DC Motor Control 결과보고서

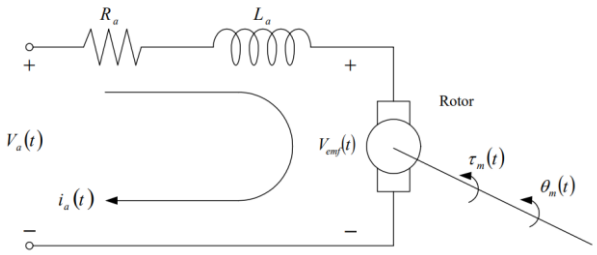
20101449 한가은

목차

1. 서론

1.1 원리 및 이론

DC motor란 고정자로 영구자석을 사용하고, 회전자로 코일을 사용하여 구성한 것으로, 전기자에 흐르는 전류의 방향을 반전시킴으로써 자력의 반발, 흡인력으로 회전력을 생성시키는 모터이다. DC motor의 구조는 다음과 같은 회로도로 나타낼 수 있다.



[그림 1: DC motor의 구조]

위의 회로도로부터, 입력 전압에 대한 모터 회전 각도의 전달함수로써 DC motor의 위치 모델을 나타낼 수 있고, 이는 다음 수식과 같다.

$$T_p(s) = \frac{\theta_m(s)}{V_a(s)} = \frac{K}{s(s+\alpha)}$$

위치 모델에서 적분기 $\frac{1}{s}$ 이 포함되어 있는 점을 통해, 일정한 전압이 인가되면 회전 각도는 누적되는, DC motor의 특성을 엿볼 수 있다.

또한, 입력 전압에 대한 모터 회전 각속도의 전달함수로써 DC motor의 속도 모델을 나타낼 수 있고, 이는 다음 수식과 같다.

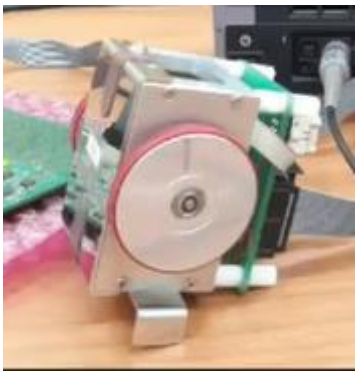
$$T_v(s) = \frac{\omega_m(s)}{V_a(s)} = \frac{K}{s+\alpha}$$

속도 모델에 V_{in}의 step response를 인가하고, final value theorem을 적용하면, $\frac{K}{\alpha}V_{in}$으로, 즉 모터의 각속도는 입력 전압 V_{in}과 비례하다는 결과를 볼 수 있다. 이는, 큰 전압을 인가할수록 큰 각속도로 빨리 회전하고, (-) 입력 전압을 인가하면 (-)의 각속도로 반대 방향으로 회전하는 모터의 특성을 보여준다.

1.2 실험 목적

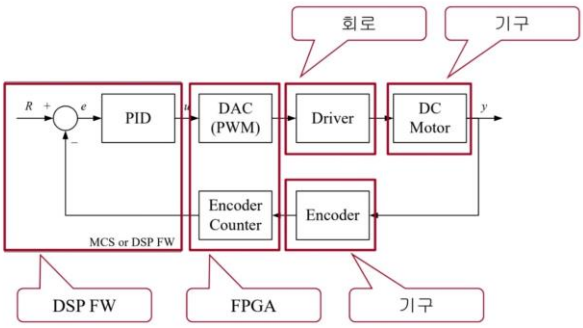
Feedback control structure을 통해 PWM 구동 방식의 DC motor position을 제어한다. 제어 목표는 목표 위치 R과, 시스템 출력 y의 오차를 0으로 만드는 것이고, 이를 위해 PID controller를 설계하고, tracking 및 2 DOF 제어 기법 도입을 도입하고자 한다. 따라서, 본 실험에서는 PID controller의 각 gain값을 조정해보며 step response 확인을 통해 시스템 특성의 변화를 살피고, 시스템의 목표 stability와 sensitivity를 만족시키는 PID controller를 설계한다. 더 나아가, 부드러운 동작을 구현할 수 있도록 사다리꼴 속도 프로파일에 대응하는 위치 궤적을 사용한 position 제어를 설계하고, 2 DOF 제어 구조를 도입하여 제어 성능을 향상시킨다.

1.3. 실험 대상 및 배경



[그림 2: 실험 대상 DC motor]

실험에 사용된 DC motor는 PWM 방식으로 구동되며, feedback 제어를 위한 magnetic incremental encoder가 부착되어 있다. 부착된 encoder는 모터 1회전 당 512개의 펄스를 발생시키며, 90도 위상차를 갖는 A, B 두 상 신호를 출력한다. 또한, DC motor와 바퀴 사이에 7.5:1 비율의 기어가 부착되어 있다.



[그림 3: 실험 대상 DC motor]

실험의 전체 제어 시스템 구조는 [그림 3]과 같다. 본 실험에서는 FPGA보드를 이용해 PWM 신호를 발생시키고, encoder counter 값을 확인하며, DSP 펌웨어를 통해 PID controller와 tracking control 기법, 2DOF 제어 기법을 도입한다.

