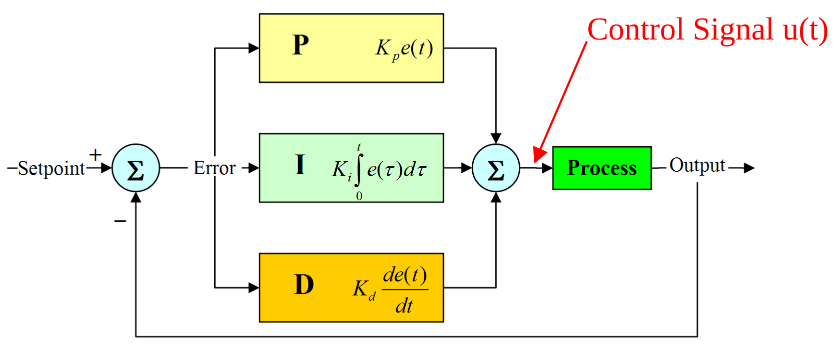
- **Preparation for Final Lab Project**
- **Simple Motor Control**

Motor Control

- A proportional–integral–derivative controller (**PID controller**) is a generic control loop feedback mechanism (controller) widely used in industrial control systems
  - A PID is the most commonly used feedback controller.
  - A PID controller calculates an "error" value as the difference between a measured process variable and a desired set point. The controller attempts to minimize the error by adjusting the process control inputs.

- The PID controller calculation involves three separate constant parameters, and is accordingly sometimes called **three-term control**:
  1) The proportional (P)
  2) The integral (I)
  3) The derivative values (D)

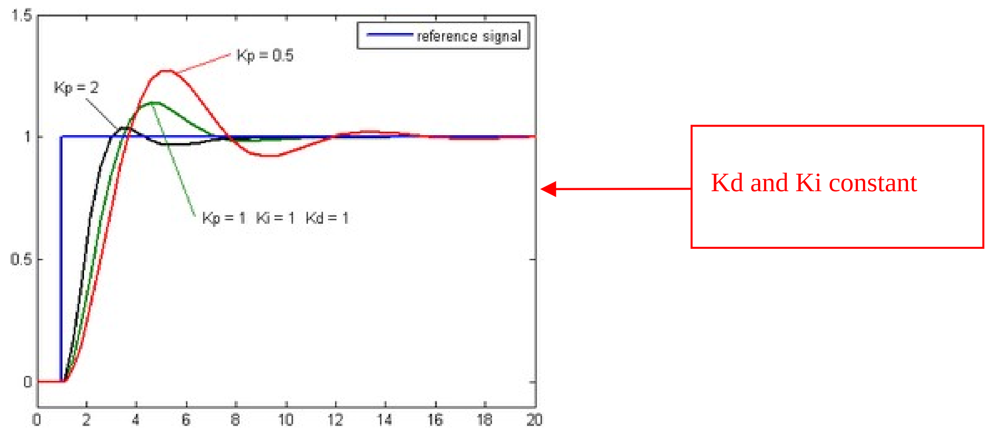
- These values can be interpreted in terms of time: *P* depends on the *present* error, *I* on the accumulation of *past* errors, and *D* is a prediction of *future* errors, based on current rate of change. The weighted sum of these three actions is used to adjust the process via a control element such as the position of a control valve, the power supply of a heating element, or the supply voltage to a motor (to control acceleration, velocity, or position).

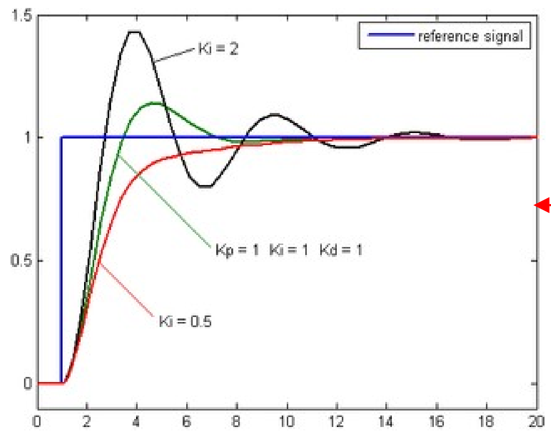


$$\mathrm{u(t)} = \mathrm{MV(t)} = \boxed{K_p e(t)} + \boxed{K_i \int_0^t e(\tau)\,d\tau} + \boxed{K_d \frac{d}{dt} e(t)}$$

