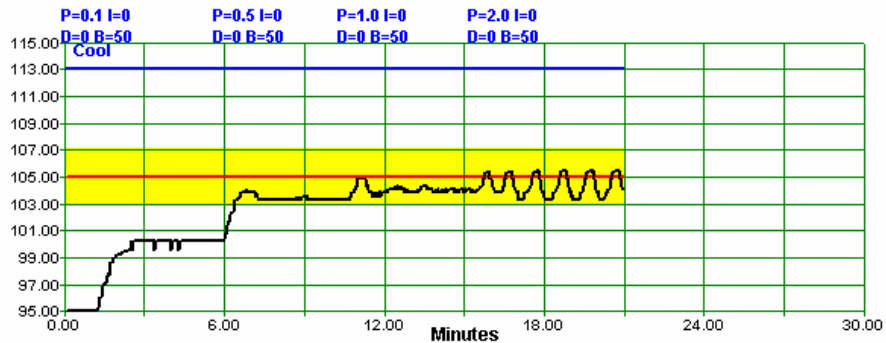
Same as Activity #1

While it seems pretty cut and dry that we can control at the setpoint using proportional control, what happens when not operating at the temperature defined by a 50% bias setting, or when a long-lasting disturbance affects the system equilibrium? In this activity we will explore controlling at many degrees away from where bias alone would be operating.

- ✓ Reset StampPlot.
- ✓ Connect on StampPlot.
- ✓ Set the Setpoint to a temperature distant from the temperature obtained with a 50% bias. In our case, we chose a temperature of 105 °F being 5 degrees away from the 50% operating temperature.
- ✓ Set a band of ± 2 .
- ✓ Set K_p to 0.1 and allow to stabilize.
- ✓ Repeat with gains of 0.5, 1.0 and 2.0.

Let's analyze the control response as seen in Figure 8-16.

Figure 8-16 Proportional Response not at 50% Bias Temperature



With a K_p of 0.1, temperature stabilized several degrees away from the setpoint. With bias at 50%, the proportional drive had to be positive to drive temperature up. But the

closer the actual temperature got to the setpoint, the less drive that proportional supplied. If the temperature actually reached the setpoint, the total drive would be 50 %, all due to bias. However, from previous activities, we found that 50% drive would result in a temperature of 98 °F.

In order for proportional drive to have control, there must be error. With a K_p of 0.1, the amount of error allowed a stable temperature around 100 °F. This error is called "Steady-State Error."

When K_p was increased to 0.5, this increased the proportional drive for the same error resulting in a temperature even closer to the setpoint. At 1.0, the steady-state temperature was even closer, but oscillations are beginning. Finally, with a K_p of 2.0, it drove even closer but had heavy oscillations, and as can be seen, the midpoint of the oscillations are still below the setpoint.

If bias drive is not correct for the setpoint, proportional drive alone cannot maintain temperature at the setpoint. Proportional drive is good for driving the system back towards the setpoint, but there needs to exist error for proportional drive to have effect.

Of course, the logical thing to do would be to adjust the bias to meet the conditions. In this case changing bias to 70% may provide sufficient energy. But what happens when the conditions change again? We are back to manual control at the setpoint.

Challenge 8-4

An oil flow system is designed and it is found that 80% drive is needed to drive the pump for the required flow rate.

1. If bias were set to 50%, would the flow stabilize above or below the setpoint?
2. How would increasing the proportional gain affect the steady state flow rate?
3. How would decreasing the proportional gain affect the steady state flow rate?

ACTIVITY #5: PROPORTIONAL-INTEGRAL CONTROL

Parts required:

Same as Activity #1

$$C_{PID} = B + (K_p E) + (K_i \Sigma E \Delta t)$$

$$\% \text{Drive}_{\text{TOTAL}} = \% \text{Drive}_{\text{BIAS}} + \% \text{Drive}_{\text{PROP}} + \% \text{Drive}_{\text{INT}}$$

So far we've looked at what occurs when quick disturbances occur to our system in equilibrium. Proportional control may be used to drive the temperature back to the desired setpoint. But what happens when the disturbance affects the equilibrium of our system over a long period of time? At the end of the last experiment, we saw what occurs when the bias drive is not sufficient to make-up for average losses. Because some error must exist for proportional drive, the setpoint temperature cannot be maintained.

Integral control can be used to drive-away any error remaining due to long lasting disturbances or imbalances in the system. These errors may be from additional losses or gains of energy that remain for a long period of time. Consider our incubator. We found a bias temperature at which a 50% bias drive was sufficient to make up for the losses in the system, maintaining it in equilibrium.