1.4. 실험 방법

Simulink를 통해 진행했던 모의 실험을 토대로 확인한 DC motor position 제어 기법을 바탕으로, PID control, tracking control, 2DOF control 기법을 통해 제어 목적을 달성하고자 한다. 따라서, 각 제어기 도입을 위해 DSP firmware 코드를 작성하는 것이 실험 진행 방법에 해당한다. 즉, PWM을 이용하여 DC 모터를 구동하기 위한 코드, encoder count값과 모터 회전 각도 사이의 변환을 위한 코드, PID 제어기, tracking 제어 기법을 위한 위치 궤적, feedforward 제어기를 펌웨어로 작성하여 제어 목적을 달성한다.

2. 실험 절차 및 결과

2.1. 실험 절차 및 내용

실험 절차 및 각 내용은 다음과 같다.

I. PWM을 이용한 DC 모터 구동

- PWM 발생 회로는 FPGA에 Verilog로 구현되어 있다. 즉, FPGA의 PWMRIGHT 레지스터의 하위 12bit 값을 통해 PWM duty를 조절할 수 있다.
- 레지스터 값을 직접 변경하는 것은 시스템 동작 이상을 불러올 위험이 있으며, 0x800이 기준 정지 동작이 되는 것은 직관에 어긋난다.
- 따라서, DSP firmware에 *PWMOut* 함수를 작성하여 -50.0~50.0 범위의 값을 인가했을 때, PWM 파형이 0%~100% 듀티비로 발생하도록 한다.
- 함수가 의도한 대로 작성되었는지 확인하기 위해, oscilloscope로 출력을 확인한다.

II. Magnetic encoder 신호를 이용한 위치 측정

- 위치 제어가 가능하기 위해서는, feedback loop이 구성되어야 한다
- 따라서, encoder count값을 모터 회전 각도로 변환시켜 이를 feedback으로 구성한다.
- 모터 1회전 당 512개의 pulse가 발생한다는 점과, 모터와 바퀴 사이 기어비가 7.5:1인 점을 고려하여 코드를 작성하고, J4의 4번과 5번 핀을 통해 encoder 파형을 관찰한다.

III. PID 제어기 구현

- 1kHz의 주기로 발생하는 타이머 인터럽트를 활용하여 PID 제어기를 구현한다. 이보다 PWM 파형의 주파수가 더 빨라야 한다는 점을 유의한다.
- 각 PID gain값을 변경시켜보며, step response를 확인하여 stability 및 sensitivity 목표를 만족시킨다.

IV. Tracking 제어 기법을 위한 위치 궤적 구현

- 부드러운 동작을 위해 사다리꼴 속도 프로파일에 대응하는 위치 궤적을 구현한다.

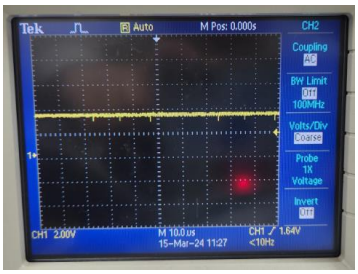
V. Feedforward 제어기 구현

- Feedforward 제어기를 구현하여 tracking 제어 시, PID gain값을 높이지 않고 tracking 오차를 효과적으로 줄일 수 있음을 확인한다.

2.2. 실험 결과 및 해석

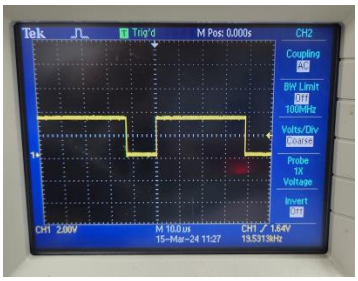
I. PWM을 이용한 DC 모터 구동

-50.0~50.0값을 인가하여 PWM duty를 발생시키는 *PWMOut* 함수가 잘 작성되었는지, oscilloscope로 그 출력을 확인한 결과는 다음과 같다.



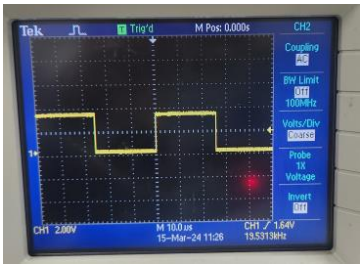
[그림 4: 100% PWM duty]

50.0을 PWMOut 함수의 argument로 인가한 결과로, 의도한 100% PWM duty가 발생함을 확인하였다.



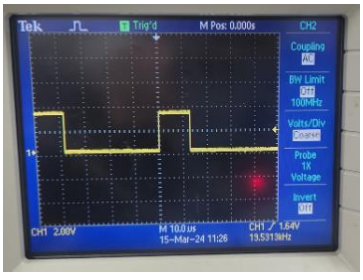
[그림 5: 75% PWM duty]

25.0을 PWMOut 함수의 argument로 인가한 결과로, 의도한 75% PWM duty가 발생함을 확인하였다.



