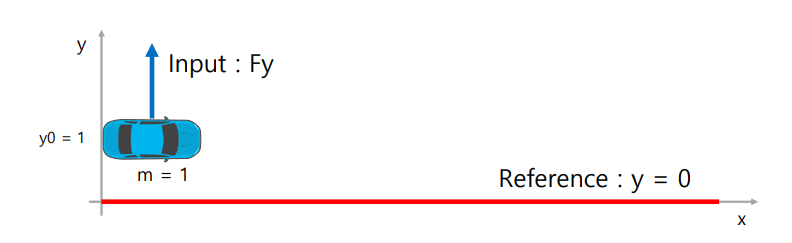
**PID Control Examples**

**Planning & Control**

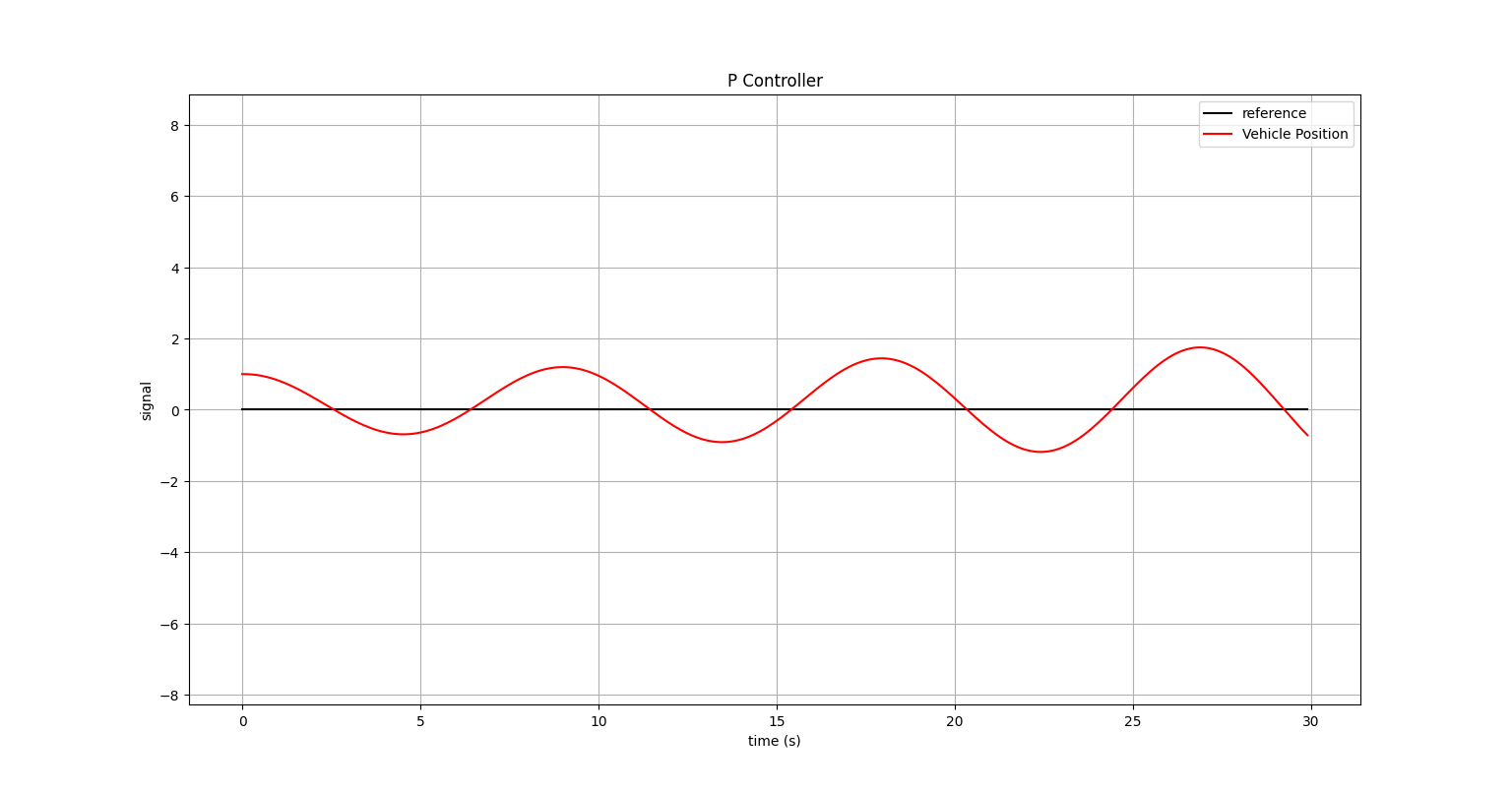
**[1기] 조환영**

**1. P controller**

- 비례제어기는 reference 값과 측정값의 오차에 단순히 비례상수를 곱하여 제어를 실행한다.



해당 시스템에 P 제어기를 사용한 결과는 다음과 같다.



뒤로 갈수록 오차가 더 많이 발생하는 걸 볼 수 있는데 이는 아마 force ratio와 bias의 영향 때문인 것으로 생각된다.

