EE682 HW 1

Hyungtae Lim

September 20th, 2018

1 Before Getting Started...

HW♯1의 안내서에 정의되어 있는 2% settling time를 코드에 timeStep으로 계산하게 되면 iteration i가 1200번이 되어야 할 정도로 굉장히 큰 값이 되어 버린다. 따라서 상기되어 있는 *"rising time 15.0 sec"*와 *"2% settling time 60.0 sec"*의 sec는 for 문을 돌 때의 i를 unit을 칭하는 것이라고 가정한다.

2 PID Controller

현재 system의 조건은 다음과 같다.

1. rising time ≤ 15.0
2. Overshoot ≤ 10.0
3. 2%settling time ≤ 60.0

따라서, 위의 조건을 만족시키는 p, i, d gain을 먼저 tuning 해보았다. 상기의 조건들보다 작은 경우는 성능이 더 좋은 경우이므로, 위의 조건을 만족한다고 가정하였다. 먼저 기본 코드에서 주어진 p, i, d gain으로 위의 세 변수들을 측정한 결과, error를 보정하는 정도가 너무 적어서 overshoot도 일어나지 않았고, settling time도 존재하지 않았다. 그래서 p gain을 키운 후 다시 측정한 결과 이번엔 overshoot이 커졌고, settling time도 발생한 것을 확인할 수 있다. 또한 d gain을 키우게 되면 overshoot, rising time, settling time 모두 감소하면서 성능이 개선되는데, 우리의 design의 경우는 적절히 settling time을 60.0이 되게끔 적절히 잡아야 하기 때문에 마냥 d gain을 키우는 게 옳은 튜닝은 아니었다. 또한 i gain을 키우게 되면 성능이 개선하기는 하지만 i gain은 차이를 적분하기 때문에 그래프의 개형에서 fluctuation이 발생하는 것을 확인할 수 있었다.

3 FLC

3.1 Tuning Given FLC

먼저 각각의 K_e, K_ce, K_u gain이 Fuzzy에 어떤 영향을 미치는지 test해보았다. 그래서 주어진 gain들 $K_e = 1.03429$, $K_ce = 0.00002$, $K_u = 4.603$에서 두 변수는 고정시킨 채 한 변수만 바꿔보면서 결과에 어떤 영향을 미치는 지 살펴보

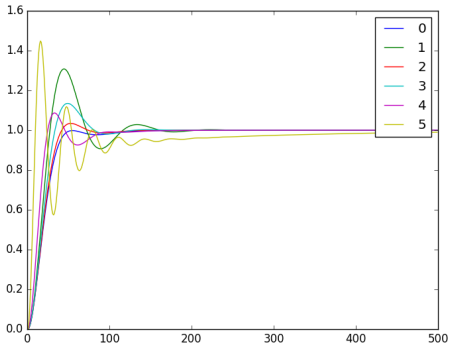


Figure 1: p, i, d gain에 따른 다양한 time response를 확인할 수 있다.

았다. Table 1, Table 2, Table 3는 각각 K_e, K_ce, K_u만 바꿨을 때의 overshoot, rising time, settling time의 변화이다.

Table 1: FLC Results with respect to the change of K_e

#	**Ke**	Kce	Ku	Overshoot(%)	Rising time(s)	Settling time(s)
0	0.03429	0.00002	4.603	-0.98%	269	428
1	0.53429	0.00002	4.603	87.24	30	498
2	1.03429	0.00002	4.603	218.91	22	498
3	1.53429	0.00002	4.603	225.81	22	498
4	2.03429	0.00002	4.603	228.06	22	498
5	2.53429	0.00002	4.603	229.551	22	498

Table 2: FLC Results with respect to the change of K_{ce}

#	Ke	**Kce**	Ku	Overshoot(%)	Rising time(s)	Settling time(s)
1	0.03429	0.00202	4.603	-1.05	272	433
2	0.03429	0.00402	4.603	-1.12	274	439
3	0.03429	0.00602	4.603	-1.18	276	444
4	0.03429	0.00802	4.603	-1.25	279	449
5	0.03429	0.01002	4.603	-1.32	281	454

앞으로 tuning을 하는 데에 있어서 K_e, K_ce, K_u 에 대응하는 overshoot, rising time, settling time을 다음과 같이 정의한다.

$$\phi(K_e, K_{ce}, K_u) = \{overshoot, rising_time, settling_time\} \tag{1}$$

위의 결과를 토대로, K_e를 높히게 되면 rising time이 빨라지지만 overshoot이 급격히 커지고 settling time도 굉장히 길어지는 것을 확인할 수 있었다. 그리고 K_{ce}의 크기를 키우게 되면 직접적으로 어느 조건이 개선되는 게 명