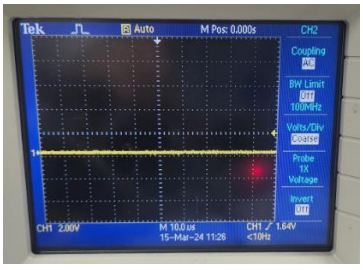
[그림 6: 100% PWM duty]

0.0을 PWMOut 함수의 argument로 인가한 결과로, 의도한 50% PWM duty가 발생함을 확인하였다.



[그림 7: 100% PWM duty]

-25.0을 PWMOut 함수의 argument로 인가한 결과로, 의도한 25% PWM duty가 발생함을 확인하였다.

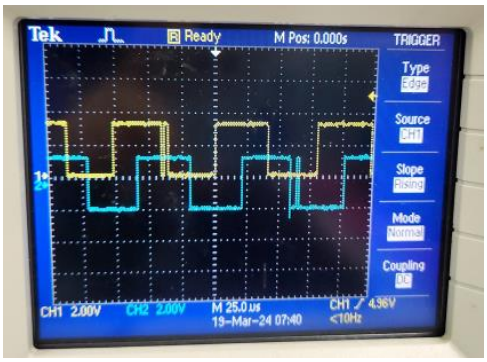


[그림 8: 100% PWM duty]

-50.0을 PWMOut 함수의 argument로 인가한 결과로, 의도한 0% PWM duty가 발생함을 확인하였다.

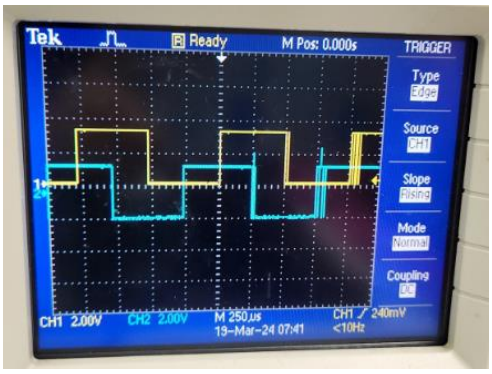
II. Magnetic encoder 신호를 이용한 위치 측정

노란색의 CH1 출력을 A상, 파란색의 CH2 출력을 B상이라 하였을 때, 바퀴를 손으로 돌려 encoder의 파형을 관찰한 결과는 다음과 같다.



[그림 9: CW 방향 회전]

바퀴를 시계 방향으로 회전시켰을 때, A상의 위상이 90도 앞서는 것을 확인하였다.



[그림 10: CCW 방향 회전]

바퀴를 반시계 방향으로 회전시켰을 때, B상의 위상이 90도 앞서는 것을 확인하였다.

이에 모터 회전축의 회전 각도를 측정하기 위해 *GetAngle* 함수를 작성하였고, 시계 방향으로약 24도, 시계 반대 방향으로 약 180도까지 회전시켜 본 결과는 다음과 같다.

```
ENCPOSR value: 0x0000(    0)
current pos:   0.00
ENCPOSR value: 0xfff1(65521)
current pos:  -1.50
ENCPOSR value: 0xff41(65345)
current pos: -17.91
ENCPOSR value: 0xfeff(65279)
current pos: -24.09
ENCPOSR value: 0xff80(65408)
current pos: -11.91
ENCPOSR value: 0xfff2(65522)
current pos:  -1.22
ENCPOSR value: 0x008c(  140)
current pos:  13.22
ENCPOSR value: 0x015e(  350)
current pos:  33.00
ENCPOSR value: 0x0285(  645)
current pos:  60.66
ENCPOSR value: 0x035f(  863)
current pos:  81.09
ENCPOSR value: 0x04c2( 1218)
current pos: 114.28
ENCPOSR value: 0x0680( 1664)
current pos: 156.09
ENCPOSR value: 0x0788( 1928)
current pos: 180.75
```

[그림 11: 모터 회전 시 encoder count값 변화와 각도 변화]

실험에 사용된 모터에는 모터와 바퀴 사이 7.5:1 비율의 기어가 장착되어 있고, 모터 1회전 당 encoder 펄스 카운터 값이 512만큼 변화하기 때문에, 바퀴 1회전(360도) 당 $512 \times 7.5 = 3840$만큼 값이 변화해야 한다는 것을 알 수 있다. 즉, 바퀴 180.75도 회전 시 가져야 하는 값을 수식으로 계산하면 다음과 같다.

$$\frac{512 \times 7.5}{360} \times 180 = 1928$$

위의 실험 결과 바퀴 180.75도 회전 시 1928의 값을 기록하였으므로, 실험 결과가 정확하다는 것을 확인할 수 있었다.