But what would happen if the room temperature were 10 degrees cooler? Continuous system losses would be higher. The 50% bias drive will be insufficient to maintain the temperature, and proportional drive will respond to the error in an attempt to drive the system back toward to the setpoint with a steady-state error remaining. The system will stabilize at a temperature below the desired setpoint. Over time, integral control can be used to drive away this error, allowing the temperature to reach the setpoint.

Integral drive is also used when a slow approach with long stabilization times are needed to ensure no overshoot. Consider the example of cooking soup. After cooking a bit, you taste, add an amount of salt you feel appropriate for what you would like the final taste to be. Do you taste immediately and add more? No, you wait a while to allow the salt to blend in, then taste, and add a bit more until you finally reach your desired taste. What if too much salt is added? Cutting back is a bit more difficult!

An industrial example may be that of adding pigment to paint for a desired color. Electronic circuitry monitors paint color and gradually adds pigment until the desired color is reached. In integral drive the amount of error is integrated over time. The larger the error and the longer it lasts, the greater the integral drive will be.

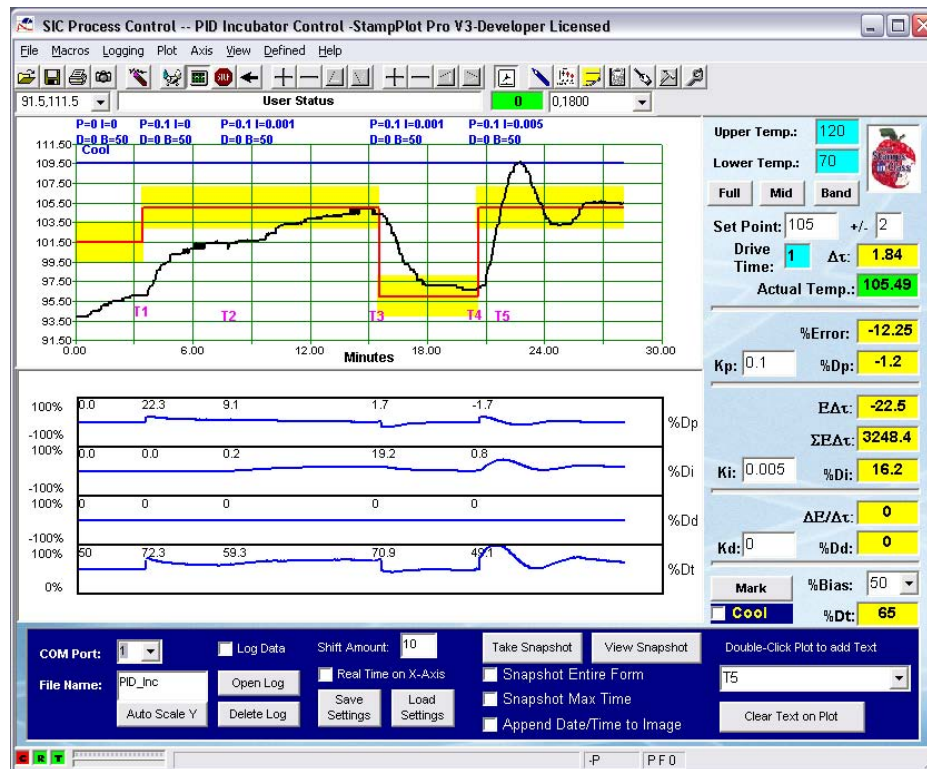
- √ Connect and plot.
- √ Update the settings with the following:

Upper Temp:	120
Lower Temp:	70
Setpoint:	101.5
Band (+/-):	2
Time:	1
Kp:	0
Ki:	0.000
Kd	0
%Bias:	50

8

- √ Allow the temperature to stabilize. Record this temperature here:_____.
- √ Set the setpoint approximately 10 degrees above this temperature.
- √ Change Kp to 0.1 and allow temperature to stabilize.
- √ Change Ki to 0.001 and allow temperature to stabilize.
- √ Change the setpoint to the temperature recorded above, and allow it to stabilize.
- √ Change the setpoint back to the +10 value.
- √ Change Ki to 0.005 and monitor.

Figure 8-17 Proportional - Integral System Response



Looking at our results in Figure 8-17, at Time 1 (T1 marked on plot), the system is fairly stable at 96 °F on only 50% bias. At this point, the setpoint is changed to 105 °F, K_p is set to 0.1, and the error causes temperature to rise.

At T2, the system has stabilized with proportional control at a temperature of 101.5 °F (pure coincidence). From the lower plot, we can see that the steady-state error is 9.1% for proportional drive. Integral control is introduced with a K_i of 0.001.