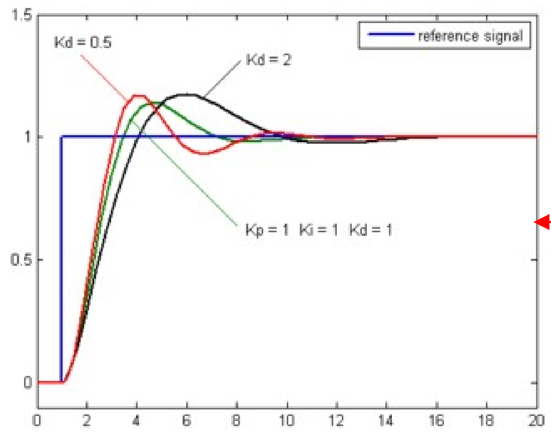
where

- K_p: Proportional gain, a tuning parameter
- K_i: Integral gain, a tuning parameter
- K_d: Derivative gain, a tuning parameter
- e: Error = $SP - PV$
- t: Time or instantaneous time (the present)





Kp and Kd constant



Kp and Ki constant

Stability

- If the PID controller parameters (the gains of the proportional, integral and derivative terms) are chosen incorrectly, the controlled process input can be unstable, i.e. its output diverges, with or without oscillation, and is limited only by saturation or mechanical breakage. Instability is caused by *excess* gain, particularly in the presence of significant lag.

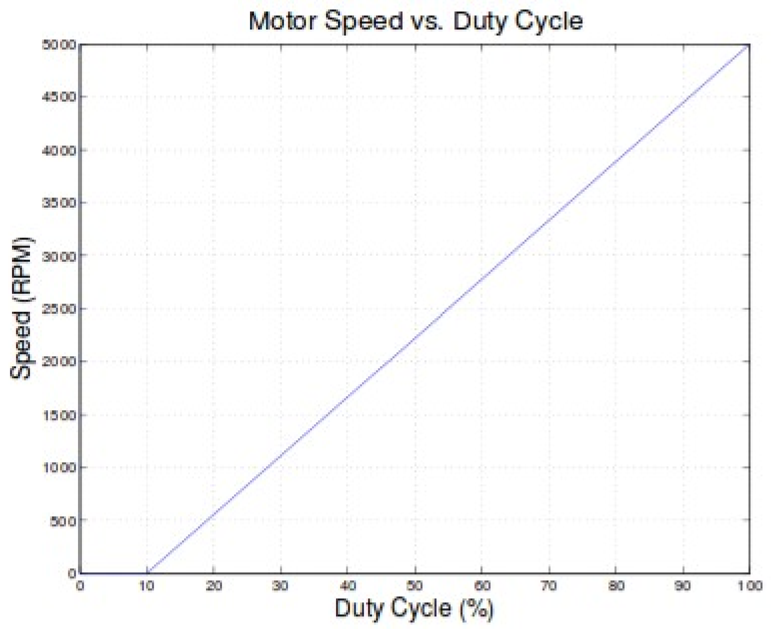
- Generally, stability of response is required and the process must not oscillate for any combination of process conditions and setpoints, though sometimes marginal stability (bounded oscillation) is acceptable or desired.

