RILEY POOLE
ME 477 – LAB REPORT 07
WINTER 2019

I. DESCRIPTION

The overall goal of this experiment was to create a feedback system that maintains the speed of a DC motor. The control algorithm uses a discrete serial biquad filter. Its input is the value read from an optical quadrature encoder attached to the motor's shaft. The program works with three threads. The main program is in one thread and sets up the entry table. This table editor functions in a second thread. A third thread is used to run the calculations in the biquad cascade and set the voltage (speed) output of the motor.

THE MAJOR TASKS PERFORMED BY THIS PROGRAM WERE:

- Control the speed of the motor with a control algorithm
- Allow the user freedom to choose the reference velocity and filter parameters while the motor is running
- Output the results of changes to reference velocity as a *.mat file for further processing

THE LIMITATIONS OF THIS PROGRAMS CAPABILITY ARE:

- Very low speeds are not attainable as the voltage is too low to overcome friction
- The Ki and Kp inputs are not intuitive to use by the user as compared to something like entering a cutoff frequency
- Step responses of only 400 RPM can cause the motor to nearly saturate and causes significant deviation from theoretical results in the first few milliseconds of response
- Changes in speed greater than 400 RPM can cause excessive current draws from the amplifier and cause it to turn off

EXPLAIN ANY ALGORITHMS

The control algorithm uses a serial biquad cascade. The input to the cascade is the difference in speed (rad/sec) between the set velocity and the reference velocity. The output of each biquad becomes the input for the next. These biquads are generated by sectioning the discrete transfer function into second order sections. The program runs in two threads as the ctable() function must run for the duration of the program. Matlab data is saved at the end of each timer interrupt and the file is saved when interrupt is signaled to stop by main (i.e. ctable() has ended).

HIERARCHICAL STRUCTURE

- main
  - myRio_Open
  - MyRio_IsNotsuccess
  - setup_table
  - Irq_RegisterTimerIrq
  - pthread_create
  - EncoderC_initialize
  - ctable2
  - pthread_join
  - Irq_UnregisterTimerIrq
  - MyRio_Close
- Timer_Irq_Thread
  - Irq_Wait
  - Nifpga_WriteU32
  - NiFpga_WriteBool
  - vel
  - update_filter
  - rpm2rad
  - cascade
  - Aio_Write
  - Irq_Acknowledge
  - openmatfile
  - matfile_addstring
  - mattile_addmatrix
  - matfile_close
- cascade
  - SATURATE (macro)
- update_filter
  - none (procedural)
- vel
  - Encoder_Counter
  - bdi_bti2rpm
- setup_table
  - none (procedural)
- rpm2rad
  - none (procedural)
- bdi_bti2rpm
  - none (procedural)

II. TESTING

1. Test that the program can maintain a range of set speeds
    a. Run the program
    b. Set the current speed to 200 RPM
        i. <1> <ENTR> <2><0><0> <ENTR>
        ii. Check that V_act on the LCD reads approximately 200 RPM
        iii. Measure the rotation speed with the optical tachometer to check that it reads approximately 200 RPM
    c. Continue step 1.b again for 600 and 1000 RPM. Make sure to do these in order without resetting the program. The amplifier cannot handle large changes in speed because of the heavy current draws. Keep speed changes below 400 RPM in magnitude.
2. Repeat step 1 with negative values
    a. End the program with <backspace> and run it again in Eclipse.
    b. let the motor halt
    c. When entering values as in 1.b prefix values with a <-> sign
3. Test the program can change the filter values correctly.
    a. Run the program
    b. Set the current speed to 200 rpm
        i. <1> <ENTR> <2><0><0> <ENTR>
    c. Apply a load and let go the motor should be overshoot approximately twice
    d. Set the Ki value to 10
        i. <5> <ENTR> <1><0> <ENTR>
        ii. Apply a load and let go the motor should overshoot more than 7 times
    e. return Ki back to its original value
        i. <5> <ENTR> <2> <ENTR>
    f. Set the Kp value to 0.5
        i. <4> <ENTR> <0><.><0><5> <ENTR>
        ii. Apply a load and let go the motor should overshoot about 4 times
    g. return Kp back to its original value
        i. <4> <ENTR> <0><.><1> <ENTR>
    h. Check that the program remains stable with different values of BTI
        i. Set the BTI to four milliseconds
            1. <6> <ENTR> <4> <ENTR>
            2. Set the current speed to 200 RPM
            3. Apply a light load and check that the response overshoots about twice
4. Test the program can output a matfile correctly
    a. Run the program
    b. Change the speed setting as in step 1. A matlab file will only be generated if there has been a change in set velocity

c. <backspace> on keypad. Matlab file will not be generated until after matfile_close() and not updated via SSH until after MyRio_close().