At T3, the system has almost reached the setpoint. Note that proportional drive is nearly 0 (1.7%) and integral drive is 19.2, for a total drive of 70.9. Because integral plus bias

are adding sufficient drive to be at setpoint, the error is nearly 0, therefore proportional drive is nearly 0. Also at T3, the setpoint is changed to the 50% drive temperature, which in our case was 96 °F. The drive built up by integral drive must again be removed based on the error and time.

At T4, the temperature is nearing the setpoint as integral is reduced. Integral drive is nearly 0 (0.8%), and total drive is 49.1%, with a small error providing -1.7% for proportional. The setpoint is returned to 105 °F with a K_I of 0.005, five times larger than previously. The integral drive builds quickly to cause an overshoot. Integral must again be reduced by error to bring temperature back down and causing undershoot. Hunting can occur with integral control, and the durations will be much longer based on the slow integration times.

At T5, note that the temperature is crossing the setpoint. What happens with the integral drive at this time? It is at this point that it turns and begins to be reduced. Depending on how long, and what magnitude of error, integral drive values can become very large.

8

Integral drive can be thought of as a dynamic bias. Based on error, integral drive will adjust to get the steady-state controlled variable to the setpoint. When integral drive has very low gain, the change in %Drive_{INT} will be very small, allowing proportional drive to respond to correct temperature. Large values of K_I are usually undesirable as they can cause the controlled output to be driven faster than the system can respond, leading to high overshoot and oscillations.

It is also important to limit the maximum value that $\Sigma E\Delta t$ reaches. A long lasting disturbance, such as a leaving the incubator door open overnight, could cause the value to reach extremely high values. Once the door is closed, the drive will be excessive, based on the high integral drive, and will require a very long time for the positive error to be driven away. This will lead to very high temperatures for a very long time. This is effect is called "Integral Windup." Feel free to try this test by removing the cover from the incubator, and allowing it run for 5 minutes, then replace the cover.

Challenge 8-5: Integral Control Testing

Can a system be controlled using integral control alone? Set %Bias to 50% and K_p to 0 and test the control action, and then discuss the results.

ACTIVITY #6: PROPORTIONAL-DERIVATIVE CONTROL**Parts required:**

Same as Activity #1

$$C_{OPID} = B + (K_p E) + (\Delta E / \Delta t)$$

$$\%Drive_{TOTAL} = \%Drive_{BIAS} + \%Drive_{PROP} + \%Drive_{DERIV}$$

Derivative control responds to a CHANGE in the error. The fundamental premise determining derivative drive assumes that the present rate of change in the error signal will continue into the future unless action is taken. Derivative drive, when properly tuned, allows a system to rapidly respond to sudden changes and react accordingly. It can also help damp the system to limit oscillations, and drive to limit the effects of short disturbances.

To achieve the best plot response, we will work around the stable temperature for 50% bias drive. Due to dealing with error over time, the smaller the resolution of our temperature, the better, so the upper and lower temperatures will be set accordingly.

- √ Connect and plot. Adjust your settings to the following:

Upper Temp:	50% Bias Temperature + 10 (e.g. 100 + 10 = 110 °F)
Lower Temp:	50% Bias Temperature – 10 (e.g. 100 – 10 = 90 °F)
Setpoint:	50% Bias Temperature (e.g. 100 °F)
Band (+/-):	2
Time:	1
Kp:	1
Ki:	0.000
Kd	0
%Bias:	50

- √ Allow temperature to stabilize.

Run #1

- √ Introduce a disturbance and monitor the resulting action.