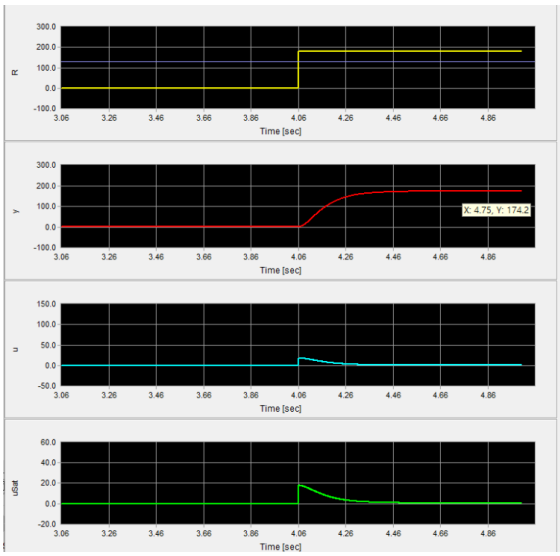
III. PID 제어기 구현

Timer ISR에서 PID 제어기를 코딩하고, 각 gain값을 변경시켜보며 step response를 확인하였다. 이에 overshoot, sensitivity, rising time을 고려하여 최종적으로 각 gain값을 $K_p = 0.35, K_D = 4.5, K_I = 0.0$ 으로 선정하였다. 위 실험의 조건은 다음과 같다.

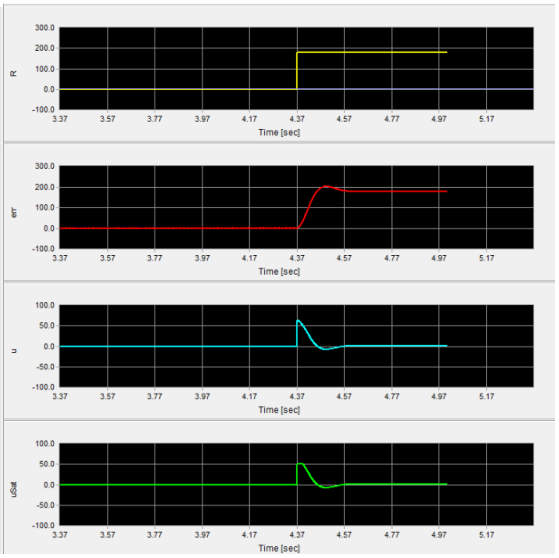
- Reference position: 2초마다 0도와 180도 반전
- Saturation: 제어 입력 u값은 -50~50으로 제한됨.
- Each channel of USB Monitor:
  Channel 0: 목표 각도 'R
  Channel 1: 시스템 출력 'y',
  Channel 2: 시스템의 제어 입력 'u',
  Channel 3: 포화된 제어 입력 'uSat'

각 gain값을 선정한 실험 매커니즘은 다음의 1~5 과정과 같다.

1. K_p값의 변화에 따른 시스템 응답



[그림 12: $K_p = 0.1, K_D = 0.0, K_I = 0.0$]



[그림 13: $K_p = 0.35, K_D = 0.0, K_I = 0.0$]

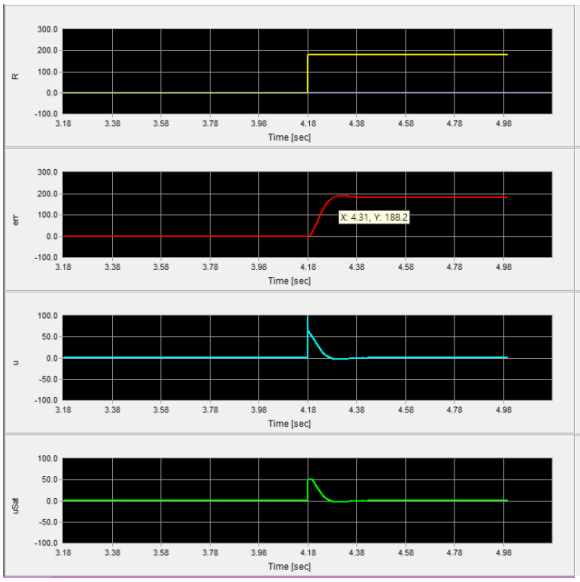
[그림 12, 13]에서의 각 percent overshoot과 steady-state error을 비교하면 다음과 같다.

PID gain	Overshoot	Steady-state error
$K_p = 0.1, K_D = 0.0, K_I = 0.0$	0%	3.22%
$K_p = 0.35, K_D = 0.0, K_I = 0.0$	12.24%	0.61%

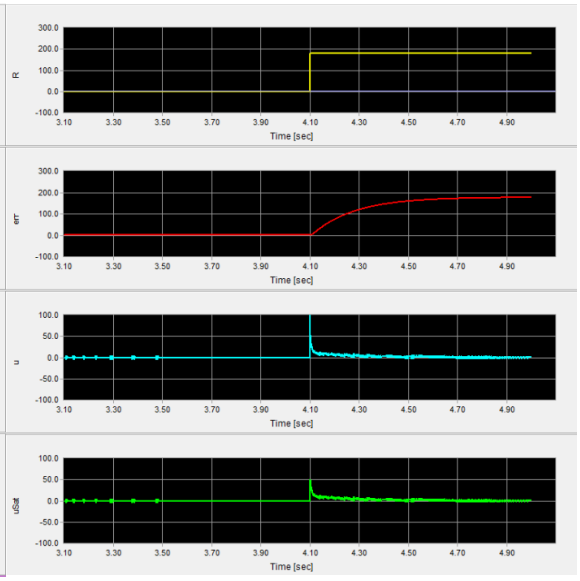
[표1: P gain에 따른 overshoot, ess값 비교]