- Plant input u(t)

u = (reference - measure) \* kp

**2. PD controller**

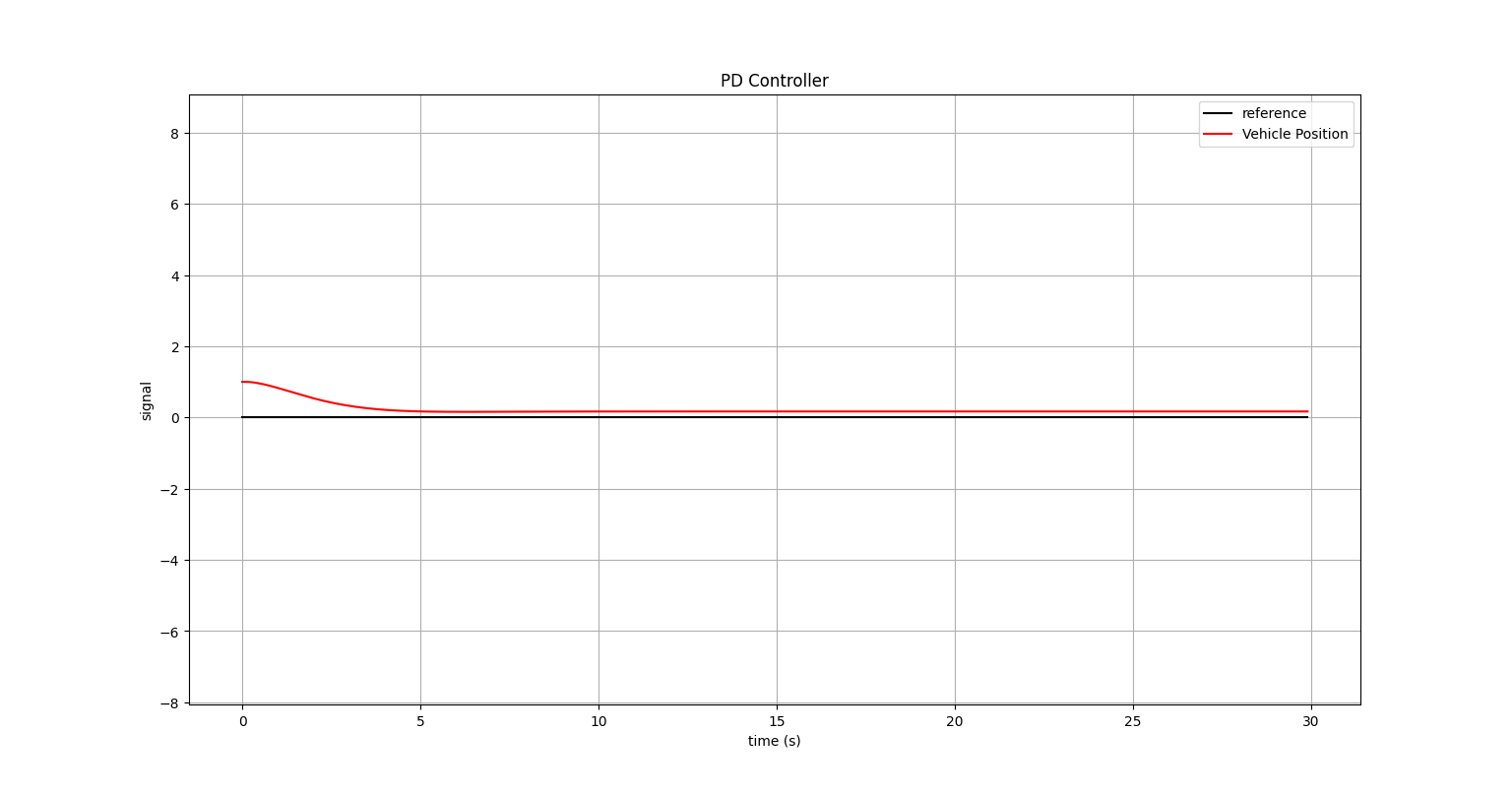
- 비례-미분 제어기는 P 제어기에 오차의 속도까지 고려하는 제어기이다.

- 즉, 오차의 변화율이 작을수록(ref에 가까워질 수록) 기존 P제어기의 시스템 input보다 작은 값들을 plant에 입력시켜주게 되어 overshoot을 방지할 수 있다.

텍스트, 스크린샷, 폰트, 라인이(가) 표시된 사진

자동 생성된 설명

위와 동일할 시스템에 PD 제어기를 사용한 결과는 아래와 같다.



PD 제어기로는 overshoot이 잡히지 않을 것이라 예상했지만, 예상과 반대로 overshoot은 나타나지 않고 steady-state 에러만 나타나면서 reference에 빠른 속도로 수렴하는 결과를 얻을 수 있었다.