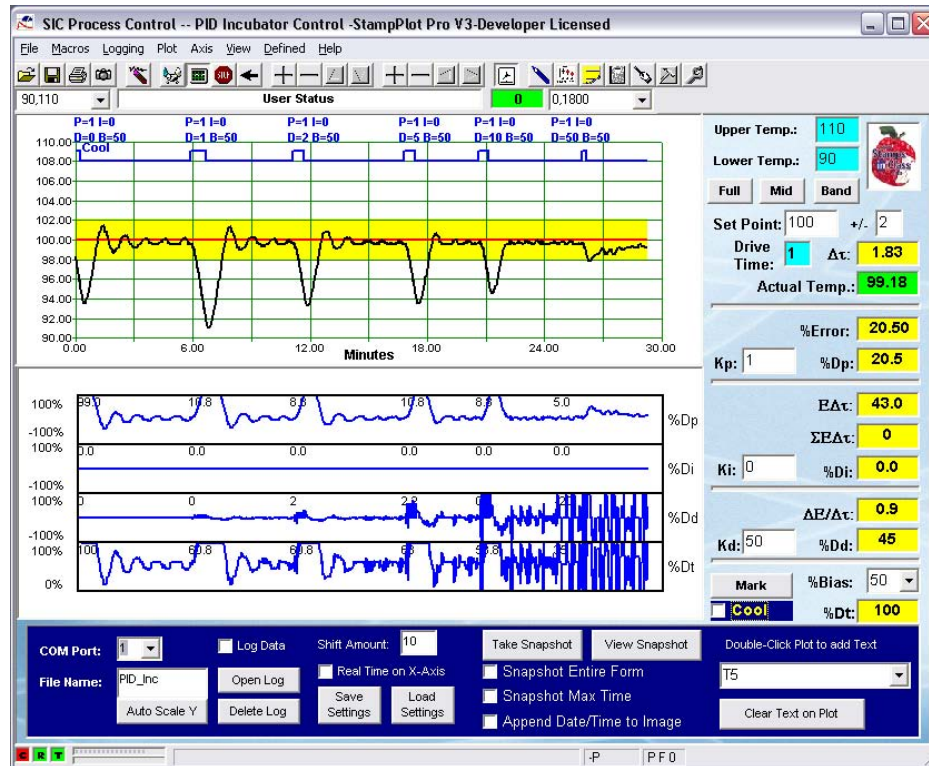
Run #2

- √ Set Kd to 1.0, or another under-damped response that provides multiple oscillations before stabilizing.

- ✓ Introduce a disturbance and monitor the resulting action.
- ✓ Starting with 1, use progressively higher gains for derivative (Kd) until the system returns as quickly as possible with little or no overshoot.
- ✓ Try a very large gain (50) for Kd and monitor the response to a small disturbance.

How did derivative control affect the response of the system? As shown in Figure 8-18, with only proportional control, there were heavy oscillations around the setpoint. As the derivative gain was progressively increased at times 6, 12, 18 and 21, the overshoot was progressively reduced and the system came under control quicker.

Figure 8-18 Proportional-Derivative System Response



Note the response of %Dd:

- Temperature increasing → %Dd is negative.
- Temperature decreasing → %Dd is positive.
- Temperature constant → %Dd is 0.

Derivative control drives to oppose the rising or falling temperature. When K_p is changed to 10, a small change in error produces very large changes in drive. The response of the system is such that there were almost no oscillations. With Proportional-Derivative control, a high proportional gain will assist control of the system by having a fast response, and results in very little proportional error. With derivative control, the heavy oscillations associated with a high proportional gain can be damped, resulting in a very fast, stable response. But too much derivative gain can lead to very unstable control of the system. Around time 24, the gain was increased to 50. With just a little disturbance, the erratic behavior of the system can be seen. As proportional drive tried to raise the temperature to the setpoint, derivative control fought any upwards motion. The total drive cycled continually between 0% and 100% as the error went one way and then the other. If this were an oil-flow system, consider what the pump must sound like during this control!

Derivative can be very good for limiting the effect of an error. Imagine operating at the setpoint, and quick, short disturbance causes the temperature to drop suddenly. Derivative drive will rapidly increase the output to limit the amount of the temperature drop. We were not able to show this response well with our slow-responding system. Can you?

Challenge 8-6: Derivative Control Testing

Can a system be controlled on derivative drive alone?

- ✓ After establishing operation near the setpoint, set K_p , K_i to 0 and %Bias to 0.
- ✓ Set K_d to a value of your choosing.
- ✓ Provide a cooling disturbance and test control of the system.
- ✓ Allow operation for several minutes.
- ✓ Discuss the results of your testing.

Final Challenge: Incubator Tuning

Tuning a PID system involves adjusting the software parameters for each factor. The goal of tuning the system is to adjust the gains so the loop will have optimal performance under dynamic conditions. As mentioned earlier, tuning is as much of an art as it is a science. The basic procedures for tuning a PID controller are as follows. This procedure assumes you can provide or simulate a quickstep change in the error signal:

1. Turn all gains to 0.
2. Begin turning up the proportional gain until the system begins to oscillate.
3. Reduce the proportional gain until the oscillations stop, and then drop it by about 20 % more.
4. Increase the derivative term to improve response time and system stability.
5. Increase the integral term until the system reaches the point of instability, and then back it off slightly.

8

As you gain experience in embedded control, you will see that the characteristics of the process will determine how you should react to error.

Part A:

- ✓ Load and run IncubatorPID-SP.bs2.
- ✓ Use StampPlot macro sic_pc_pid_sp_calc.spm.
- ✓ Tune the PID settings for the fastest stabilization for the incubator temperature of 101.5 °F \pm 1 °F with a bias of 50% and drive time of 2 seconds. Start from a stable temperature at the setpoint, and provide standard disturbance.
- ✓ Discuss your results and capture a screenshot of control action.