P gain의 초기값을 $K_p = 0.1$의 작은 값으로 설정하고 실험을 시작하였다. [그림 12]로부터, K_p가 작은 경우 응답 속도가 느리고, 정상상태 큰 것을 알 수 있다. 따라서 이를 개선하고자 K_p값을 증가시켜보며 시스템 응답을 확인하였다. 과도한 P gain은 overshoot을 증가시켜 시스템의 불안정을 초래한다는 것을 확인하고, saturation이 발생하지 않는 범위인 $K_p = 0.35$로 설정하였다. 그 결과는 [그림 13]과 같고, P gain의 도입으로 발생한 overshoot을 해소하기 위해 D gain 도입이 필요함을 보인다.

2. K_D값의 변화에 따른 시스템 응답



[그림 14: $K_p = 0.35, K_D = 3.0, K_I = 0.0$]



[그림 15: $K_p = 0.35, K_D = 50.0, K_I = 0.0$]

[그림 14, 15]에서의 각 percent overshoot과 steady-state error을 비교하면 다음과 같다.

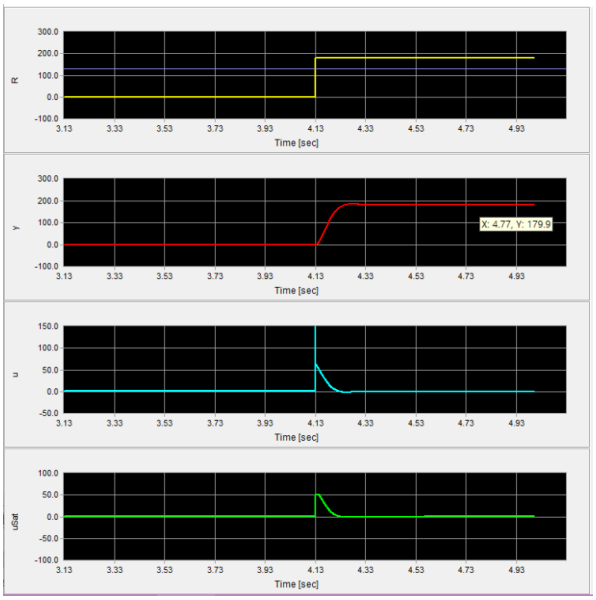
PID gain	Overshoot	Steady-state error
$K_p = 0.35, K_D = 3.0, K_I = 0.0$	4.58%	0.33%
$K_p = 0.35, K_D = 50.0, K_I = 0.0$	-1.04%	0.11%

[표2: D gain에 따른 overshoot, ess값 비교]

$K_D = 3.0$ 의 적절한 D gain을 도입하였을 때, overshoot이 감소하는 것을 확인하였다. 이에 D gain을 점점 키워보며 시스템 응답의 변화를 관찰하고, 과도한 D gain이 불러오는 문제를 분석하고자 하였다. 실험 결과, D gain을 증가시키자 overshoot은 감소하나, 시스템이 노이즈에 민감해진다는 것을 알 수 있었다. 이는 [그림 15]에서, 오차가 0인 부분에서 제어 입력 u값이 발생하는 것을 통해 알 수 있다. PID 각 이득 값이 $K_p = 0.35, K_D = 50.0, K_I = 0.0$이고, 오차가 0인 상황에서 이론적으로 P gain과 I gain에 의한 u 값은 발생할 수 없으므로, 이는 시스템이 노이즈에 민감해져 D gain에 의해 제어 입력 u값이 발생했다고 분석할 수 있기 때문이다.

3. P gain, D gain tunning

1~2의 실험 과정을 통해, 시스템 응답의 overshoot, 응답 시간, noise 민감도의 trade-off를 고려하여 $K_p = 0.35, K_D = 4.5, K_I = 0.0$로 p gain과 d gain을 선정하였다.



[그림 16: $K_p = 0.35, K_D = 4.5, K_I = 0.0$인 시스템의 응답]

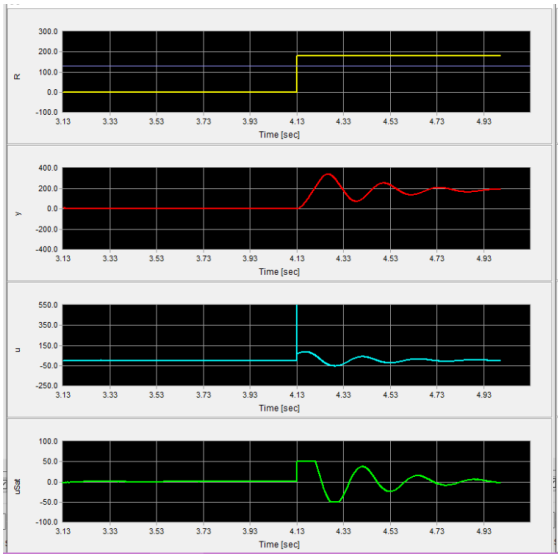
결정한 P gain, D gain에서의 percent overshoot과 steady-state error는 다음과 같다.