- Plant Input u(t)

error = reference - measure

u = error \* kp + (error - prev\_error) \* kd / step\_time

prev\_error = error

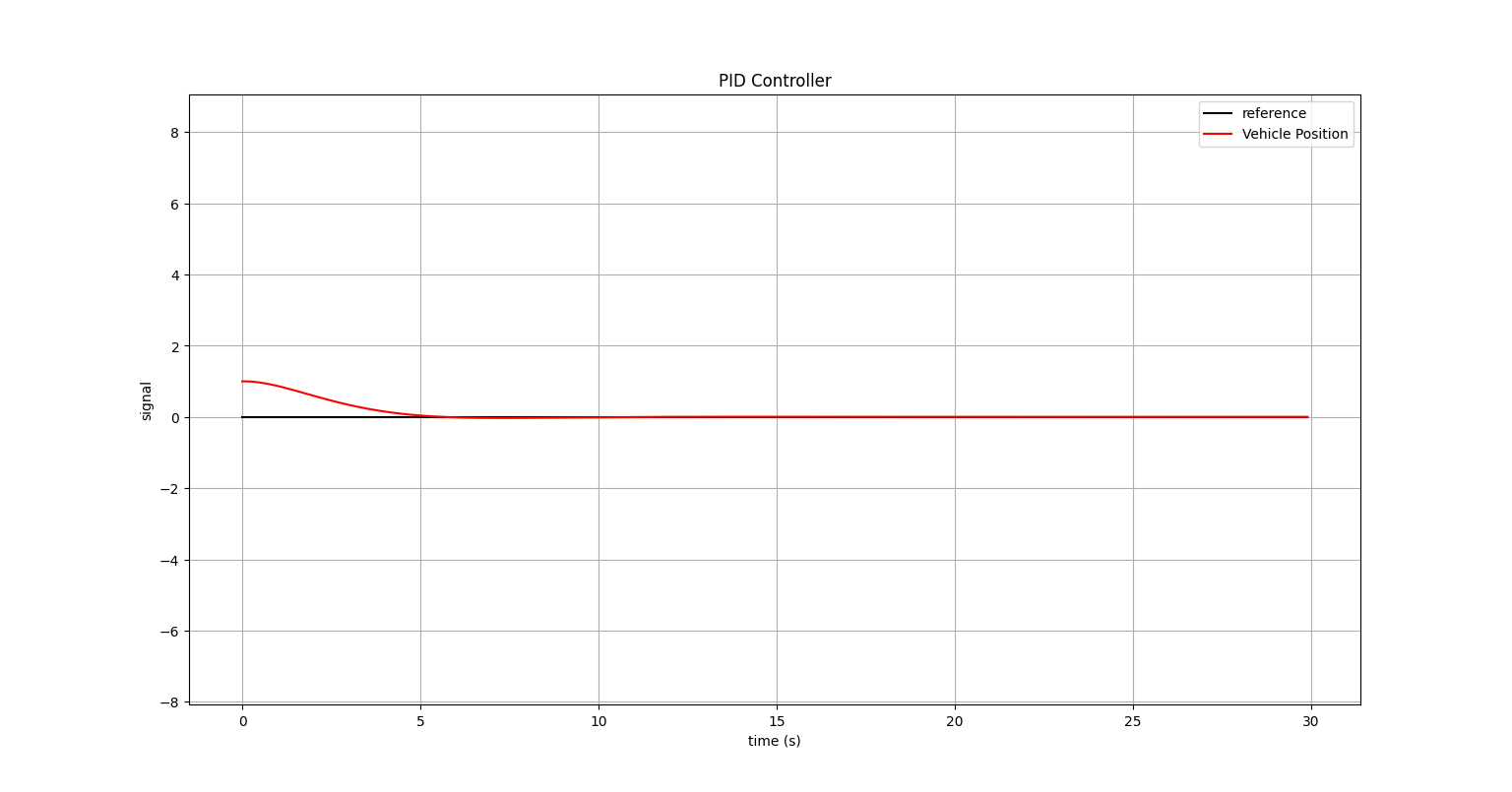
**3. PID controller**

- I term을 추가해 steady state error를 상쇄시켜 보자.

텍스트, 스크린샷, 폰트, 라인이(가) 표시된 사진

자동 생성된 설명

위와 동일할 시스템에 PID 제어기를 사용한 결과는 아래와 같다.



이전에 PD 제어기만을 사용했을 때 발생했던 정상상태 오차가 거의 발생하지 않는다.

- Plant Input u(t)

error = reference - measure

p\_term = kp \* error

d\_term = kd \* (error - prev\_error) / step\_time

i\_term += ki \* error \* step\_time

u = p\_term + d\_term + i\_term

prev\_error = error

**4. Comparing PID controller**

- 각 제어기의 특성과 장, 단점, 그리고 어떤 상황에 사용하면 좋을지에 대한 간략한 의견

**1) P 제어기**

- 원하는 값과 실제 측정 값 사이의 오차에 비례하여 제어 입력을 생성한다.

- 구현이 매우 쉬워 간단한 프로세스에 적용시킬 때는 좋을 것 같다.