Part B:

A new species of eggs have arrived!

- ✓ Re-tune your system to control the temperature at 110 °F, \pm 2 °F with a 50% bias and drive time of 1 second. Start from a stable temperature at the setpoint, and provide standard disturbance.
- ✓ Discuss your results and capture a screenshot of control action following a disturbance.

Stand-Alone Control with the BASIC Stamp

Also included in the file distributions, but not used in the activities, are the following:

- BASIC Stamp 2 program IncubatorPID-BS2.bs2
- StampPlot macro sic_pc_pid_b2_calc.spm

This macro and program pair work together for an interface, but the BASIC Stamp performs the math. This allows the routines that read StampPlot to be commented-out, and stand alone operation to be performed. Also, with StampPlot interaction, the speed is somewhat greater. But in order for the BASIC Stamp to perform the necessary calculations, some limits were imposed:

- Values for gains are based on a scalar used (to get a K_I of 0.001) for example, and only limited values for gains are possible. Drop-down boxes on the interface are updated with allowable values when the connection opens.
- Maximum errors of 200% and drives of 200% in most cases.
- Maximum integral sum of 30000.
- An "Update" checkbox on the interface must be checked to have the BASIC Stamp read any updated values (including temperature settings and drive time).

```
' -----[ Title ]-----
' IncubatorPID-BS2.bs2
' Controls an incubator using PID control with StampPlot Interface
' BASIC Stamp performs the calculations
' {$STAMP BS2}
' {$PBASIC 2.5}

' -----[ Declarations ]-----

ADC_DataIn  VAR Byte      ' Analog to Digital Converter data
TempF       VAR Word     ' Calculated temperature in tenths

LED         PIN 0         ' LED output pin
ADC_CS      PIN 13        ' ADC Chip Select pin
ADC_Clk     PIN 14        ' ADC Clock pin
ADC_Dout    PIN 15        ' ADC Data output
ADC_Vminus  PIN 11        ' Control ADC offset
ADC_Vref    PIN 10        ' Control ADC Span

Heater      PIN 5         ' Incubator Heater
Fan         PIN 0         ' Incubator Fan

DriveTime   VAR Byte     ' Amount of time to drive heater in seconds
Err         VAR Word     ' Calculated error in tenths
TempDrive   VAR Word     ' Working variable
```

```

I_Edt      VAR Word      ' Integral variables
I_Sign      VAR Bit

DriveTotal  VAR Word      ' Calculated total drive (tenths)
LastErr     VAR Word      ' Last error amount
Bias        VAR Word      ' Amount of Bias drive
Band        VAR Byte      ' Size of temperature control band

Span        VAR Byte      ' ADC Span (tenths)
Offset      VAR Byte      ' ADC Offset (tenths)

SetP        VAR Word      ' Setpoint Value (tenths)
PWMVal      VAR Byte      ' Calculated PWM value (0-255)

x           VAR Byte      ' Working variable
SignBit     VAR Bit      ' Holds calculation sign (+/-)

' The actual gain is a function of K/Scalar
' Ex: Gain Prop = Kp/Kp_Scalar = 0 to 15 divided by 5 = 0.0 to 3.0
'      in increments of 0.2 (1/5)
Kp          VAR Nib      ' Proportional Gain Constant (scaled 0-15)
Kp_scalar   CON 5        ' Scalar for Proportional Gain
Ki          VAR Nib      ' Integral Gain Constant (scaled 0-15)
Ki_scalar   CON 1000     ' Scalar for Integral Gain
Kd          VAR Nib      ' Derivative Gain Constant (scaled 0-15)
Kd_scalar   CON 1        ' Scalar for Derivative Gain

' -----[ Initialization ]-----
' Set initial values
LOW Fan     ' Cooling off
SetP = 1015 ' Temp in tenths
Band = 20   ' Temperature band in tenths
Kp = 0      ' 0-15, actual kp = kp/scalar)
Ki = 0      ' 0-15, actual ki = ki/scalar)
Kd = 0      ' 0-15, actual kd = kd/scalar)
Offset = 80 ' Lowest temp
Span = 40   ' Highest - lowest
Bias = 50   ' Bias Drive
DriveTime = 1 ' Drive time

PAUSE 1000 ' Allow connection to stabilize
GOSUB ConfigSP ' Configure StampPlot Controls
GOSUB ReadSP ' Read SP updates, comment out to disable
              ' StampPlot interactivity

' -----[ Main Routine ]-----
DO
  GOSUB ReadADC
  GOSUB CalcError