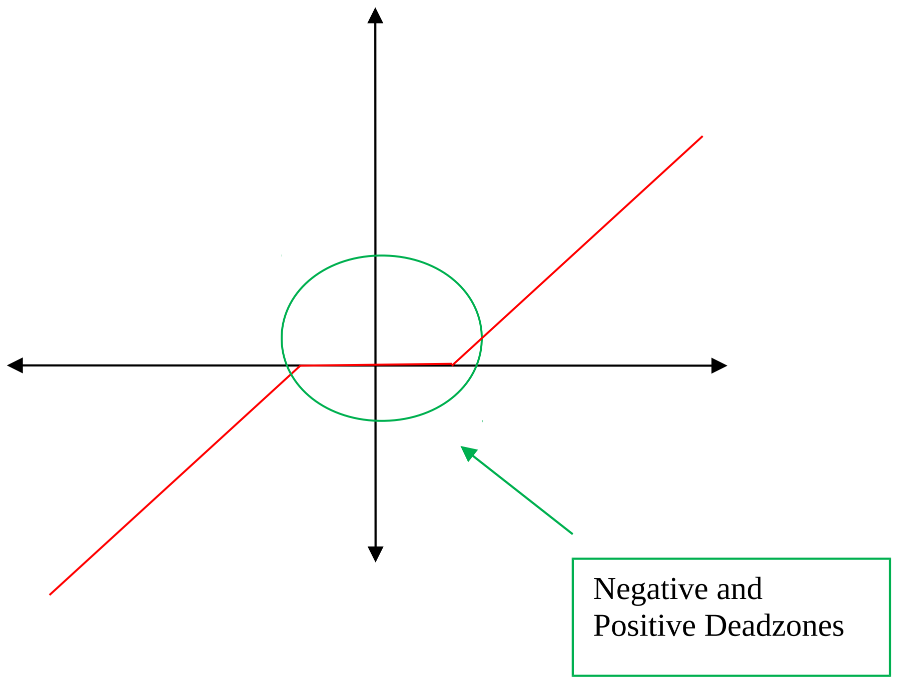
Optimum behavior

- The optimum behavior on a process change or setpoint change varies depending on the application.
- Two basic requirements are *regulation* (disturbance rejection – staying at a given setpoint) and *command tracking* (implementing setpoint changes) – these refer to how well the controlled variable tracks the desired value.
- Specific criteria for command tracking include rise time and settling time. Some processes must not allow an overshoot of the process variable beyond the setpoint if, for example, this would be unsafe. Other processes must minimize the energy expended in reaching a new setpoint.

  - Rise time (**tr**): time for the output signal to go from **10% - 90 %** of the step height (or any input signal).
  - Settling time (**ts**): time it takes for the signal to stabilize (be within **5%**, **1%**, or other).
  - Percent overshoot (**PO**): is the maximum value minus the step value divided by the step value. In the case of the unit step, the overshoot is just the maximum value of the step response minus one.

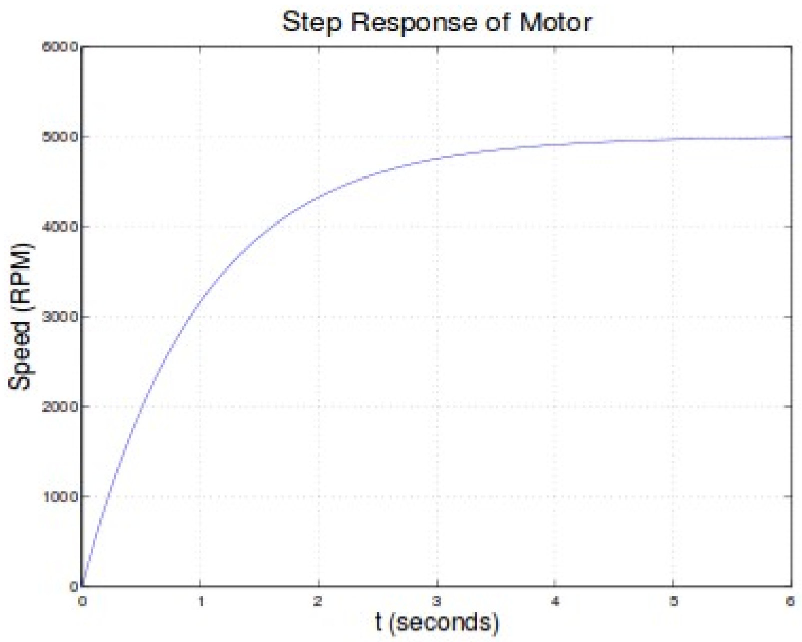
• Consider a motor which has a maximum speed of 5000 RPM. The speed vs. duty cycle may look something like this:





• The motor doesn't start rotating until it is driven with a 10% duty cycle (**??**), after which it will increase speed linearly with the increase in duty cycle.