- 하지만 정확한 제어가 필요한 경우, 오차가 클수록 반응이 큰 제어기의 특성 상 이용이 어려울 수 있을 것 같다.

- 다시 말해, 오차가 0으로 수렴하려고 하지 않고 근사값에 가까워지면서 진동이 발생한다.

- 예를 들어 inverted pendulum 같은 미세한 제어가 필요한 시스템에서 사용이 어려울 것 같다.

**2) PD 제어기**

- 오차의 변화율에 비례한 항이 추가되어 변하는 정도(속도)에 맞춰 더 미세한 제어가 가능하다.

- P 제어기에서 발생하던 overshoot를 감소시킬 수 있어 plant의 에너지 손실을 줄일 수 있다.

- 안정성이 보장되는 반면, 고정 오차에 대한 보상이 이루어 지지 않아 이상적이지 않은 현실 세계에서의 시스템을 제어할 경우 steady state error가 발생할 확률이 크다.

- 실제 산업현장에서는 PD 제어만으로 대부분의 기계 시스템을 제어할 수 있다고 배웠다.

- 그 이유는 PID를 사용했을 때 누적되는 오차로 인해 제어 시스템의 안정성에 영향을 미칠 수 있어 가끔 사용이 제한될 수 있다고 한다. 또한 많은 상황에서 I 요소를 제거하더라도 충분한 성능을 얻을 수 있는 경우가 많아 PD 제어기를 사용하는 것이 더 효과적일 수 있다.

- 대표적인 예로 자동차 서스펜션 시스템 설계, 로봇 팔 제어, 드론 자세제어 등의 분야에서 PD 제어기가 사용되는 것으로 알고 있다.

**3) PID 제어기**

- 오차 누적을 고려하여 정상상태 오차가 클 경우 출력이 desired value에 더 가까워질 수 있다.

- P, I, D 모든 요소가 포함된 만큼 파라미터를 튜닝하는 것이 어려울 수 있다.

- 정상상태 오차를 0에 수렴하도록 설계하여 정확한 제어가 필요한 경우에 사용될 것 같다.

- 사실 P, I, D의 모든 요소를 사용하지 않아도, gain 값들을 적절히 설정하면 overshoot을 감소시키고 stead state error를 어느정도 줄일 수 있다. 하지만 모든 시스템에 적용되지 않기 때문에 결론적으로, PD 제어기에서 D gain을 잘 선택해 overshooting과 진동을 최소화하면서, PI를 함께 사용해 정상상태 오차를 최소화하는 것이 이상적일 것 같다.

**5. Tuning PID Controller**

- 동일한 상황에서 어떤 외부 환경에 의해 시스템이 조금 바뀌었다. 적절한 제어기를 설계하고 튜닝하여 만족스러운 제어 성능을 얻어보자.

