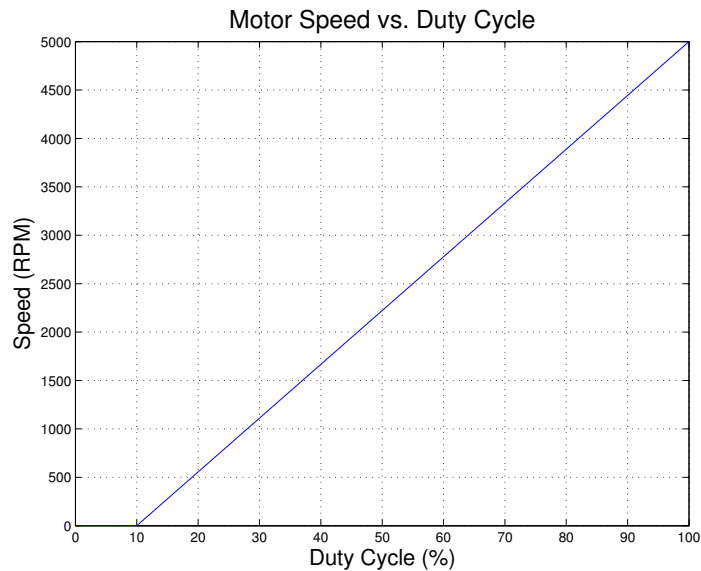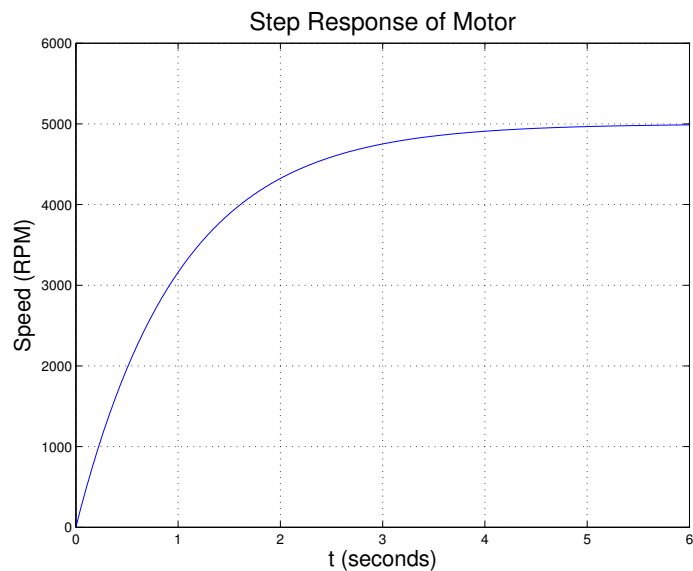# Motor Control

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:

1

We will control the motor by adjusting the duty cycle with the HCS12. We will do this by measuring the speed and updating the duty cycle on a regular basis. Let's do the adjustments once every 8 ms. 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:

$$S(t) = S_f + e^{-t/\tau}(S_i - S_f)$$

where S_i is the speed at time 0, S_f is the speed at time ∞, and τ is the time constant of the system. With a duty cycle of D, the final speed will be:

$$S_f = \alpha D + S_0$$

where S_0 is the speed the motor would turn with a 0% duty cycle if the speed continued linearly for duty cyclces less than 10%, and α is the slope of the speed vs. duty cycle line (5000/0.9 in this example).

Here I 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 S_i to

$$S(T) = S_f + e^{-T/\tau}(S_i - S_f)$$
$$S(T) = (\alpha D + S_0)(1 - e^{-T/\tau}) + e^{-t/\tau}S_i$$
$$S[n] = (\alpha D + S_0)(1 - e^{-T/\tau}) + e^{-t/\tau}S[n-1]$$

where $S[n]$ is the speed at the nth cycle.

