Overview

This file is meant to describe some coding issues that are of importance. Ken described most of this after troubleshooting for charles and Koen.

This file deals with Memory Allocation, Integer arithmetic, prop gain and integral gain, and how it is implemented in C

Additional resources

Need more help?

Check the resources, and then see Ken

Main content

**Part 1: Memory Allocation**

the C programming language uses pointers to allocate memory.  This is more confusing than  MATLAB.  For example in matlab, you can do these without error.

x = [1 1 2 3 5];

a = x(1)

    a=1  \* the first index in matlab is i=1

a = x(3)

    a=2

x(6)=8

    x=[1 1 2 3 5 8]  \* this dynamically resized the array

However, you might end up with a memory/processing deficit in the end.

In C, you define memory

For example

short int \* x=3  //defines a pointer to x value x, \* defines a pointer

x(0)

    3

x(1)

    16384

x(0) called the pointer that was defined and had the value 3 placed in it.

x(1) called a memory location a short int away from the original memory and red out what the value of that would be.

So to make an array you should allocate the memory.  For example

buffer\_variable\_name = (short int \*) malloc(number\_of\_points \*size of (short int))

Also make sure you set the allocated memory to zero.

for (i=0;i<number\_of\_points;i++)

{

    buffer\_variable\_name[i]=(short int)0;

}

At the end of your c code make sure to free memory

free(buffer\_variable\_name)

One issue with the current code is that for large arrays, there is a lot of calculations used.

for (i=0;i<number\_of\_points-1;i++)

{

    buffer\_variable\_name[i]=buffer\_variable\_name[i+1];

}

buffer\_variable\_name[i-1]=current\_value

which means that this buffer: [0 0 0 1] is changed to [0 0 1 2] if the next current value is 2.

If the array size is too big, then one might be able to move pointers.  for example i you have the buffer [0 2 4 6 8 ... N(i)], then on the third iteration (or i+3rd iteration), if the next value is 1, you might ger [0 2 1 6 8 ... N(i)].

**Part 2: Proportional Gain**

The current formulation says for proportional gain is

short int fl\_polarity  // 1 or -1

short int prop\_gain

short int error\_value  //the difference between the current position and the target position

short int correction

correction = (short int)

((long int)fl\_polarity \*

(((long int)prop\_gain \* (long int)error\_value))/10000);

if the error\_value is greater than 5V, you might get some of the looping problems as described above.  (however this has worked....)

Note that there is a lot of casting to different types.  This is important because with out it there is significant round off error and the motor response will "step" instead of having a "smooth" response.

The sensitivity of the system is defined by the prop gain.  If the minimum correction is 1, then the minimum error\_value is 10000/prop\_gain.  So the larger the prop gain, the less sensitive to noise it becomes.

Now, the inherint noise of the system is ~ 2mV (peak to peak) which means a correction of 13 is noise, if the error\_value is less than ~13000/propgain, then it'll basically be responding to noise.

**Part 3: Integral Gain**

short int fl\_polarity    // 1 or -1

short int ki             // integral gain parameter

short int ki2

short int integral\_points

short int gain\_asymmetry

long  int error\_holder   //sum of previous 10 error\_values

if (out\_diff\_accumulator<0)

{

Ki2 = Ki + gain\_asymmetry;

}

else

{

Ki2 = Ki;

}

correction = (short int)

((long int)fl\_polarity \*

((long int)Ki2 \* (error\_holder / (long int)integral\_points)))/10000);

Note that error values was a short int, so dividing by integral\_points guarantees that it is a short int and it wont be a value that loops.

(The if statement allows for the integral gain to increase on the relenthening.  This appears to help get "flatter" better than adjusting prop gain.)

This implementation of integral gain also has a bit of stepping (or oscillations) based on the integral points.  large integral points leads to larger amplitude and slower frequency oscillations during the hold.

The sensitivity of the integral gain is defined by the divisor and ki.  for a minimum correction of 1, then the minimum error\_holder/integration\_points value is 10000/ki.

**Part 4: Getting a True Isotonic Hold**

...is basically impossible. There are probably a few reasons. 1) biological: the muscle stiffness changes as the muscle is relengthened. This means that the system is nonlinear. 2) "droop": (see wikipedia "pid control" and "voltage droop". Previously we counteracted this using isoinc. This is likely an important factor.

Other possible future considerations is to make a time-varying or nonlinear PI control. Note that non-linear PI control (e.g. error\_value^2) is quite difficult, especially with integer math.