PID gain	Overshoot	Steady-state error
$K_p = 0.35, K_D = 4.5, K_I = 0.0$	1.93%	0.05%

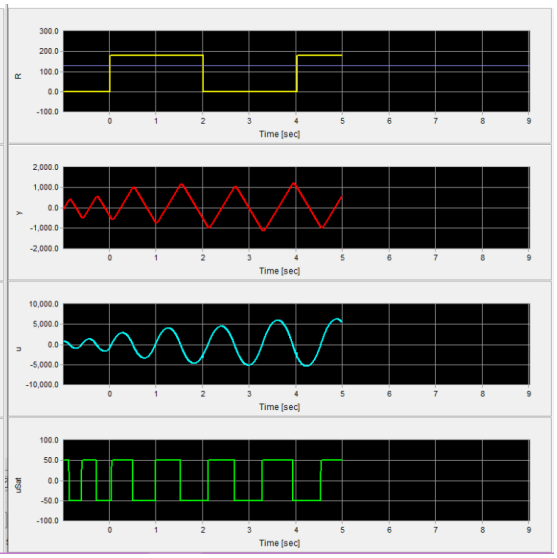
[표3: 선정한 p, d gain값에서의 시스템 응답 결과]

4. K_I의 도입

K_P, K_D튜닝 후, steady-state error을 0으로 만들어보고자 I gain을 도입해보았다.



[그림 17: $K_p = 0.35, K_D = 4.5, K_I = 0.01$]



[그림 18: $K_p = 0.35, K_D = 4.5, K_I = 0.03$]

$K_I = 0.01$의 작은 값으로 초기값을 선정하였는데도 불구하고, 시스템에 oscillation이 발생한 것을 확인할 수 있었다. 이에 PID 출력 u를 확인해보면, 순간적으로 873.8의 큰 값이 나가고 있는 것을 확인할 수 있었고, 이는 PID 수식에서 $K_I \int_0^t e(\tau)d\tau$에 큰 값이 누적되어 발생한 현상으로 이해할 수 있었다. Windup 현상을 확인해보고자 $K_I = 0.03$으로 값을 증가시켰고, 그 결과 [그림 18]에서 알 수 있듯이, PID 수식의 오차 적분값에 쌓인 값을 해소하기 위해 느리게 oscillation하는 PID 출력 u를 얻어 시스템이 불안정해짐을 확인할 수 있었다. 따라서 실제 모터 동작의 경우, 2초마다 0도와 180도를 반전시키도록 R값을 부여하였지만, windup 현상으로 멈추지 않고 계속 회전하는 동작을 볼 수 있었다.

5. 최종 PID gain 튜닝

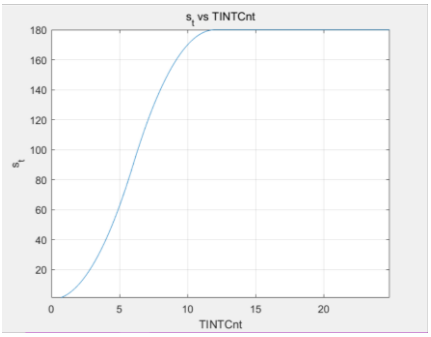
시스템이 성능을 향상시키는 K_I값을 찾아보고자 다른 이득 값은 유지한 채, $K_I = 0.000001$까지 줄여보았지만, 여전히 PID 출력 u값이 순간적으로 875.3의 큰 값을 갖는 모습을 볼 수 있었다. Percent overshoot, 정상상태 오차 또한 개선사항이 뚜렷하지 않았기에, 최종적으로 $K_p = 0.35, K_D = 4.5, K_I = 0.0$의 각 이득 값으로 PID gain 튜닝을 종료하였다.

IV. Tracking 제어 기법을 위한 위치 궤적 구현

Position 제어를 위해 step 입력을 인가한다는 것은, 순간적으로 속도가 무한대에 달하는 의미를 갖는다. 즉, 실제 시스템에서 목표 위치를 스텝 입력으로 인가하면 안전 및 기구 부식 측면에서 위험성을 발생시키기에 부드러운 동작을 구현하고자 목표 각도를 시간에 대한 궤적으로 적용하였다. 위 실험의 조건은 다음과 같다.

