Duy Cu

Jiaxiang Xu

11/24/2018

EE138

**Lab 5: Motor Control**

**Introduction**

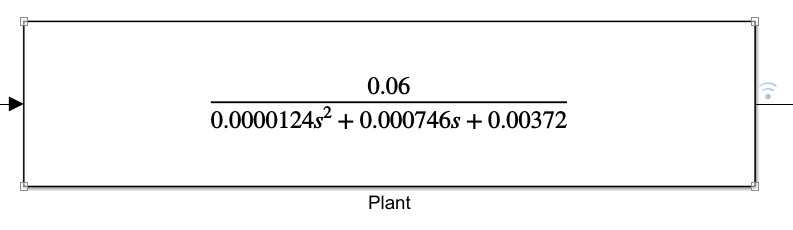
The purpose of this experiment is to model a PID controller using a SAM D20 microcontroller. PID controller is a mechanical controller that controls a system based on current, past, and predicted future position of the object. In this experiment, the SAM D20 microcontroller controls the motor using an implemented PID controller.

The first task is to use the PID controller to control the desired speed. The motor will be able to accelerate to 1500 RPM, the desired speed, in approximately 5 seconds, and return to its original RPM even with external disturbances. The motor will return to 0 RPM within 5 seconds if the button, “0” is pressed.

The second task is similar, using the PID to control the position of the motor. This can only be done when the motor speed is at 0 RPM. The PID will bring its position back to original when there is an external disturbance acted on the rotary.

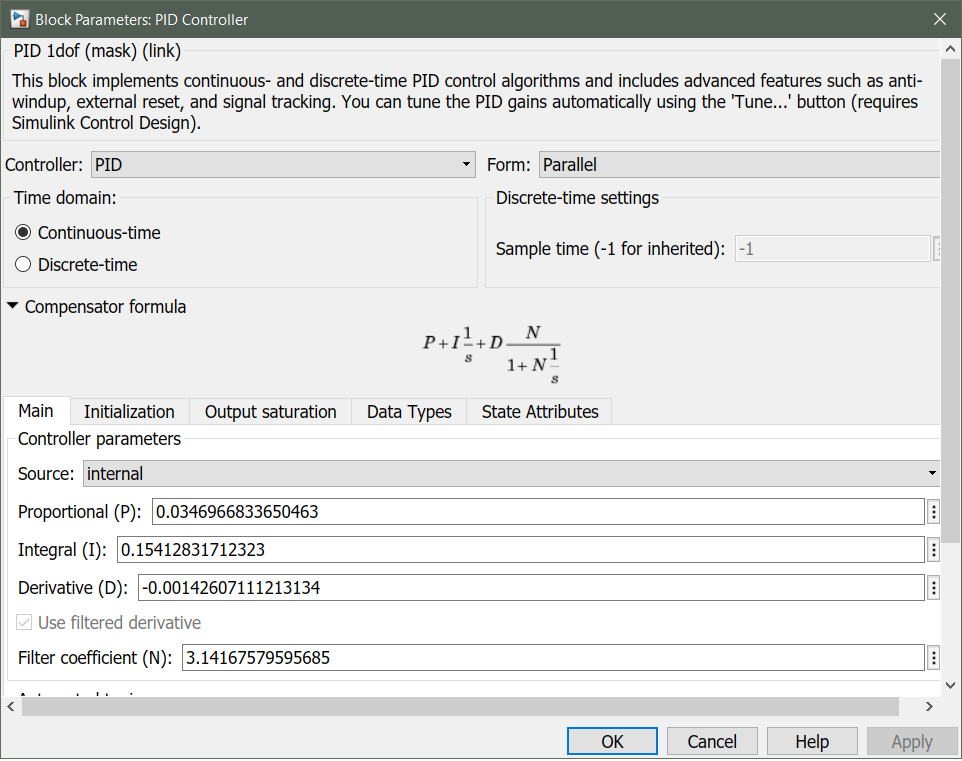
**Method**

For this lab, we can simulate the control by using Matlab. Things we need to consider are the parameters for the brushless DC motor so we can implement the plant transfer function in simulink. We used the parameters of the example from lecture notes and got a transfer function from Figure 1.