III. RESULTS

HOW SUCCESSFULLY THE PROGRAM RUNS AND UNSOLVED PROBLEMS

This program is successful as it can maintain a set speed and direction. The motor is also able to adapt to changes in loading. Increasing the friction in the system and causing the speed to drop will result in the filter outputting a higher voltage. One unsolved problem is that the filter does not adapt to large changes in speed and allows saturation to occur.

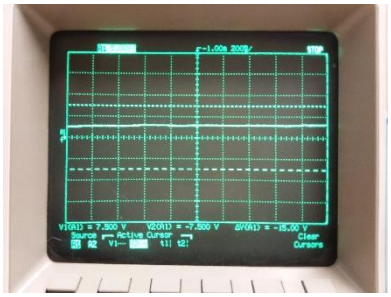
SPECIFIC QUESTIONS IN THE ASSIGNMENT

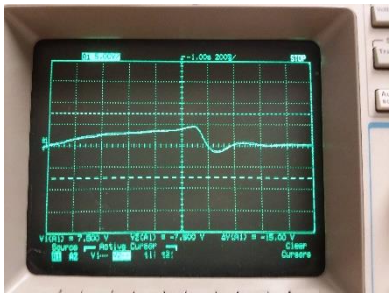
2. Measure the steady-state speed of the motor with the tachometer. Compare this value to the reference speed that you specified in the table. How close is it.

I set the motor reference speed to 150 RPM.
With the tachometer I measured 148.9 RPM. This is an error of +0.7%.

3. While the motor is at steady-state speed, gently apply a steady load torque to the motor shaft. What are the responses of the actual speed and control voltage? Explain.

When applying a steady load torque to the motor shaft the actual speed remains about the same as the reference after it settles. The control voltage increases to account for the loading and increase the torque as see in the first figure below. Letting go acts as a sort of step and we see the inverted step response to a positive step as seen in the second figure below.





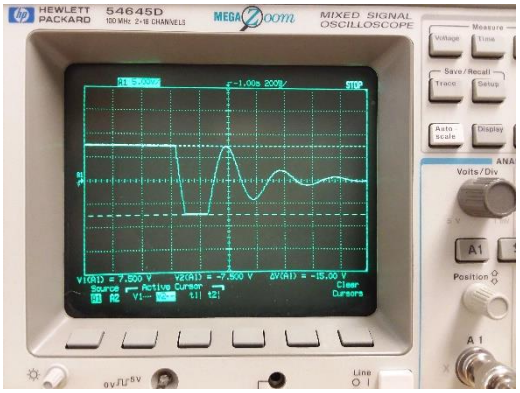
4. Beginning with the base set of parameters, explore the effect of varying the Proportional Gain Kp on the transient response. Try small (0.05) and large (0.2) value of Kp. What are the effects on the oscillation frequency and on the damping? Explain in terms of the transfer function parameters.

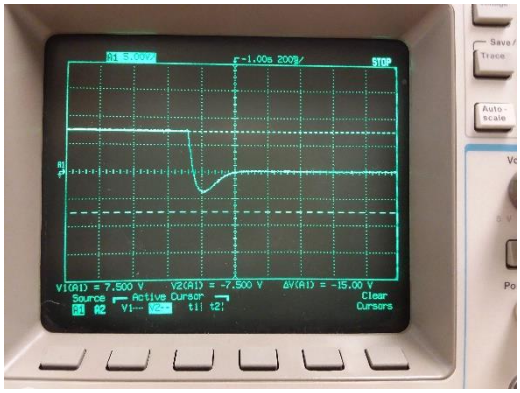
With a kp of 0.05 (figure one below) we see a longer settling time as compared to kp of 0.2 (figure two below). It appears that overshoot is higher for a low value of kp. I also notice that the time between peaks is about the same maybe a little bit less for high values of kp. **These observations suggest that the damping ratio has increased with increasing values of kp. In**

fact, it has shifted from underdamped (<1) to roughly critically damped (~1). The natural frequency has remained about the same. These results make sense when considering the analytical equations for damping and natural frequency. Where damping ratio is directly proportional to Kp and independent to natural frequency.