• If the motor is initially stopped, and is then turned on (with 100% duty cycle), the speed vs. time might look something like this (the step response of the motor):



• <u>We will control the motor by adjusting the duty cycle with the MC9S12</u>. We will do this by measuring the speed and updating the duty cycle on a regular basis. <u>Let's do the adjustments once every 8 ms</u>. This means that we will adjust the duty cycle, wait for 8 ms to find the new speed, then adjust the duty cycle again. How much change in speed will there be in 8 ms? The motor behaves like a single time constant system, so the equation for the speed as a function of time is:

$$\mathbf{S(t) = Sf + e^{-t/\tau}\,(Si - Sf\,)}$$

where Si is the speed at time 0, Sf is the speed at time 1, and τ is the time constant of the system. With a duty cycle of D, the final speed will be:

$$\mathbf{Sf = \alpha DC + S0}$$

where S0 is the speed the motor would turn with a 0% duty cycle if the speed continued linearly for duty cycles less than 10%, and α is the slope of the speed vs. duty cycle line (5000/0.9 in this example).

• Here we assume that the time constant of the small motors we are using is about 1 second — i.e., it takes about 5 seconds (5 time constants) for the motor to go from a dead stop to full speed. If T = 8 ms, the motor will have changed its speed from Si to

$$\mathbf{S(T) = Sf + e^{-T/\tau}\,(Si - Sf\,)}$$
$$\mathbf{S(T) = (\alpha DC + S0) + e^{-T/\tau}\,(Si - (\alpha DC + S0)\,)}$$
$$\mathbf{S(T) = (\alpha DC + S0)(1 - e^{-T/\tau}) + e^{-T/\tau}\,Si}$$

The speed at the nth cycle, S[n], will be

$$\mathbf{S[n] = (\alpha DC + S0)(1 - e^{-T/\tau}) + e^{-T/\tau}\,S[n-1]}$$

• Consider an integral controller where the duty cycle is adjusted according to:

$$\mathbf{DC[n] = DC[n-1] + k(Sd - Sm[n])}$$

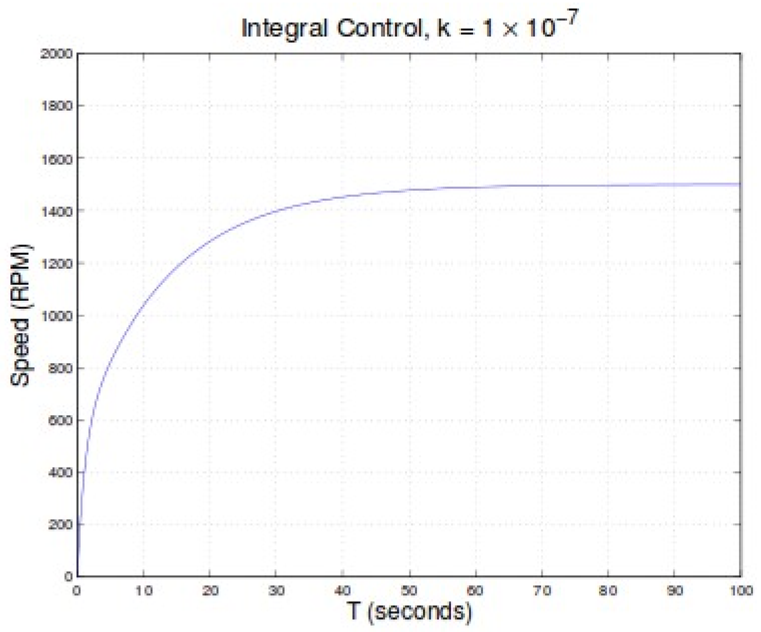
We can simulate the motor response by iterating through these equations. Start with Sm[1] = 0, D[1] = 0 (Duty Cycle), and Sd = 1500. Then we calculate:

$$\mathbf{Sm[n] = (\alpha DC[n-1] + S0)(1 - e^{-T/\tau}) + e^{-T/\tau}\, Sm[n-1]}$$
$$\mathbf{DC[n] = DC[n-1] + k(Sd - Sm[n])}$$