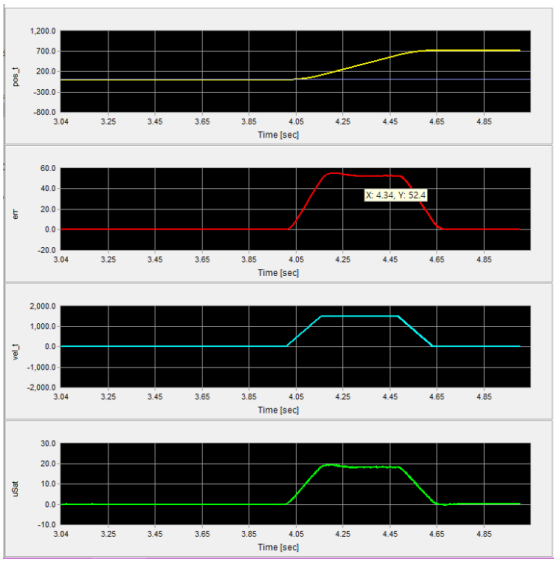
- Reference position: 720 degree, 가속도: 10000, 속도: 1500
- PID gain: $K_p = 0.45, K_D = 3.5, K_I = 0.0$
- Saturation: 제어 입력 u값은 -50~50으로 제한됨.
- Each channel of USB Monitor:
  Channel 0: 시간에 대한 목표 각도 입력 'pos_t'
  Channel 1: pos_t와 시스템 출력의 차이인 'err',
  Channel 2: 사다리꼴 속도 궤적 'vel_t',
  Channel 3: 포화된 시스템 제어 입력 'uSat'

시간에 대한 목표 위치 궤적을 만드는 *GetRefAngle* 함수를 작성한 뒤, 동작의 검증을 위해 MATLAB으로 위치 궤적의 목표 각도를 180도, 가속도를 5, 최대 속도를 100으로 설정하여 생성된 위치 궤적은 [그림 19]와 같다.



[그림 19: MATLAB을 통한 위치 궤적 생성 시뮬레이션 결과]

GetRefAngle 함수의 동작을 검증한 뒤, 목표 각도를 720도, 가속도를 10000, 최대 속도를 1500으로 설정하여 실제 모터에 인가한 결과는 [그림 20]과 같다. 이 때 PID 각 gain 값은 앞서 tunning한 결과인 $K_p = 0.35, K_D = 4.5, K_I = 0.0$을 사용하였다.



[그림 20: Reference position을 위치 궤적을 인가한 결과]

사전 계산을 통해 적절한 모양의 사다리꼴 속도 프로파일을 갖도록 목표 각도 720도, 가속도 10000, 최대 속도 1500으로 설정하였다. 이 경우, 가속, 등속, 감속 구간이 가져야 하는 시간 t, 위치 s를 계산하여 실험 결과와 비교하여 검증한 결과는 다음과 같다.

- 가속 구간에서의 t_1, s_1

$$t1 = \frac{1500.0}{10000.0} = 0.15$$
$$s1 = \frac{1500.0^2}{2 \cdot 10000.0} = 112.5$$

- 등속 구간에서의 t_2, s_2

$$t2 = \frac{720.0}{1500.0} = 0.48$$
$$s2 = 720.0 - 112.5 = 607.5$$

- 감속 구간에서의 t_3, s_3

$$t3 = t1 + t2 = 0.15 + 0.48 = 0.63$$
$$s3 = 720$$

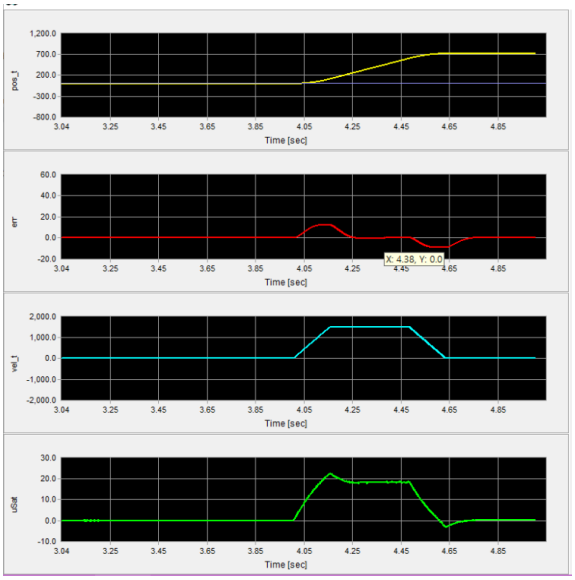
실험 결과와 계산 결과를 대조하여 일치함을 보여, 타당한 실험 결과를 얻었음을 확인하였다.

V. Feedforward 제어기 구현

System에 saturation block이 있는 경우, 제어 입력 u값의 최대값엔 한계가 있다. 이로 인하여, 원하는 목표 값에 도달할 수 없는 상황이 발생한다. 또는, 과도하게 큰 u값으로 인해 시스템이 불안정해지는 현상이 발생할 수 있다. 따라서 PID 각 gain을 크게 키우지 않고 시스템의 오차를 줄일 수 있는 방안을 고안해야 하고, 이에 발생하는 에러와 비슷한 모양을 갖는 사다리꼴 프로파일의 속도 궤적 feedforward 제어기로 추가하여, PID 제어기는 잔존오차를 제거하는 용도로 사용하는 2 DOF 제어 구조를 도입하였다. 위 실험의 조건은 다음과 같다.

