Jacob Sickafoose

Lab # 4

3/18/21

Description -

This lab required being able to run our DC motor at a controlled speed either using open loop control, or closed loop control. The lab started out needing to modify RotaryEncoder.c to allow interrupt driven sensing of position. An algorithm was then implemented to determine the speed at which the magnet was spinning, using the known distance it was spinning and a fixed time interval it was read at. This was then tested with the motor hooked directly into 12V power. The DCMotorControl.c file was then made in order to control the speed of the motor. This utilized the PWM mode of the OC module onboard the PIC32. After the speed was measurable, and the motor speed was controllable, the two were put together in the open loop application. FeedbackControl.c was then created to calculate the feedback control for the motor. This calculated the new speed for the motor to spin by looking at the current speed it was at, and comparing that to the requested speed of the motor. This would then give more or less power to drive the motor to what was requested.

Method -

Rotary Encoder -

RotaryEncoder.c had to be modified to include the option of interrupt control mode to the initialization. Using the SPI module in interrupt driven mode was fairly straightforward. The interrupts were the same, with the addition of changing mode, and adding an interrupt priority. Rather than sending the whole 10 steps that a ReadRawAngle required, the CS line just had to be driven low, then the interrupt would trigger and give a valid read which was then saved and the

CS line was driven back high. Timer2 was also added, set to interrupt on rollover every 1ms. This functioned to drive CS low and trigger a periodic SPI of the encoder position.

Our speed then had to be calculated using the newly interrupt driven method of finding angle. Now that angle was read at a period of 1ms, finding the speed was made easier. The equation for the speed was simply $\omega = \Delta\Theta/\Delta t$. With the Δt being a constant 1, all that was necessary was to find the change in $\Delta\Theta$ every rate check. A MAX_RATE value also needed to be calculated to account for the huge leap in angle the encoder would see on each rollover. The largest possible jump was calculated with 24V in mind, and every rate reported larger than that would be modified. The calculation of MAX_RATE went as follows: Max encoder counts per single tick $142 * 84 \approx 12{,}000$RPM $\longrightarrow$ 12,000RPM/60sec = 200RPsecond $\longrightarrow$ 200RPsecond * 0x3FFF(max angle reading) = 3,276,600 counts/sec $\longrightarrow$ (3,276,600 counts/sec)/500Hz = 6553.2 MAX_RATE. That MAX_RATE was then rounded up because any rollover would cause the rate to go WAY beyond what was calculated so it was best to go higher. The rate was already calculated with 24V in mind even with the motor only running on 12V. Finally, a test harness was made to see the change in speed when the motor was spun up, and turned off while directly hooked into 12V.

DC Motor Drive -

DCMotorDrive.c was made in a very similar way to the RCServo.c. The OC module was utilized once again, only this time in actual PWM mode. A duty cycle needed to be calculated. This turned our number from 0-1000 representing 0% - 100.0%, to a number of timer ticks before our signal could lower. With the module in PWM mode, an interrupt was no longer necessary as the OC3RS, the register controlling how long the pulse would be held, is set directly in the SetMotorSpeed() function. The duty cycle to timer ticks conversion was as simple as multiplying the duty cycle by 20. When the duty cycle input was a negative value, the motor was put in reverse by setting !IN1 and IN2, and the given duty cycle was multiplied by -20 to remove the negative before setting the OC3RS register to it. An SetBrake() function was also created, where it must set IN1 and IN2 to the same value to apply the H-gates built in brakes for the motor. Helper functions of setForward() and setBack() were also added to make the SetSpeed function more readable. With DCMotorControl.c fully implemented, an open loop tester application was then created setting the motor speed to whatever was requested of the motor, then reporting back the speed to the lab interface.

Feedback Control -

FeedbackControl.c simply required implementation of the given algorithm 2. This was implemented in the function, FeedbackControl_Update(). The only real issue ran into here was with the #defines. Parentheses were required to get -MAX_CONTROL_OUTPUT to work properly because the MAX_CONTROL_OUTPUT = 1<<FEEDBACK_MAXOUTPUT_POWER and it slipped by that this was the cause of error. The Init function serves simply to set all values to 0 because there are no onboard modules being utilized in the FeedbackControl. The rest of the functions are simply to interface with the black box, and set the internal gain values as well as to get them back, and reset some of the Update variables. The tester for this simply plugged the given values into their gains, then continuously updated, and reset for new values.

Lab 5 Closed Loop Application -

The final application required setting a commanded rate, calculating the current rate at 1KHz, then at 200Hz running the PID loop, scaling the output, and setting the motor speed. Writing the application was fairly simple. Writing the program step by step using the steps provided made the whole process easy to understand. The only issue encountered was scaling the PID loop output to a motor speed. The line: newSpeed = ((newSpeed * MAXMOTORSPEED) >> FEEDBACK_MAXOUTPUT_POWER) was used. This line was given to us, but it took a lot of time and energy before it was figured out, newSpeed had to be larger than an int in order for the large bit shift to work. One solution was to cast it to an (int64_t). For this lab, the value was simply initialized as a long long int from the get go, just in case.

<u>Conclusion</u> -

The rotary encoder is now able to operate in interrupt driven mode, as well as use another function to calculate the rate. The DC motor drive is now capable of setting a modulating the output pulse, based on the duty cycle input from 0-1000. It also can change directions based on the polarity of the duty cycle, as well as apply brakes through another function. The Feedback control is then capable of running the algorithm provided in order to calculate a new speed for the motor to run, based on the current speed, and the target. These three parts were then implemented in the final closed loop application.