$$\text{the natural frequency:} \quad \omega_n = \sqrt{\frac{K_i K}{J}} \quad \text{r/s}$$

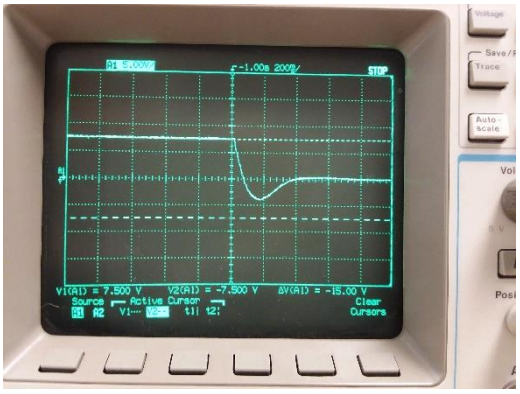
$$\text{the damping ratio:} \quad \zeta = \frac{K_p}{2}\sqrt{\frac{K}{JK_i}}$$

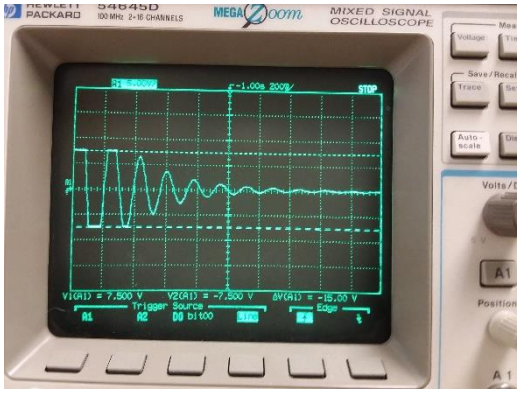




5. Beginning with the base set of parameters, explore the effect of varying the Integral Gain on the transient responses. Try small (1) and large (10) values of Ki. What are the effects on the oscillation frequency and on the damping? Explain in terms of the transfer function parameters.

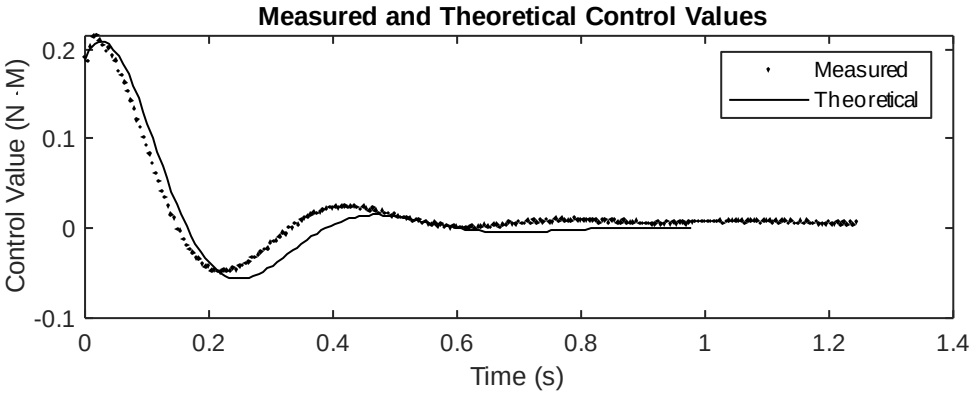
With a Ki of 1 (figure one below) we see a shorter settling time and smaller overshoot as compared to Ki of 10 (figure two below). Looking carefully, we also notice that the period looks a little shorter with a higher value of Ki. **In other words, increasing Ki has increased the natural frequency and decreased the damping ratio.** These observations are consistent with the analytical formulas as **damping ratio has an inverse relationship with Ki and natural frequency shares a proportional relationship.**