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

$$D[n] = D[n-1] + k(S_d - S_m[n])$$

We can simulate the motor response by iterating through these equations. Start with $S_m[1] = 0$, $D[1] = 0$, and $S_d = 1500$. Then we calculate:
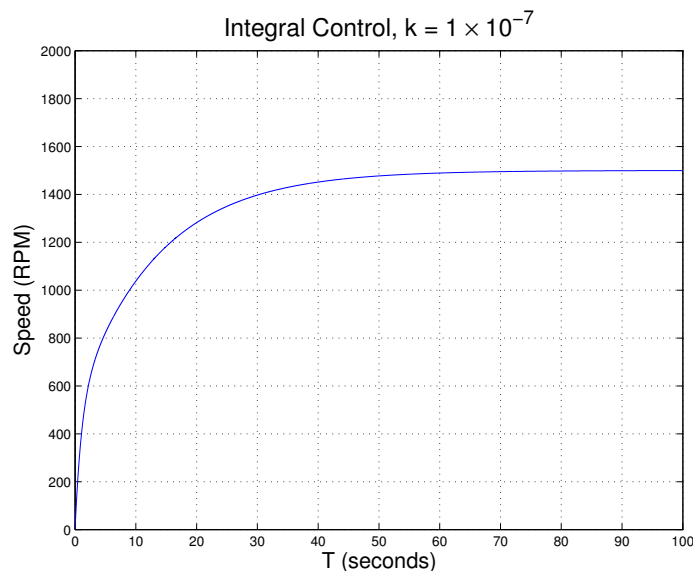
$$S_m[n] = (\alpha D - S_0)(1 - e^{-T/\tau} + e^{-t/\tau} S_m[n-1]$$

$$D[n] = D[n-1] + k(S_d - S_m[n])$$

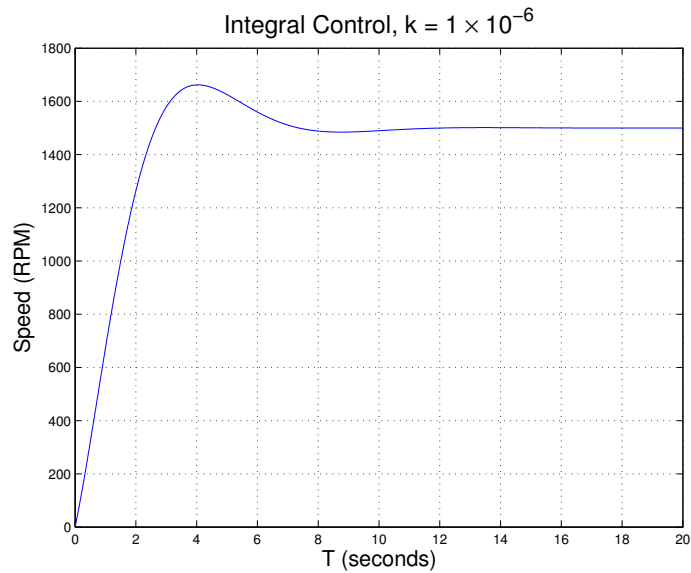In MATLAB we can simulate this as:

```
Sm = 0;
D = 0;
ee = exp(-T/tau);
for n=2:1000
    Sm(n)=(alpha*D(n-1) + S0)*(1-ee) + ee*Sm(n-1);
    D(n) = k*(Sd - Sm(n)) + D(n-1);
end
```

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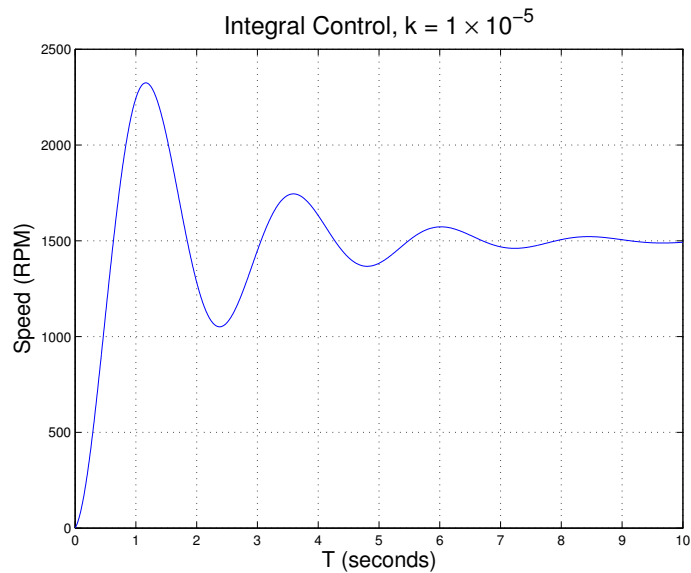

With this value of k, it will take about 1 minute for the motor to get to the desired speed.