```

```

GOSUB AddBias
GOSUB CalcP_Drive
GOSUB CalcI_Drive
GOSUB CalcD_Drive
GOSUB Control_Incubator
GOSUB ReadSP          ' Comment out to disable StampPlot interactivity
LOOP

' -----[ Subroutines ]-----
' **** Sets ADC Span & Offset, Reads ADC Value, Calculates Temp F,
' Updates StampPlot for temperature
READADC:
  PWM ADC_Vminus,Offset * 255/500,100
  PWM ADC_Vref,Span * 255/500,100
  LOW ADC_CS
  SHIFTIN ADC_Dout,ADC_Clk, MSBPOST,[ADC_DataIn\9]
  HIGH ADC_CS
  TempF = ADC_Datain * Span /26 + (Offset*10)
  DEBUG CR,DEC SetP,"",          ' Send setpoint
        DEC Band, "",          ' Send Band
        DEC TempF,"",          ' Send Actual Temp
RETURN

' **** Calculates the amount of Error based on setpoint and temperature
' Based on band calculates %Error, maximum 200% in tenths
' Signbit saved and recalled to perform math on positive integers.
CalcError:
  Err = SetP - TempF
  SignBit = Err.BIT15
  Err = ABS(Err) * (1000/Band) MAX 2000
  IF Signbit = 1 THEN Err = Err * -1
  DEBUG SDEC Err,"",          ' Send %Error
RETURN

' **** Adds bias to total drive
AddBias:
  DriveTotal = DriveTotal + (Bias * 10)
RETURN

' **** Calculates %proportional drive and adds to total drive
CalcP_Drive:
  signBit = err.BIT15
  TempDrive = ABS(Err)* Kp / Kp_Scalar MAX 2000
  IF signbit = 1 THEN TempDrive = TempDrive * -1
  DEBUG SDEC TempDrive,"",          ' Send %Drive-P
  DriveTotal = DriveTotal + TempDrive
RETURN

' **** calculates Integral Drive.
' Calculates integral sample based on error * time
' Integrated error is accumulated in I_Et

```



```

CalcI_Drive:
  IF (Ki <> 0) THEN
    SignBit =Err.BIT15
    ' Scale to hold very high readings for SumEdt
    TempDrive = ABS(Err)/20
    IF SignBit = 1 THEN TempDrive = TempDrive * -1
    I_Edt = I_Edt + (TempDrive * DriveTime)
    SignBit = I_Edt.BIT15
    I_Edt = ABS(I_Edt) MAX 31000
    IF SignBit = 1 THEN I_Edt = I_Edt * -1
    TempDrive = ABS(I_Edt)/10
  ' %Integral Drive based on total integrated error and gain
  TempDrive = TempDrive * Ki / (Ki_Scalar/200) MAX 2000 '(scalar /100) * 2
  IF SignBit = 1 THEN TempDrive = TempDrive * -1
  ELSE
    Ki = 0 then reset total integrated error
    I_Edt = 0
    TempDrive = 0
  ENDIF
  ' Update StampPlot and add to total drive
  DEBUG SDEC I_Edt,"," ' Send SumEdt
  DEBUG SDEC TempDrive,"," ' Send Drive_I
  DriveTotal = DriveTotal + TempDrive
RETURN

' **** Calculate %Derivative based on change in error over change in time
' Added to total drive
CalcD_Drive:
  TempDrive = Err - LastErr
  signBit = TempDrive.BIT15
  TempDrive = ABS(TempDrive)/ DriveTime
  IF SignBit = 1 THEN TempDrive=TempDrive * -1
  IF Kd = 0 THEN TempDrive = 0
  DEBUG SDEC TempDrive,"," ' Send dE/dt
  TempDrive = ABS(TempDrive) * Kd / Kd_Scalar MAX 2000
  IF signBit = 1 THEN TempDrive=TempDrive * -1
  DEBUG SDEC TempDrive,"," ' Send Drive-I
  DriveTotal = DriveTotal + TempDrive
  LastErr=Err
RETURN

' **** Drive incubator with PWM based on Drive total
Control_Incubator:
  ' Ensure => 0 and < 1000 (100%)
  IF DriveTotal.BIT15 = 1 THEN DriveTotal = 0
  IF DriveTotal > 1000 THEN DriveTotal = 1000
  DEBUG SDEC DriveTotal,CR ' Send Drive-T
  DEBUG IBIN Fan,CR '
  ' Convert to 0-255 PWM value
  PWMVal = DriveTotal/10 * 255/100
  ' If cooling, do not heat