- 기존 시스템(vehicle model) 🡺 노이즈 표준편차 R = 0, Force Ratio = 0.99, Force Bias = 0.05

- 바뀐 시스템(vehicle model) 🡺 노이즈 표준편차 R = 0, Force Ratio = 0.4, Force Bias = -0,1

텍스트, 스크린샷, 폰트, 라인이(가) 표시된 사진

자동 생성된 설명

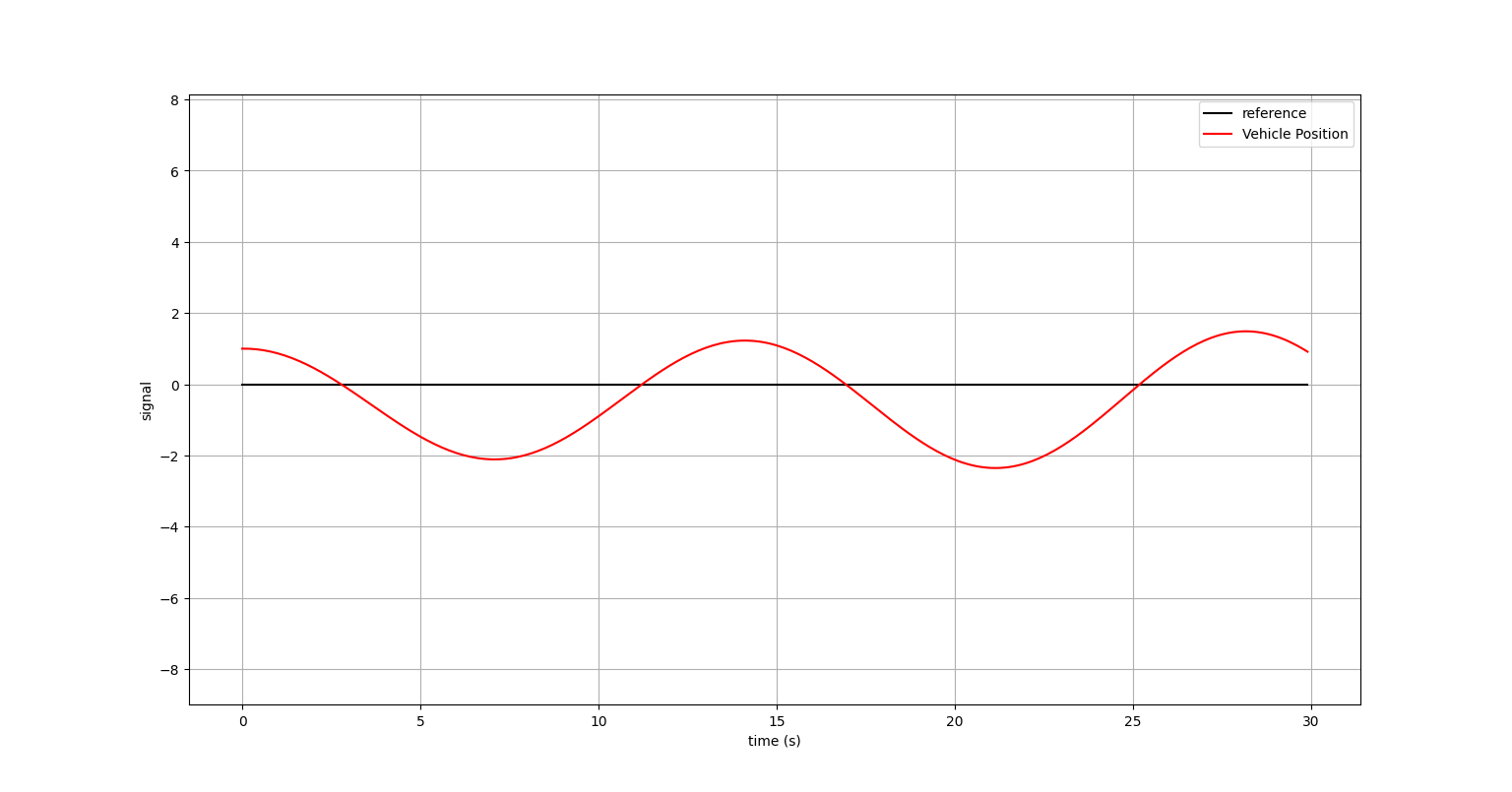
- 제어기 튜닝은 다음과 같은 절차를 거쳤다. (gain 값 gradual 하게 높이면서 튜닝)

**1) P 제어로 어느정도 응답 특성을 맞춘다.**

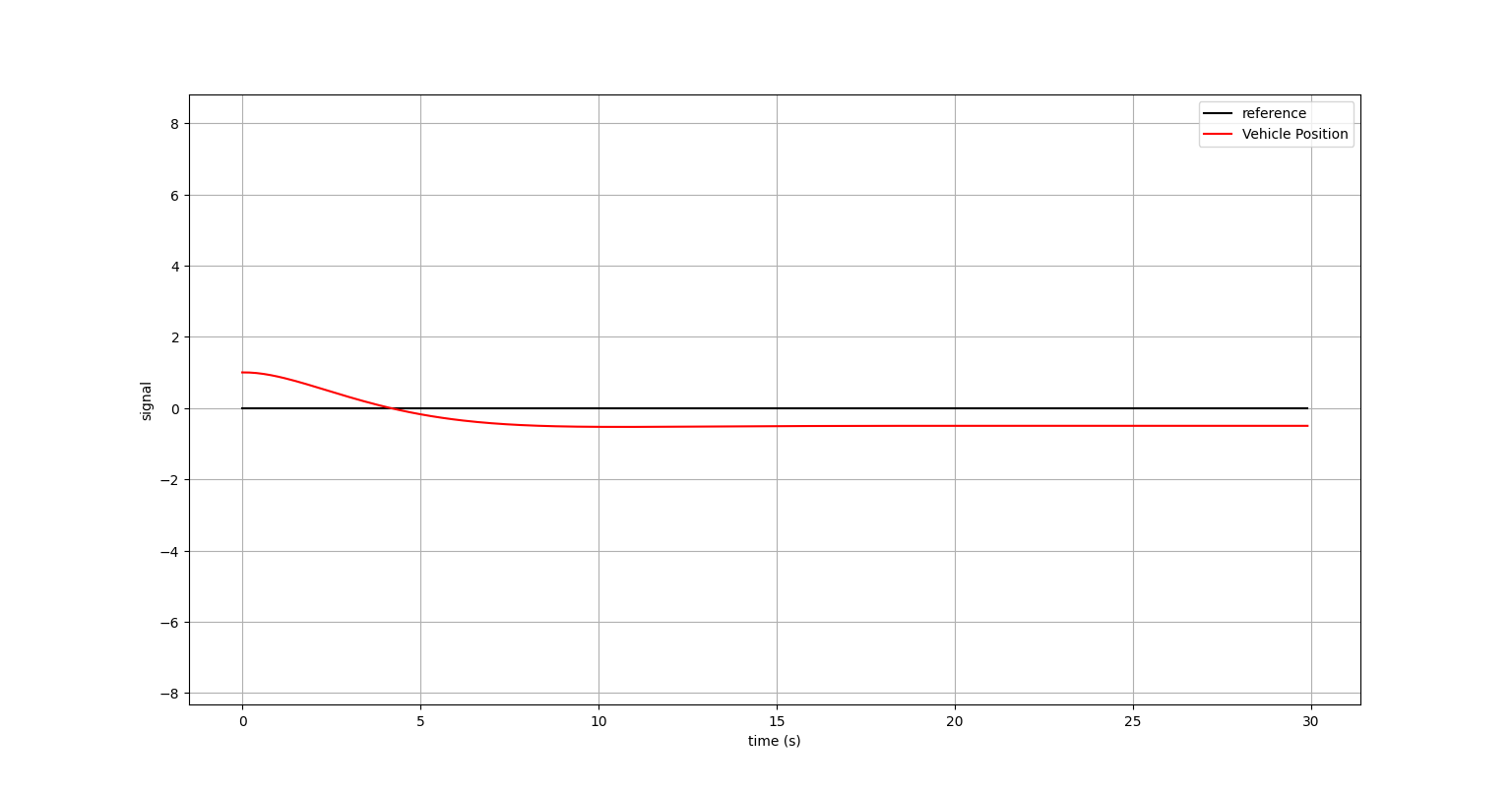
**2) 이후 D 제어로 overshoot을 줄이고 rise time, settling time 등의 성능을 개선한다.**