In MATLAB we can simulate this as:

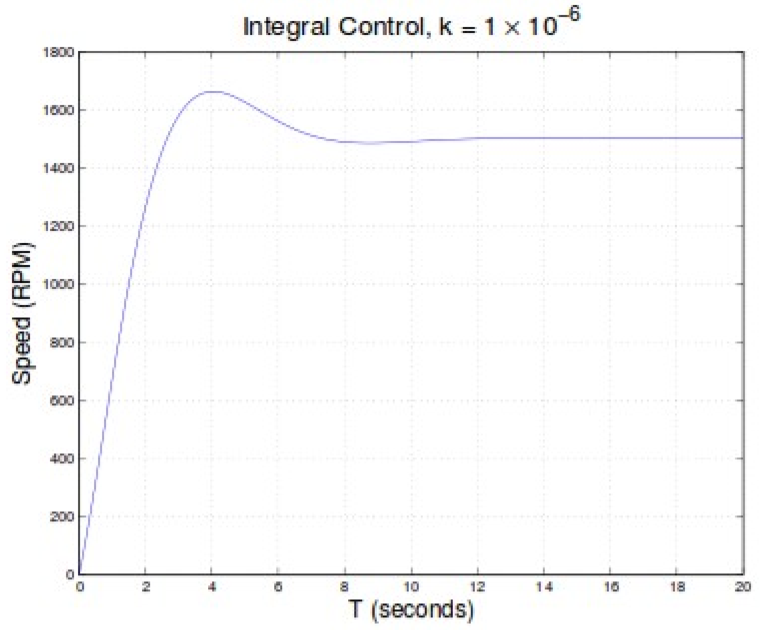
```
alpha = 5000/0.9;   % Max speed 5,000 RPM; turns on at 10% duty cycle
Sd = 1500;          % Desired Speed
S0 = -alpha*0.1;    % Speed motor would turn at 0% duty cycle if linear
tau = 1;            % One second time constant
T = 8e-3;            % Update rate is 8 ms
k = 1e-7;           % Constant for integral control
Sm = 0;             % Measured speed starts at 0
D = 0.1;             % Duty cycle starts at 10%
t = 0;
ee = exp(-T/tau);   % Precalculate this commonly used value
for n=2:10000                % Make end value bigger if needed
        Sm(n)=(alpha*D(n-1) + S0)*(1-ee) + ee*Sm(n-1);
        D(n) = k*(Sd - Sm(n)) + D(n-1);
        t(n) = t(n-1)+T;
end
plot(t,Sm);
```

• By changing the value of k we can see how this parameter affects the response. Here is the curve for $k = 1.0 \times 10^{-7}$:



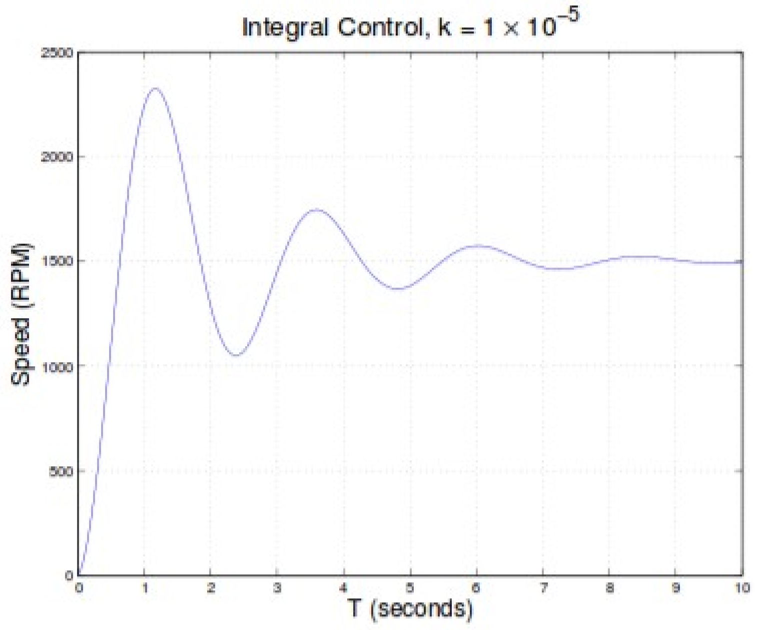
• With this value of k, it will take about 1 minute for the motor to get to the desired speed.