Table 3: FLC Results with respect to the change of K_u

#	Ke	Kce	Ku	Overshoot(%)	Rising time(s)	Settling time(s)
1	0.03429	0.00002	1.403	-32.43	-	-
2	0.03429	0.00002	2.203	-15.75	-	-
3	0.03429	0.00002	3.003	-7.07	439	-
4	0.03429	0.00002	3.803	-2.85	336	498
5	0.03429	0.00002	4.603	-0.98	269	428

확하지는 않지만 K_e와 K_u와 상호작용을 하면서 어느 정도 성능을 보완해준다는 것을 볼 수 있었다. 그리고 K_u 같은 경우에는 크기를 높히니 전반적으로 성능이 모두 개선되었는데, 이를 통해서 K_e, K_ce, K_u이 각각 i, d, p gain의 특성을 지니는 것을 짐작할 수 있었다.

이러한 실증적인 경험을 바탕으로 직접 튜닝을 해본 결과, $\phi(1.03429, 1.00002, 3.103) = \{0.84\%, 35, 82\}$에서 그나마 성능이 괜찮게 나오는 것을 확인할 수 있다.(Fig. 2) 하지만, overshoot이 PID controller로 제어했을 때에 비해서 이상하고, overshoot이 발생한 이후로 undershoot이 발생한 이후 다시 1로 수렴하는 것을 확인할 수 있었다. 그리고 이러한 undershooting이 response가 1로 수렴하는데 settling time을 굉장히 늦추는 것이라 볼 수 있다. 따라서 FLC를 바탕으로 이러한 문제를 해결하기 위해서는 먼저 membership set을 수정하고, FLC의 rule도 조금 수정해보기로 하였다.

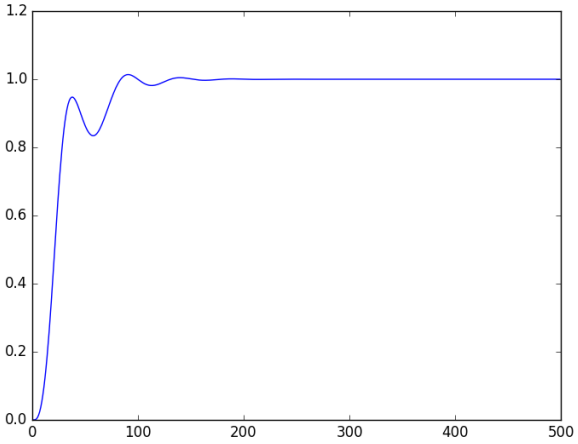


Figure 2: $\phi(1.03429, 1.00002, 3.103)$에 따른 response의 개형. 수렴은 굉장히 잘 되지만 overshoot이 적절치 않아 1에 가까워지는 시점에서 gain을 잘 걸어주어야 하는게 제어에서 가장 중요한 점이라고 판단됨.

3.2 Tuning Revised FLC

3.2.1 Adding additional Rule

우선적으로, 성능을 개선시키기 위해서는 추가적으로 좀 더 state를 묘사하는 rule이 좀 더 *fuzzy*화되어야 한다고 판단했다. 특히 우리의 경우 undershooting이 어느 condition에서 발생하는 지를 확인해야 했다. 0에서 시작하여 error가 음수인 상황이므로, 그 상황에 대한 묘사가 좀 더 필요하다고 판단하였다. 따라서 Fig. 3을 추가하였다. 추가한 rule은 error가 NB일 때 d error가 PM이면 output이 NS가 되게, error가 NB일 때 d error가 PB이면 output이 ZO가 되게, error가 NM일 때 d error가 PS이면 output이 NS가 되게끔 세 식을 추가하였다.

```
rule[19]= Fuzzy::strength(error, Fuzzy::NB, d_error, Fuzzy::PM); out[19] = Fuzzy::NS;
rule[21]= Fuzzy::strength(error, Fuzzy::NB, d_error, Fuzzy::PB); out[21] = Fuzzy::ZO;
rule[20]= Fuzzy::strength(error, Fuzzy::NM, d_error, Fuzzy::PS); out[20] = Fuzzy::NS;
```

Figure 3: Fig. 2의 response를 본 후에 추가된 rule들. output이 0에서 1로 향할 때 좀 더 잘 수렴했으면 하는 상황을 위해 rule을 추가함.