**3) I gain을 추가하여 steady state error를 보정한다.**

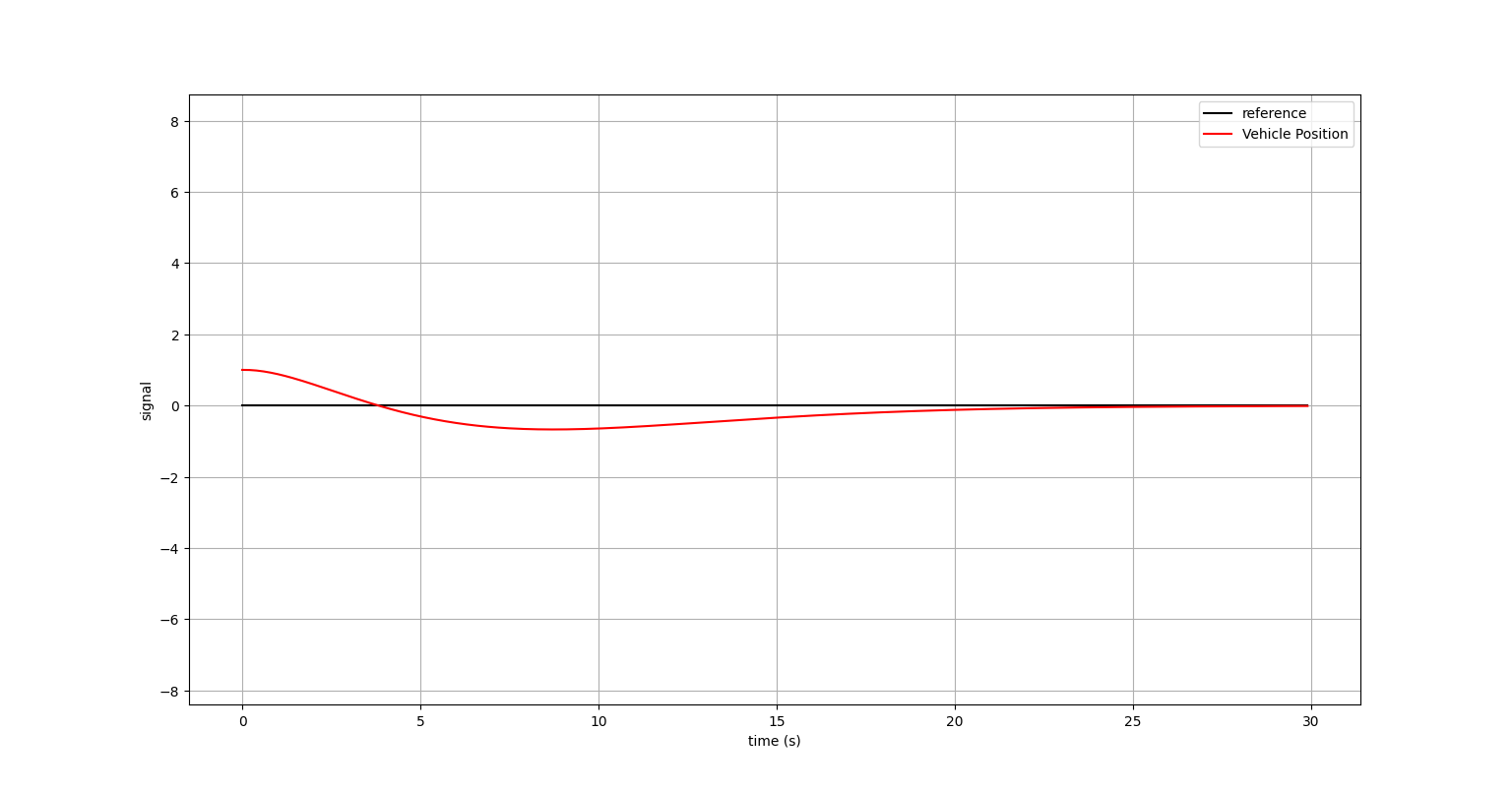
- **P gain**: 0.4 ~ 0.6 정도의 값이 적절했다. 🡪 너무 크면 진동 발산, 너무 작으면 reference 로 복귀하는데 오래 걸림 🡪 **kp = 0.5로 선정**