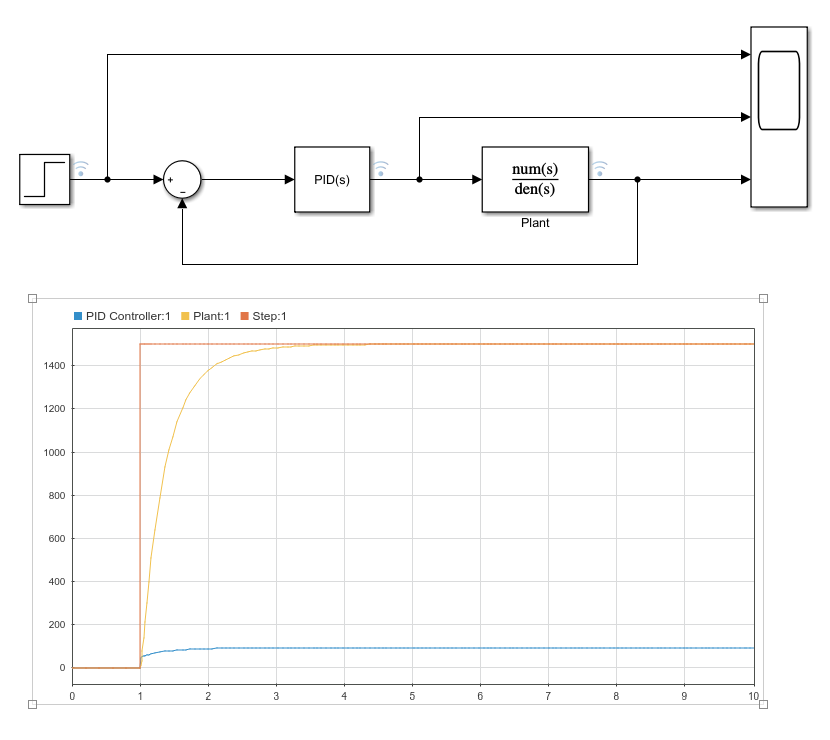
**Figure 1:** Transfer function for the Brushless DC motor

Since Matlab provide a PID function block, we took that and used it as our PID controller. Figure 2 shows the constants for the PID values:



**Figure 2:** PID Constants parameter

We include a scope that will show the input for the step response, the PID output (actual RPM values), and the plant output (line counter for every 0.005 seconds). Figure 3 shows the result from running the simulink. The scope shows very reasonable desired output so we went ahead and used these PID constants for our PID controller.



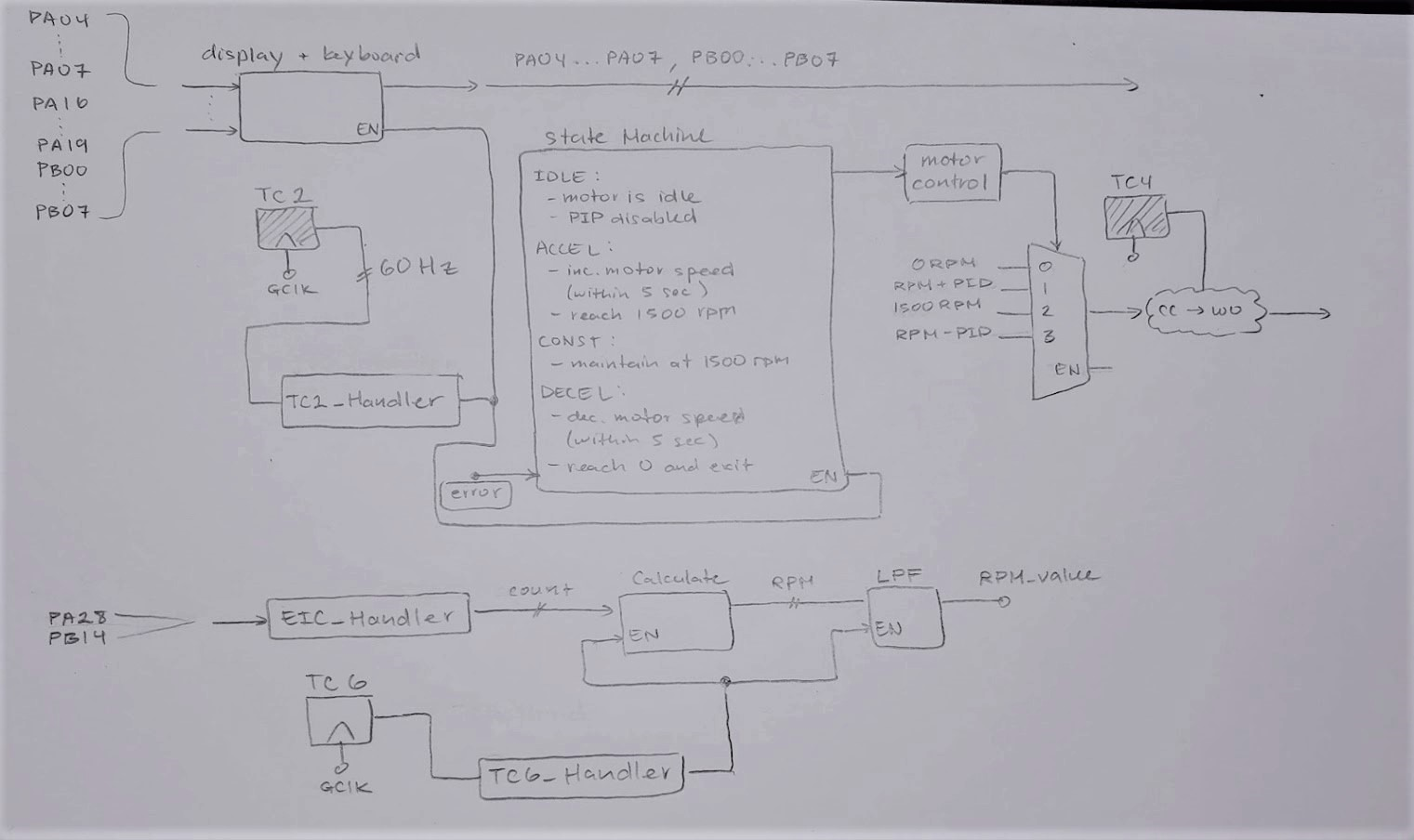
**Figure 3:** Step response of the PID controller + Plant