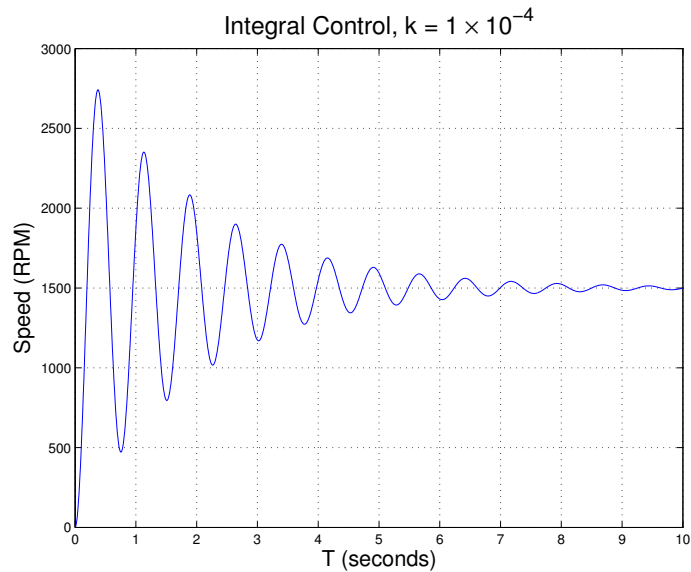Here is the curve for $k = 1.0 \times 10^{-6}$:

Now it takes about 10 seconds to get to the desired speed, with a little bit of overshoot.
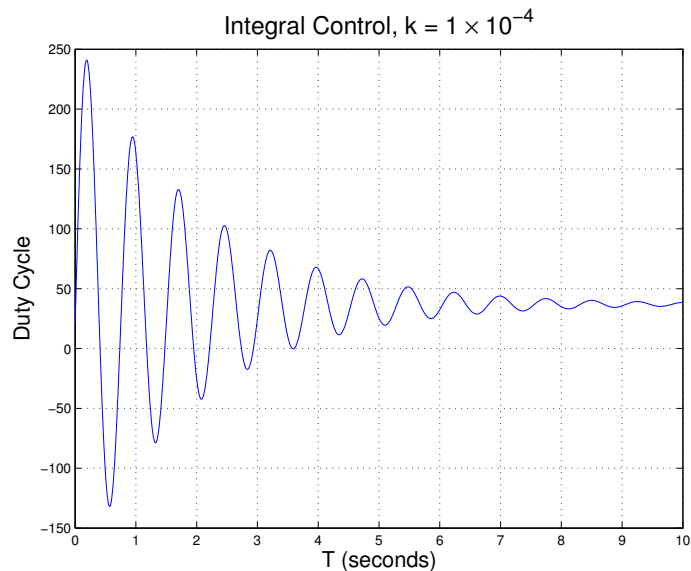
Let's try $k = 1.0 \times 10^{-5}$:

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

Integral Control, $k = 1 \times 10^{-4}$

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:

Integral Control, $k = 1 \times 10^{-4}$

To get this oscillating response, the duty cycle must go to over 100%, and below 0%, which is clearly impossible. 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:

```
Sm = 0;
D = 0;
ee = exp(-T/tau);
for n=2:1000
    Sm(n)=(alpha*D(n-1) + S0)*(1-ee) + ee*Sm(n-1);
    D(n) = k*(Sd - Sm(n)) + D(n-1);
    if (D(n) > 1)
        D(n) = 1;
    end;
    if (D(n) < 0.2)
        D(n) = 0.2;
    end;
end
```