****

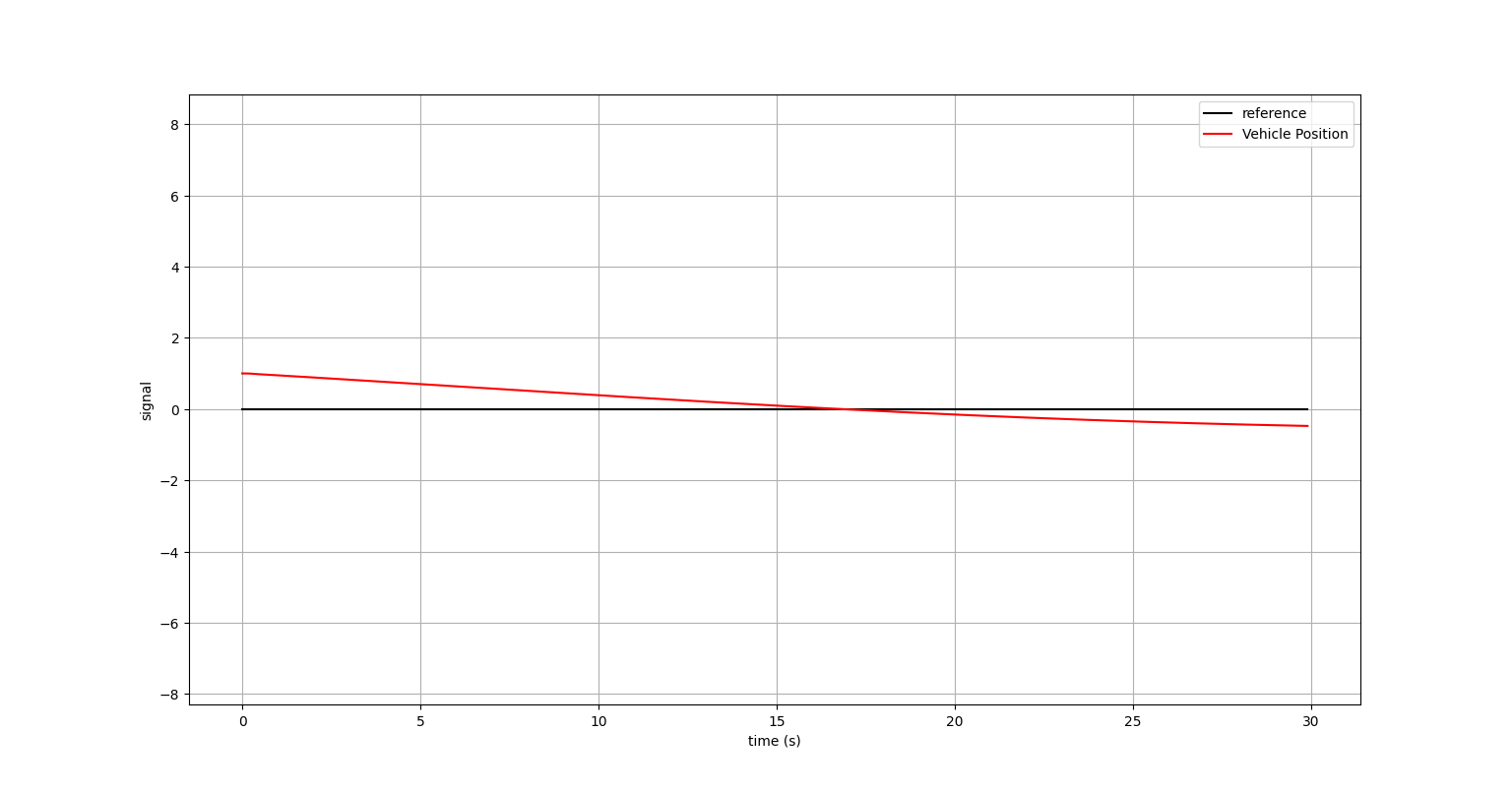
- **D gain**: overshooting 이 일어나지 않는 선에서 가장 작은 값은 택해야 안정적이면서 빠르게 목표값에 도달할 수 있다. 🡪 1.7 이후부터 overshoot이 발생하지 않는다. 🡪 값이 커질수록 rise time이 커져 빠른 제어에 불리하다 (이건 자동차 시스템이기 때문에 빠른 제어가 좋을 지 느리지만 승차감이 좋은 제어가 좋을지… 더 확실한 요구사항에 맞춰 설계할 필요가 있지만 여기서는 그냥 빠른 제어를 목표로 설계하겠다.) 🡪 **kd = 1.7로 선정**