• Here is the curve for $k = 1.0 \times 10^{-6}$:

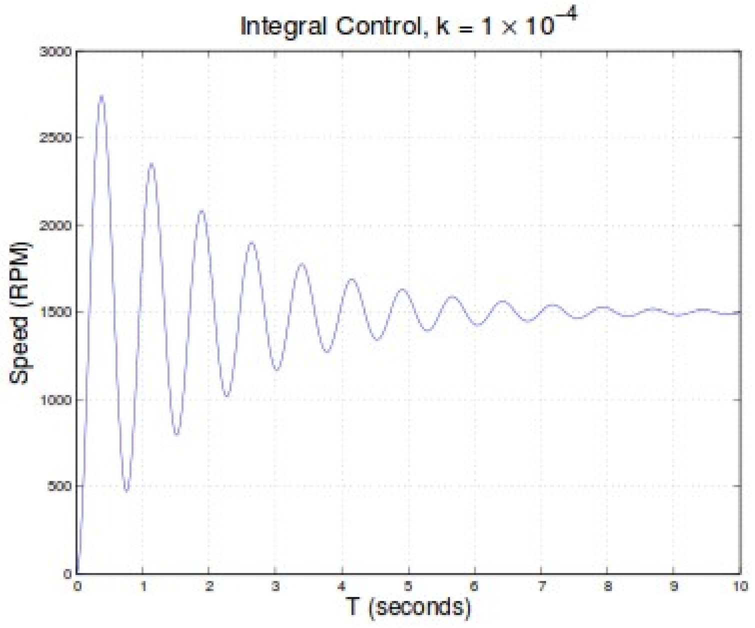


• Now it takes about 10 seconds to get to the desired speed, with a little bit of overshoot.

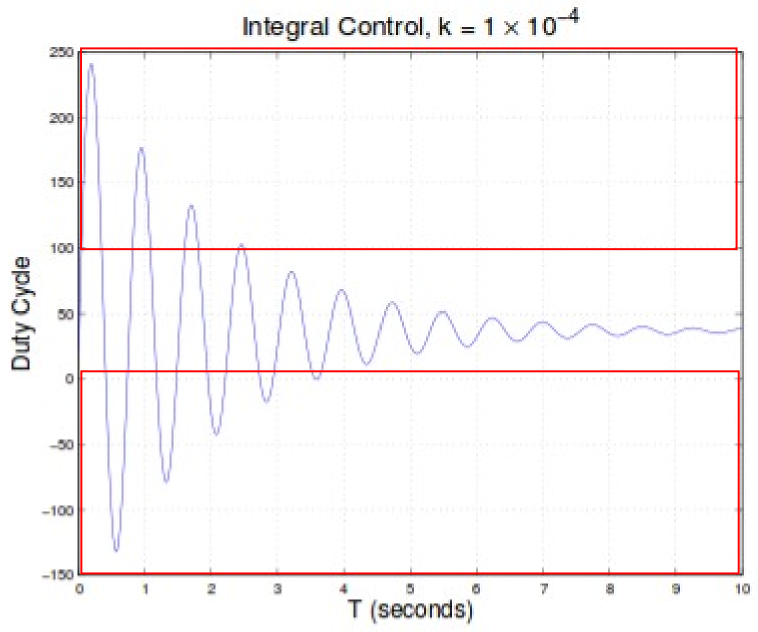
Let's try $k = 1.0 \times 10^{-5}$:



• This gets to the desired value more quickly, but with a lot of oscillation. Let's increase k to 10^{-4}.