```

```

    IF Fan = 1 THEN PWMVal = 0
    '      Drive heater for drive time at 250mSec each
    FOR x = 1 TO DriveTime * 4
        PWM Heater,PWMVal,250
    NEXT
    '      Reset Total Drive
    DriveTotal = 0
RETURN

' **** Read StampPlot for updates
ReadSP:
    DEBUG CR,"!READ (chkSet)",CR
    DEBUGIN DEC signbit
    '      If 'UPDATE' checked, read all other values
    IF signbit = 1 THEN
        PAUSE 50
        DEBUG "!Read (txtLower)",CR
        DEBUGIN DEC Offset
        PAUSE 50
        DEBUG "!Read [(txtUpper),-, (txtLower)]",CR
        DEBUGIN DEC Span
        PAUSE 50
        DEBUG "!READ [(txtSetP),*,10]",CR
        DEBUGIN DEC SetP
        PAUSE 50
        DEBUG "!READ [(txtBand),*,20]",CR
        DEBUGIN DEC Band
        PAUSE 50
        DEBUG "!READ (txtTime)",CR
        DEBUGIN DEC DriveTime
        PAUSE 50
        DEBUG "!READ (drpBias)",CR
        DEBUGIN DEC Bias
        PAUSE 50
        DEBUG "!READ [(txtKp),*,", DEC Kp_Scalar,"]",CR
        DEBUGIN DEC Kp
        PAUSE 50
        DEBUG "!READ [(txtKi),*,", DEC Ki_Scalar,"]",CR
        DEBUGIN DEC Ki
        PAUSE 50
        DEBUG "!READ [(txtKd),*,", DEC Kd_Scalar,"]",CR
        DEBUGIN DEC Kd
        PAUSE 50
        DEBUG "!READ (chkCool)",CR
        DEBUGIN DEC Fan
        PAUSE 50
        DEBUG "!O ChkSet=0(CR)!BELL",CR
    ENDIF
RETURN

'**** Configure controls on StampPlot

```

```

ConfigSP:
'    Clear gain drop-downs
  DEBUG CR,"!O txtKp.clear",CR,
    "!O txtKi.clear",CR,
    "!O txtKd.clear",CR

'    Populate each for allowable values based on the Scalar
  FOR x = 15 TO 0
    DEBUG "!O txtKp.add = [", DEC x,"/",", DEC Kp_Scalar,",",",FORMAT,0.0]",CR
    DEBUG "!O txtKi.add = [", DEC x,"/",", DEC Ki_Scalar,",",",FORMAT,0.000]",CR
    DEBUG "!O txtKd.add = [", DEC x,"/",", DEC Kd_Scalar,",",",FORMAT,0]",CR
  NEXT

'    Update current values for settings
  DEBUG "!O txtKp=[", DEC Kp,"/",",DEC Kp_Scalar,",",Format, 0.0]",CR,
    "!O txtKi=[", DEC Ki,"/",",DEC Ki_Scalar,",",Format, 0.000]",CR,
    "!O txtKd=[", DEC Kd,"/",",DEC Kd_Scalar,",",Format, 0]",CR,
    "!O drpBias=", DEC Bias,CR,
    "!O txtBand=", DEC Band,"/",20]",CR,
    "!O txtSetp=", DEC SetP,"/",10]",CR,
    "!O txtTime=", DEC DriveTime, CR,
    "!O txtLower=", DEC Offset,CR,
    "!O txtUpper=", DEC Offset + Span,CR,
    "!O btnFull.Run",CR,
    "!BELL",CR

RETURN

```

CONCLUSION

With PID control, the actual temperature is used as feedback and three separate drive evaluations are performed to calculate the final drive output to the control element. Bias drive is used to estimate the drive needed to sustain a setpoint value under nominal conditions.

Proportional drive acts by adding an amount of drive in proportion to the amount of error that exists between the setpoint and the actual value. The higher the proportional gain, the greater the controller's response, though overshoot and oscillations are more likely. Some error must exist for proportional drive to act, often resulting in a stable but offset condition.

Integral drive is used to drive away steady-state error conditions that persist over a period of time. Integral control is also a good choice for a very slow approach to a setpoint when long system settling times are needed and overshoot is undesirable.

Derivative control acts by taking action based on a change of error, often from one reading to the next. It evaluates the slope of the changing output and acts in opposition to the change. Derivative control can prevent hunting and oscillations, but too much drive can send a system into wild oscillations.

Each control mode has its own unique characteristic response to maintaining the desired output. Volumes have been written on the subject of PID control and tuning. Not all systems require all three evaluations and drive. The floating ball, discussed earlier, operates fine without integral, but derivative is a must because of the fast disturbance reaction needed. A car's cruise control doesn't need derivative, because in general fast disturbances are unlikely and the control action would be probably be uncomfortable to feel.