For the schematics of coding our lab, we used Figure 4 to generally map out what is needed for each function. Our code will consists of 3 timer counter (2 of which will have interrupts), EIC handler, digital LPF, and GPIO for detecting keypad and phase detection.

There will be a total of 3 FSM:

* FSM for the keyboard (running in 60 Hz interrupt)
* FSM for the motor control (running in 200 Hz interrupt)
* FSM for the PID control (running in 60 Hz interrupt)

The keyboard and the motor control FSM can be found in lab 1 and lab 3, so we only need to write PID control for this lab.



**Figure 4:** Schematics for Motor Control

The FSM for the PID should have a total of 5 state: IDLE, ACCEL, DECEL, SPEED CONTROL, and POSITION CONTROL. In one of these states, it will check the error value before changing the FSM of the motor control.

The FSM for the motor control will have a total of 5 state: MOTOR\_ACCEL, MOTOR\_DECEL, MOTOR\_0, MOTOR\_CW, and MOTOR\_CCW. The last two state is used for the position control.

**Result**

**Task 1: Speed Control**

The motor accelerates to approximately 1300 RPM in saturation under 5 seconds. The acceleration portion of the experiment did not meet the expected requirement of acceleration to 1500 RPM in 5 sections. When in position mode, the motor is able to return to its original position when it is moved by external force (fingers).

A debug session for acceleration portion of the experiment is conducted. The result of the debug session shows that the PID value when the motor accelerations to 1300 RPM is small. Therefor, the speed of the motor is actually increasing but very slowly. A proper scaling of the PID value is needed before writing into CC registers in order to avoid accelerating in a slow rate.

We first used the PID constants from Simulink but the motor did not function properly. We have to play around with the values for quite a long time in order to get the desired result. We later found out that the PID calculation values (PID\_p, PID\_i, PID\_d ) needs to be a float data type in order to have a more accurate PID values. The integral values of the PID also needs to reset to 0 every time the control is set to 0 RPM. One of the reason why we need to do this is because the PID controller in this lab is not accurate (we approximate when we calculate the integral and the derivative values). The PID values can increase extremely high and thus cause a high spike in both speed and position control, so we implement a saturation level to prevent the PID to reach a certain value. This way, the speed control will stay within the 0-255 mark (TC4 is an 8-bit motor control) when something is constantly blocking the motor.

**Task 2: Position Control**

This was also the case for the position control. We have to manually reset the integral values every time it reaches its destination. We also saterate the PID values for the position control just to prevent it getting bigger and bigger when someone holds a position for a long time.

**Conclusion**

In conclusion, the PID controller works as expected and control both the position and speed really well. The PID values can we adjusted a little more for a better response but this can be time consuming. This PID application can be used in a lot of control systems to constantly stabilize the system in a certain orientation.