- Reference position: 720 degree, 가속도: 10000, 속도: 1500
- PID gain: $K_p = 0.45, K_D = 3.5, K_I = 0.0$
- Feedforward gain: $K_{ff} = 0.01225$
- Saturation: 제어 입력 u값은 -50~50으로 제한됨.
- Each channel of USB Monitor:
  Channel 0: 시간에 대한 목표 각도 입력 'pos_t'
  Channel 1: pos_t와 시스템 출력의 차이인 'err',
  Channel 2: 사다리꼴 속도 궤적 'vel_t',
  Channel 3: 포화된 시스템 제어 입력 'uSat'

Tracking 오차가 0이 되는 Feedforward gain $K_{ff} = 0.01225$ 이었고, 이 때 실험 결과는 [그림 21]과 같다.



[그림 21: Reference position을 위치 궤적을 인가한 결과]

3. 결론

3.1. 오차 분석

'*II. Magnetic encoder 신호를 이용한 위치 측정*' 실험 과정에서 발생한 오차에 대해 분석하고자 한다. 바퀴를 손으로 돌려 encoder의 파형을 관찰하는 과정에서, oscilloscope로 측정한 결과, 파형에 glitch가 발생한 것을 확인할 수 있었다.

[그림 22: encoder 파형에 glitch가 발생한 모습]

이에 glitch 발생 원인을 분석한 결과, 크게 세 가지의 원인을 알 수 있었다.

1. 기어와 기어 사이의 틈새가 벌어진, backlash로 인한 glitch 발생
2. 모터 자체의 결함으로, 일정한 PWM 신호가 들어옴에도 일정한 속도로 회전하지 않아 glitch 발생
3. 기구의 aging 이슈로 인한 glitch 발생

따라서 기구적인 문제가 glitch 발생의 주 원인임을 알 수 있었고, 이러한 경우 노후화된 부품을 교체하는 등의 방법으로 오차 발생 원인을 해결할 수 있을 것으로 판단된다. '

3.2. 실험 결과에 대한 고찰

각 실험에서의 고찰은 다음과 같다.

VI. PWM을 이용한 DC 모터 구동 실험에서의 고찰

- PID 연산 결과는 모터에 인가할 전압 값으로, 이를 PWM의 duty를 바꿔주는 register에 바로 작성하면 오작동이 발생한다는 것을 알 수 있었다. 따라서 의도한 전압 값을 register에 쓰일 값으로 변환시키는 과정이 필요함을 알 수 있었다.

VII. Magnetic encoder 신호를 이용한 위치 측정 실험에서의 고찰

- 실험에 사용된 encoder가 incremental 방식이었기 때문에, 회전 각도의 초기화 과정이 반드시 필요한 점을 알 수 있었다.
- 모터 회전축의 회전 각도를 측정하기 위해 *GetAngle* 함수를 작성할 때, encoder count 값을 읽어 들이는 ENCPOSR 레지스터의 하위 16 bit만을 사용하기 때문에 16 bit인 short 자료형을 쓰면 편리한 코드 구현이 가능하다는 점을 알 수 있었다.

VIII. PID 제어기 구현

- PWM 주기가 제어를 위한 샘플링 주파수보다 빨라야 정상적으로 제어 입력 u값을 모터에 인가할 수 있다는 것을 알 수 있었다.
- 타이머 인터럽트의 ISR 함수에는 최소한의 연산만을 작성해야 한다는 것을 알 수 있었다. 특히 ISR 내부에 *MACRO_PRINT* 를 사용한 결과, 출력을 수행하는 데 걸리는 시간으로 의도한 1kHz의 샘플링 주파수가 틀어져 원하는 결과를 얻지 못함을 확인하였다.
- 또한, PID gain tunning 과정에서 다른 gain 값을 통제하고, 특정 gain만 과도하게 줄이거나, 키워 봄으로써 각 gain의 역할을 파악할 수 있었다. 특히 시스템에 saturation block이 존재하고, 적분기가 포함되어 있는 DC motor의 위치 모델의 제어 시스템에 I gain이 도입되면 windup 현상이 일어날 수 있음을 확인할 수 있었다. 이에 Anti-windup control 기법을 조사하는 계기를 가질 수 있었다.

IX. Tracking 제어 기법을 위한 위치 궤적 구현

- c언어로 프로그램을 작성하며, 자료형의 중요성을 이해할 수 있었다. 수식 계산에서, 0.5의 값을 의도하고 1/2를 코드로 작성한 결과, 0이 되어 결과가 틀어지는 것을 확인할 수 있었다. 이를 방지하고자, 1.0/2와 같은 형태로 코드를 작성해야 함을 유의할 수 있었다.

X. Feedforward 제어기 구현

- Feedforward 제어기를 통해 PID gain값을 높이지 않고 tracking 오차를 효과적으로 줄이고, PID 제어기는 잔존오차를 제거하는 용도로 사용할 수 있다는 점을 이해할 수 있었다.

4. 참고문헌

- 메카트로닉스(류정래 / 복두출판사 / 2012)
- MathWorks - PID Controller 블록을 사용한 안티와인드업 제어[1]
- Wikipedia - Feed forward (control)[2]

[1] https://kr.mathworks.com/help/simulink/slref/anti-windup-control-using-a-pid-controller.html

[2] https://en.wikipedia.org/wiki/Feed_forward_(control)