We have just scratched the surface of process control theory through feedback. Our focus has been limited to control action based on feeding back information from the output of our process. When disturbances affect our process, changes in the output are detected and generate an error signal. PID is tuned to drive the error away as quickly as possible. Tight control of the process variable is possible with PID, but the fundamental premise of feedback control is to respond to error. Error is expected and, to a certain degree, tolerated.

As we leave this chapter, consider an alternative to feedback control. That is feed-forward control. In feed-forward control you measure those factors that disturb a process. Understanding how they affect the variable you are holding constant will allow for output action to be taken before an error signal results. If you could measure changes in ambient temperature and wind speed from the fan, could you use this information to better control your incubator?

SOLUTIONS TO CHAPTER 8 CHALLENGES

Challenge 8-1 Solution

Part A:

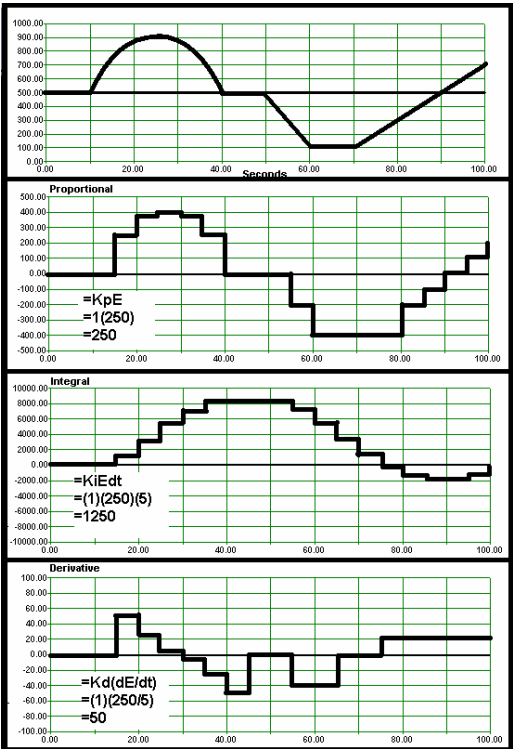
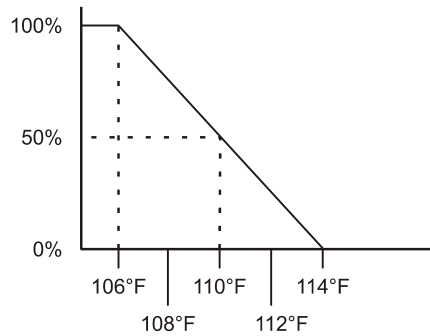


Figure 8-19
Challenge 8-1 Part A
Solution

Part B:

Below is data representing the systems conditions and data for PID evaluations. Calculate the output for each evaluation. $K_p = 2$ $K_i = 0.01$ $K_d = 4$ Setpoint = 500

Table 8-2: PID Evaluation Activity Solutions Given $K_p = 2$ $K_i = 0.01$ $K_d = 4$ Setpoint = 500							
Actual	Δt	E	ΔE	$P(K_p E)$	$E \Delta t$	$I(K_i \Sigma E \Delta t)$	$D(K_d \Delta E / \Delta t)$
500	1.0	0	0	0	0	0	0
510	1.0	10	10	20	10	.1	40
490	0.5	-10	-20	-20	-5	0.05	-160
480	0.5	-20	-10	-40	-10	-0.05	-80
470	2.0	-30	-20	-60	-60	-0.65	-40
520	1.0	20	50	40	20	-0.45	200
500	0.5	0	-20	0	0	-0.45	-160
500	1.0	0	0	0	0	-0.45	0

Challenge 8-3 Solution**Figure 8-20**

1. 200% proportional band curve for a system with a setpoint of 110 °F and a band of ± 2 °F

2. Proportional Gain = $1/200\% = 1/2 = 0.5$
3. 111 °F = 37.5%
106 °F = 100%
115 °F = 0%

Challenge 8-4 Solution

1. A positive error would remain and flow would stabilize to less than the setpoint.
2. Increasing gain would result in a smaller error, but would result in more oscillations.
3. Decreasing gain would result in a larger error with fewer oscillations.

Challenge 8-5 Solution

Yes, a system can be controlled using integral alone, though response will be very slow with low settings of K_I , and larger values can lead to excessive oscillations. It would be appropriate in systems where disturbances are relatively small but long lasting.

Challenge 8-6 Solution

No, a system cannot be controlled on derivative drive alone over the long haul. Temperature may stay fairly stable for a couple minutes as derivative bucks the change, but the temperature will drift lower, eventually to room temperature. Derivative responds to the rate at which error changes only, and is not based on the actual error.

8**Final Challenge Solution**

Solutions will vary, as they will be unique for each system.