When we use this to simulate the motor response, we get:
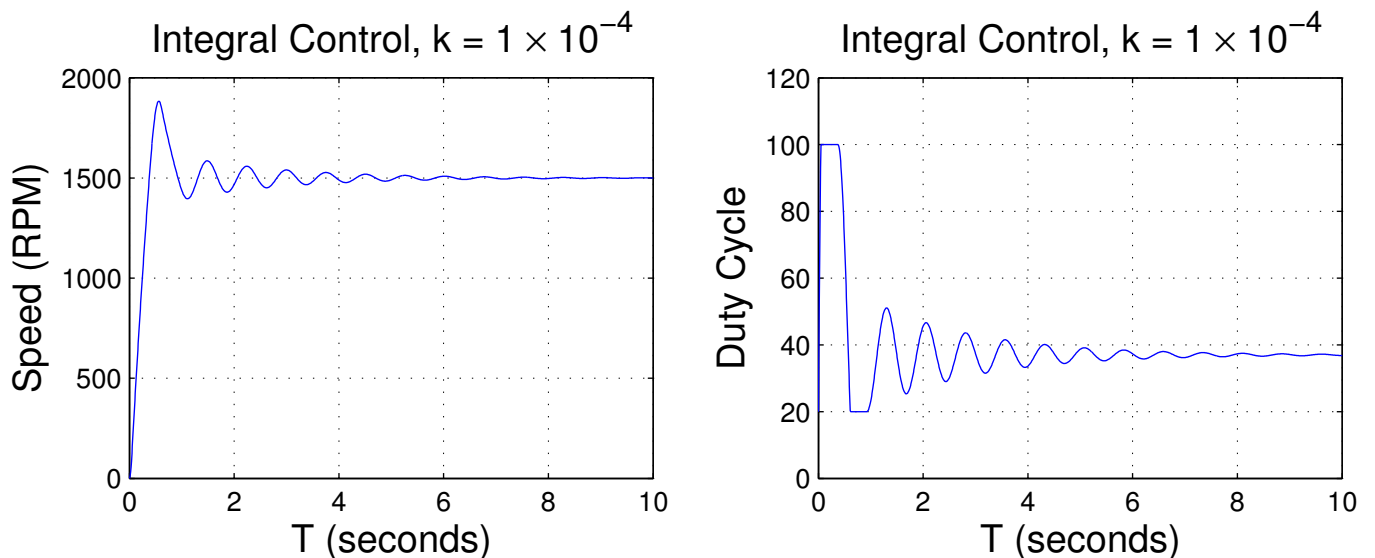
In your program for Lab 5, you will use a Real Time Interrupt with an 8 ms period. In the RTI interrupt service routine, you will measure the speed, and set the duty cycle based on the measured speed. Your ISR will look something like this:

```
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 print the measured speed, desired speed, and duty cycle to the screen.

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.

Using Floating Point Numbers with the Gnu C Compiler

It will be much easier to do the necessary calculations by using floating point numbers. Here is an example of a program which uses floating point:

```
#include "DBug12.h"

main()
{
    float x;
    x = 10.2;
    printf("x = %d\r\n",(short) x);
}
```

To use floating point numbers with the Gnu C compiler, go to the **Options** menu, **Project options** submenu, and add `-fshort-double` to the list of compiler opitons:
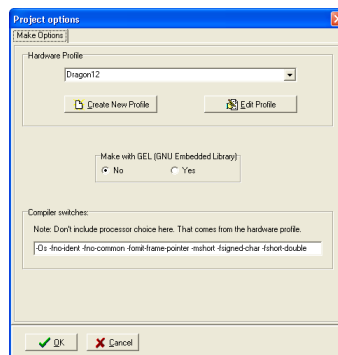
You cannot use math functions such as `sqrt()`. The size of the code which will be created if you use the math library for the Gnu C compiler will be too large to fit in the memory of the 9S12. You can do standard arithmetic operations such as addition, multiplication and divison. Also, you cannot

print floating point numbers using `DB12FNP->printf()`. You must convert numbers to integer before printing them.