3.2.2 Revising Membership Functions

그 후, overshoot이 global maximum이지 않은 부분을 단순히 rule을 추가해서 수정할 뿐만 아니라 membership set을 수정하여 발전시켜 보기로 하였다. Fig. 2에서 보면 알 수 있듯이, 빠르게 상승하다가 주춤하는 response가 performance를 저하시킨다고 판단했다. 이를 해결하기 위해 membership function들을 좀더 원점에 쏠리게 배치하면 원점으로 수렴할 때 조금 큰 gain이 걸리지 않을까 라는 생각에 membership function의 배치를 Fig. 4(b)와 같이 변경 시킨 후, 다시 $\phi(1.03429, 1.00002, 3.103)$로 response를 측정했더니 Fig. 4(c)와 같이 undershooting이 좀 덜 완화되는 것을 확인할 수 있었다.

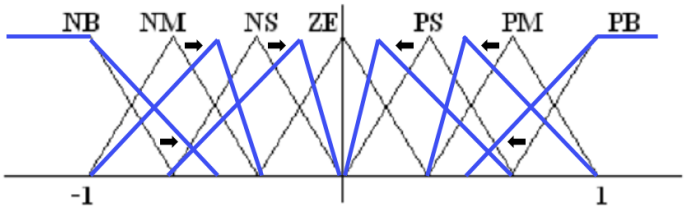
4 Conclusion

Tuning해본 결과, K_e, K_ce, K_u의 변수들은 직관적이진 않지만 각각 PID에서의 i, d , p gain과 비슷한 성향을 띈다는 것을 확인할 수 있었다. K_u의 경우 커질수록 overshoot이 생기기 시작하면서 rising time도 줄어드는 것을 확인할 수 있었다. K_e를 높히게 되면 rising time이 빨라지지만 overshoot이 급격히 커지고 settling time도 굉장히 길어지는 것을 확인할 수 있었다. 그리고 K_{ce}의 크기를 키우게 되면 직접적으로 어느 조건이 개선되는 게 명확하지는 않지만 K_e와 K_u와 상호작용을 하면서 어느 정도 성능을 보완해준다는 것을 볼 수 있었다. 그리고 K_u 같은 경우에는 크기를 높히니 전반적으로 성능이 모두 개선되는 것도 확인할 수 있었다.

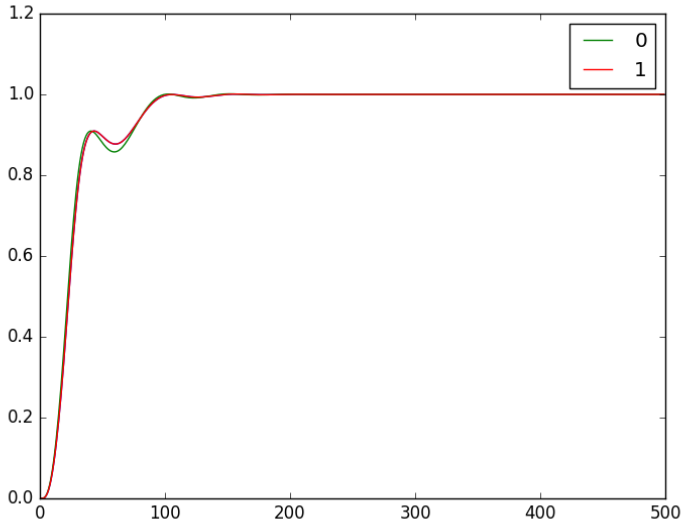
그리고 FLC를 통해 control의 성능을 개선시키는 방법에 대해서도 실험을 통했다. 실험을 통해 구체적으로 알 수는 없지만 membership set들과 fuzzy fuzzy의 rule들을 통해서 제어의 성능을 더 높힐 수 있음을 확인할 수 있었다.

```
std::vector<double> NB = {-INF,   -1,   -1/2.};
std::vector<double> NM = {-1,   -1/2., -1/3.};
std::vector<double> NS = {-2/3., -1/6.,  0};
std::vector<double> ZO = {-1/3.,  0,    1/3.};
std::vector<double> PS = { 0,    1/6.,  2/3.};
std::vector<double> PM = { 1/3.,  1/2.,  1};
std::vector<double> PB = { 1/2.,  1,    INF};
```

(a)



(b)



(c)

Figure 4: (a),(b): 기존의 membership set을 좀 더 원점 근처로 밀집시켰다. 그래서 1에 가까워 질 경우에 좀 더 강한 gain을 걸 수 있게 수정할 수 있을 것이라고 가정하였었다. (c) 기존의 주어진 코드와 수정된 Fuzzy code의의 graph를 비교해보았다. undershooting이 좀 더 완화되는 것을 확인할 수 있었다.