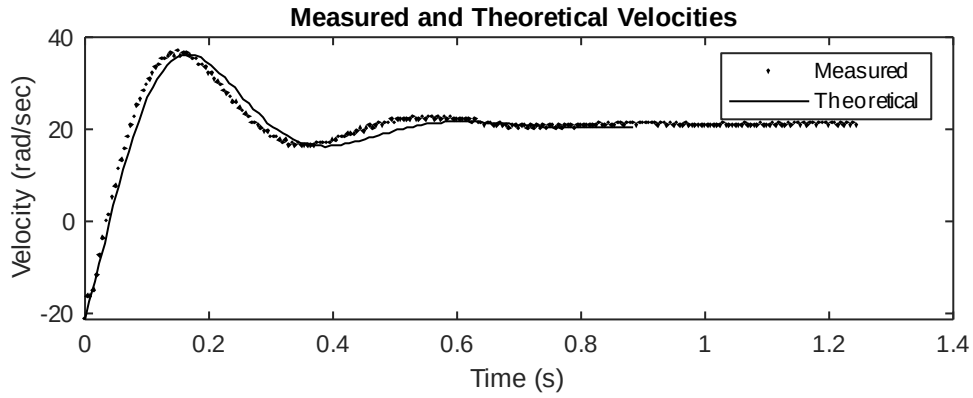




6. Finally, using the base set of parameters, record the control torque and actual velocity responses for a step change in the reference velocity that starts from -200 rpm and goes to +200 rpm. Use MATLAB to compare the experimental responses with the analytical responses for the continuous system approximation. What do you conclude?

I conclude that the measured responses match well with the theoretical ones. The theoretical control value is essentially just a scaled version of the velocity plot and so the errors are being magnified. I conclude that my discrete filter and control algorithm are working well.





POSSIBLE IMPROVEMENTS AND EXTENSIONS

- Ability for the user to set the type of filter with the keypad (high, low, etc.)
- Greater precision in velocity measurement
- More rapid updating of the table
- Visual conformation that the MATLAB data has been collected
- Ability to respond to greater step inputs without overloading the current amplifier
    - Intelligent filter

IV. FUNCTIONS OF THE PROGRAM

- **main**
  - Prototype:
    - int main(int argc, char **argv)
  - Purpose:
    - Sets up threads and interrupts; ends the timer thread
    - Initializes the encoder
    - Runs the table editor
  - Rationale for creation:
    - Need a procedural section to control the threads, interrupts, and table editor
  - Inputs and parameters:
    - No inputs or parameters used
  - Return:
    - int 0 – for normal operation
    - Procedural
      - table on the LCD screen
      - control of timer thread
- **Time_Irq_Thread**
  - Prototype:
    - void* Timer_Irq_Thread (void* resource)
  - Purpose:
    - catches the interrupt thread
    - Initializes the analog I/O
    - calculates velocity error and calls cascade to determine next voltage
    - Saves data to matlab file
    - Update the filter based on the table information
  - Rationale for creation:
    - Need a procedural section to catch the timer interrupt and calculate analog out values
  - Inputs and parameters:
    - void* resource – this is the ThreadResouce structure and is immediately recast
  - Return:
    - void* 0 – for normal operation
    - updates the filter
- **cascade**
  - Prototype:
    - double cascade(double xin, struct biquad* fa, int ns, double ymin, double ymax)
  - Purpose:
    - calculate the output level based on the filter parameters set in the biquad
    - check for a saturation condition
  - Rationale for creation:
    - need a convenient way to calculate the outputs of the biquad structure
  - Inputs and parameters:

        - None
    - Return:
        - double y0 – the output of the serially cascaded biquad structures
- **update_filter**
    - Prototype:
        - void update_filter(void)
    - Purpose:
        - update the filter based on the current table values
    - Rationale for creation:
        - need a convenient and compact way to take values from the table and calculate the filter parameters
    - Inputs and parameters:
        - None
    - Return:
        - double y0 – the output of the serially cascaded biquad structures
- **vel**
    - Prototype:
        - double vel(void)
    - Purpose:
        - calculate the current velocity in rpm
    - Rationale for creation:
        - need a convenient way to calculate the velocity of the motor
    - Inputs and parameters:
        - None
    - Return:
        - double rpm – the velocity of the motor in rpm
- **setup_table**
    - Prototype:
        - setup_table(void)
    - Purpose:
        - initialize the table used in main
        - set the convenient names for calling elsewhere in the program
        - assign the table to the ThreadResource structure
    - Rationale for creation:
        - need a convenient and compact way to set up the table for its first run
    - Inputs and parameters:
        - None
    - Return:
        - no return
        - Procedural
            - initialized table and table variables
            - table set to the correct element of ThreadResource
- **rpm2rad**
    - Prototype:
        - double rpm2rad(double rpm)
    - Purpose:

        - change the units of the input from rpm to radians
    - Rationale for creation:
        - want to avoid “magic numbers”
    - Inputs and parameters:
        - double rpm – the value in units of rpm
    - Return:
        - double rad – the value in radians/second
- **bdi_bti2rpm**
    - Prototype:
        - double bdi_bti2rpm(double bdi_bti)
    - Purpose:
        - change the units of the input from bdi/bti to rpm which is more natural
    - Rationale for creation:
        - want to avoid “magic numbers”
    - Inputs and parameters:
        - double bdi_bti
    - Return:
        - double rpm – the value of the input variable with units changed from bdi/bti to rpm

V. ALGORITHMS AND PSEUDOCODE

```
main()
      set up the table
            set the timeoutValue unit relationships
            initialize the table editor variables
            set easier names for the table variables
      register the the timer interrupt
      create the timer thread
      initialize the encoder
      run ctable()
      after ctable receives a backspace
      set the interrupt thread to false
      wait for the thread to finish
      unregister the timer
      close the MyRio

Timer_Irq_Thread
      set the timeoutvalue unit relationships
      recast the threadResource structure
      initilize the analong I/O
      set the motor voltage to 0
      while the main thread singals this one to run
            wait for the next interrupt
            if the interrupt is asserted
                  schedule the next interrupt
                  call vel() to update the velocity
                  update the filter values with update_filter()
                  calculate the error between the current velocity and the
set velocity
                  call cascade to calculate the next output voltage
                  write the output voltage to the
                  store the values in the MATLAB arrays
                  acknowledge the interrupt
      write the matlab file

caacade()
      set the input to the first biquad equal to the current analog input
      for each of the sections in the biquad strucuture
            calculate the output of the biquad
            increment the biquad
            place the output of the last biquad in the next

update_filter
      set the filter variables based on the discrete transfer function and
the current values of kp and ki

vel
      the first time this function is called
            set the previous counter and the current counter to the current
encoder value
```

```
      else calculate the difference between the current count and the
previous count
      set the previous counter equal to the current counter
      return the difference

setup_table
      set the timeoutValue units
      initialize the table values
      assign easier names
      assign the table to the thread resource structure

rpm2rad
      convert rpm to radians/sec

dbi_bti2rpm
      convert natural units of interrupt thread to rpm
```

VI. MATLAB SCRIPT

```
clear all; close all; clc;
load("RileyLab7.mat")

%time vector
time = [];
time(1) = 0;
for i = 1:length(Velocity)-1
    time(i+1) = Bti/1000*i; %time in seconds
    time = time';
end

%transfer function
s = tf('s');
cont = Kp*(s+Ki/Kp)/s;
Kvi = 0.41;
Kt = 0.11;
J = 3.8e-4;
B = 0; %assume very little damping
g1 = Kvi;
g2 = Kt;
g3 = 1/(J*s+B);
sys_vel = feedback(cont * g1 * g2 * g3,1);
sys_tor = cont * g1 * g2 / (1+cont*g1*g2*g3);

%compute theoretical velocity response
step_scale = (Current_Reference - Previous_Reference)*2*pi/60 ;
[theoretical_velocity,theoretical_velocity_time] = step(sys_vel);
theoretical_velocity = (theoretical_velocity * step_scale) + Velocity(1);

%compute theoretical torque response
step_scale = (Current_Reference - Previous_Reference)*2*pi/60 ;
[theoretical_torque,theoretical_torque_time] = step(sys_tor);
theoretical_torque = (theoretical_torque * step_scale);

subplot(2,1,1);
%plot the measured and theoretical torque
plot(time,Torque,'k.',theoretical_torque_time,theoretical_torque,'k');title("
Measured and Theoretical Control Values");
xlabel("Time (s)"); ylabel("Control Value (N\cdotM)");
legend("Measured","Theoretical");

subplot(2,1,2);
%plot the measured and theoretical velocitieis
plot(time,Velocity,'k.',theoretical_velocity_time,theoretical_velocity,'k');t
itle("Measured and Theoretical Control Values");
xlabel("Time (s)"); ylabel("Velocity (rad/sec)");
legend("Measured","Theoretical");
```