Lab #8 DC-Motor PID Position Control

1. Description

In this laboratory, we implemented a real-time PID controller that controls the position of a DC-motor. The program in this laboratory is similar to that of Lab #7, with the addition of using a MATLAB toolbox, pidtune(), to create the biquad structure necessary to implement the controller. In addition, we also learn how to interpret the position of the motor from the motor encoder, and providing an appropriate ramp input to guide the motor to go to a certain position through Sramps().

In the program, the main thread registers the Timer IRQ thread, defines the desired reference positions, and defines the information to be displayed in the LCD. Furthermore, the main thread calls the ctable2() function that displays the reference position, actual position, and the output voltage to the LCD screen. The main thread loops in ctable2, which can be exited through pressing the DEL key. When the DEL key is pressed, the main thread and the Timer IRQ thread are terminated.

The Timer IRQ thread initializes the IO channels, the encoder, and the Sramps() function. When the thread has not been terminated by main(), it waits for timeout and schedules the next interrupt. If the IRQ is asserted, then the reference position is calculated through Sramps(), the actual position is measured through pos(), the error (control input) is calculated and properly scaled. The input error is passed to the cascade function that calculates the control output based on the discrete transfer function defined from PIDF.h. Lastly, the output voltage is passed to the digital to analog converter to control the motor and the important information is saved to a MATLAB file. If the interrupt is terminated by main, the Timer IRQ exits the loop and imports the MATLAB file.

The limitation of this program is the inability to define the desired position through the LCD screen, rather it is all defined inside the code. Therefore, once the program is executed, we can only observe the motor moving to the reference position, and read the information in the screen, but nothing more.

Below is the functional hierarchy of this laboratory.

```
main()
|_ myRio_Open()
|_ Irq_RegisterTimerIrq()
|_ pthread_create()
    |_Timer_Irq_Thread()
        |_ Irq_Wait()
        |_ NiFpga_UWrite32()
        |_ NiFpga_WriteBool()
        |_ Sramps()
        |_ Aio_Read()
        |_ Aio_Write()
        |_ pos()
        |_ cascade()
        |_ Irq_Acknowledge()
        |_ pthread_exit()
|_ ctable2()
|_ pthread_join()
|_ Irq_UnregisterDiIrq()
```

2. Testing

In order to test the program, execute main.c in myLab8 folder. Ensure that PIDF.h and ctable2.h is included in the folder.

1. Check the screen, it should display information about the reference position, actual position, and the output digital to analog voltage.
2. The DC-motor should automatically turn into the desired position defined in the program. The reference position should increase incrementally in a ramp to a maximum position, then rotates the other direction back to the initial position.
3. Ensure the direction of rotation: initially, the motor should only rotate clockwise, then after the peak position, the motor should rotate counterclockwise.
4. Compare the value of the VDAout from the LCD screen, and the one in the osciloscope, and roughly confirm the actual position reading for sanity check.
5. Once the motor has returned to its original position, press the DEL key, and the motor should stop moving and the LCD screen should stop showing the table.
6. Export the MATLAB file and plot actual position and torque, and ensure that the results make sense. If it does not, check the pointers assigned to import variables to the MATLAB file.
7. Confirm the PIDF coefficients with the one obtained from the MATLAB file.

To test the MATLAB script that prepares the coefficients of the transfer function, perform the following tasks:

1. After the transfer function of the plant and the controller has been created, plot the step response of the position and torque transfer function
2. The results of the plots should all settle to a final value with no oscillations. The position should settle at 1 and the torque should settle at 0.

3. Results

The testing sequence went as expected. The main issue I encountered was with assigning the variables needed to import to MATLAB file. One of the early bug I experienced was that the DC motor made weird noises and the oscilloscope showed that the voltage bounces between saturation value. The reason for this is due to the pos() not being scaled when assigned to pref, which results in the inaccurate value for the control input, which is the error.

The MATLAB plots of the response is shown in the figure below.

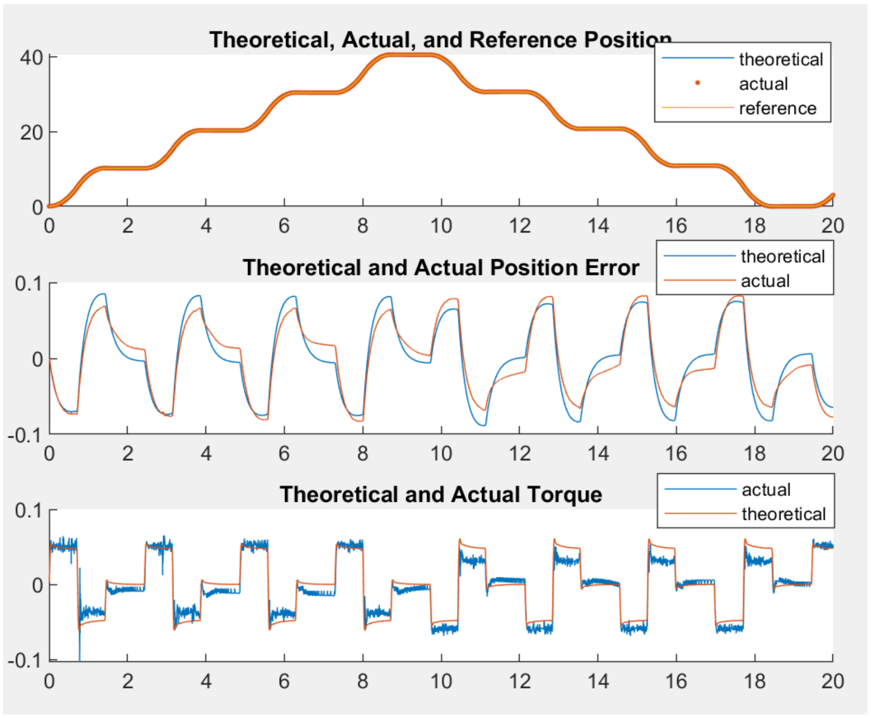

Figure 1: Position, Error, and Torque Plots

As observed from the first and second plot, the actual position closely resembles the reference position. Taking a closer look at the error plot, most of the time, the error of the actual position is lower than that of the theoretical position. One possible explanation for this is because in the theoretical system is assumed to have no damping, but damping is present in reality.

Moving on to the torque plots, the theoretical torque has higher overshoots than that of the measured. The same explanation that motor damping is present causes this occurrence. In addition, the actual torque plots fluctuate in the steady state. One possible reason is because of the quantization in taking position measurement. This will cause the error to continue to exist even though the final position has been reached, which will cause the system to continually correcting the motor position. Another explanation is that the vibration from the table causes mini disturbances that kept the system away from the steady state value.

One way that that this program can be improved is to add the capability to set the reference position, so that the user can change the desired motor position while the program is operational.