• For this value of k there is a significant oscillation. However, a real motor will not act like this. If we look at the duty cycle vs time, we see:

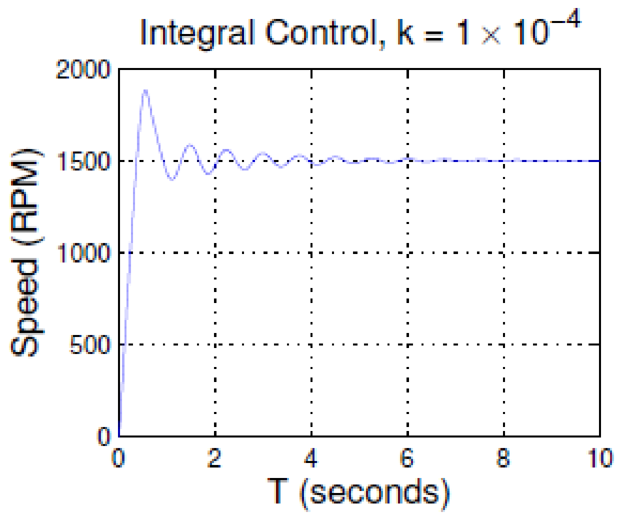


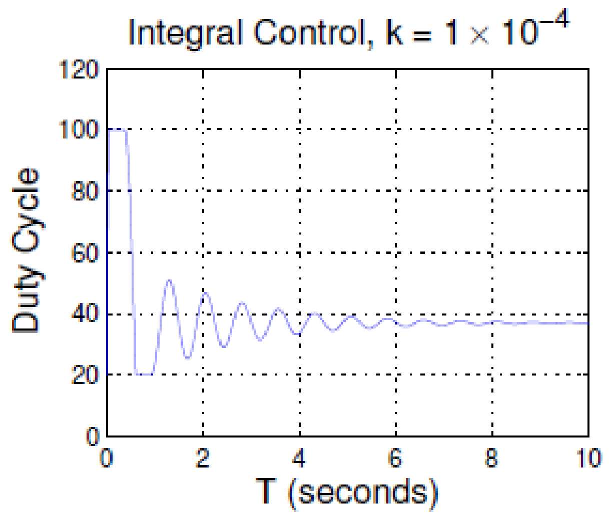
• <u>To get this oscillating response, the duty cycle must go over 100%, and below 0%, which is clearly impossible</u>. To get the response we expect in the lab, we need to limit the duty cycle to remain between 20% and 100%. Thus, we change our simulation to be:

```
alpha = 5000/0.9;   % Max speed 5,000 RPM; turns on at 10% duty cycle
Sd = 1500;          % Desired Speed
S0 = -alpha*0.1;     % Speed motor would turn at 0% duty cycle if linear
tau = 1;            % One second time constant
T = 8e-3;            % Update rate is 8 ms
k = 1e-7;           % Constant for integral control
Sm = 0;              % Measured speed starts at 0
D = 0.1;             % Duty cycle starts at 10%
```

```
t = 0;
ee = exp(-T/tau);              % Precalculate this commonly used value
for n=2:1000                   % Make end value bigger if needed
  Sm(n)=(alpha*D(n-1) + S0)*(1-ee) + ee*Sm(n-1);
  if (Sm(n) < 0) Sm(n) = 0; end; % Motor speed cannot be less than 0
  D(n) = k*(Sd - Sm(n)) + D(n-1);
  if (D(n) > 1.0) D(n) = 1.0; end;  % Keep DC between 20% and 100%
  if (D(n) < 0.2) D(n) = 0.2; end;
  t(n) = t(n-1)+T;
end
plot(t,Sm);
```

• When we use this to simulate the motor response, we get:





•In your program for next Lab, you will use a Real Time Interrupt with an 8 ms period. In the RTI interrupt service routine, you will measure the speed, and set the duty cycle based on the measured speed. Your ISR will look something like this:

```
void INTERRUPT rti_isr(void)
{
        Code to read potentiometer voltage and convert into RPM

        Code to measure speed Sm in RPM

        Code which sets duty cycle to:

                DC = DC + k*(Sd-Sm)
                if (DC > 1.0) DC = 1.0;
                if (DC < 0.2) DC = 0.2;

        Code which writes the PWM Duty Cycle Register to generate
        duty cycle DC.

        Code which clears RTI flag
}
```

• In the main program, you will display the measured speed, desired speed, and duty cycle on the LCD display.

• <u>Your values of k will probably be different than the values in these notes because speed vs. duty cycle, time constant, and maximum speed will most likely be different than the values I used</u>.