- **I gain**: I gain은 너무 큰 값을 넣어주면 오차 누적으로 인해 진동 발산해버린다. 🡪 또 너무 작은 값을 선정하면 정상상태 오차를 줄이기 힘들다. 🡪 **ki = 0.05로 선정**

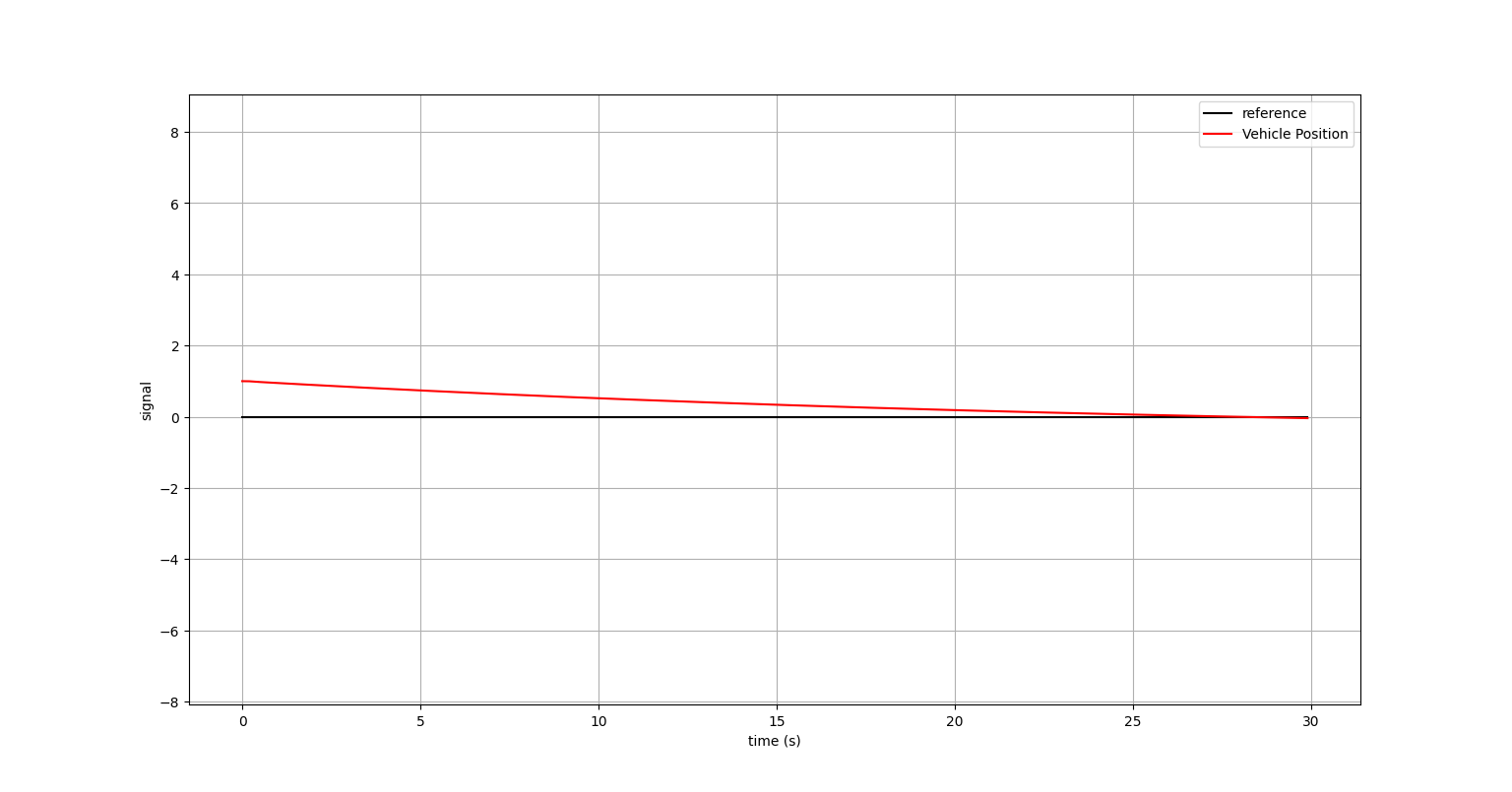


- I gain을 맞춰주고 나니 기존 정상상태 오차 때문에 overshoot이라고 생각하지 않았던 부분이 overshoot으로 나타났다. 🡪 이를 제거하기 위해 D gain값을 다시 튜닝해줬다. 🡪 **kd = 13** 선정



- overshoot을 제거하는 것에 초점을 둬서 d gain 값이 비교적 많이 커져 여기에 맞춰 다시 **I gain을 0.001**로 맞춰주었다.

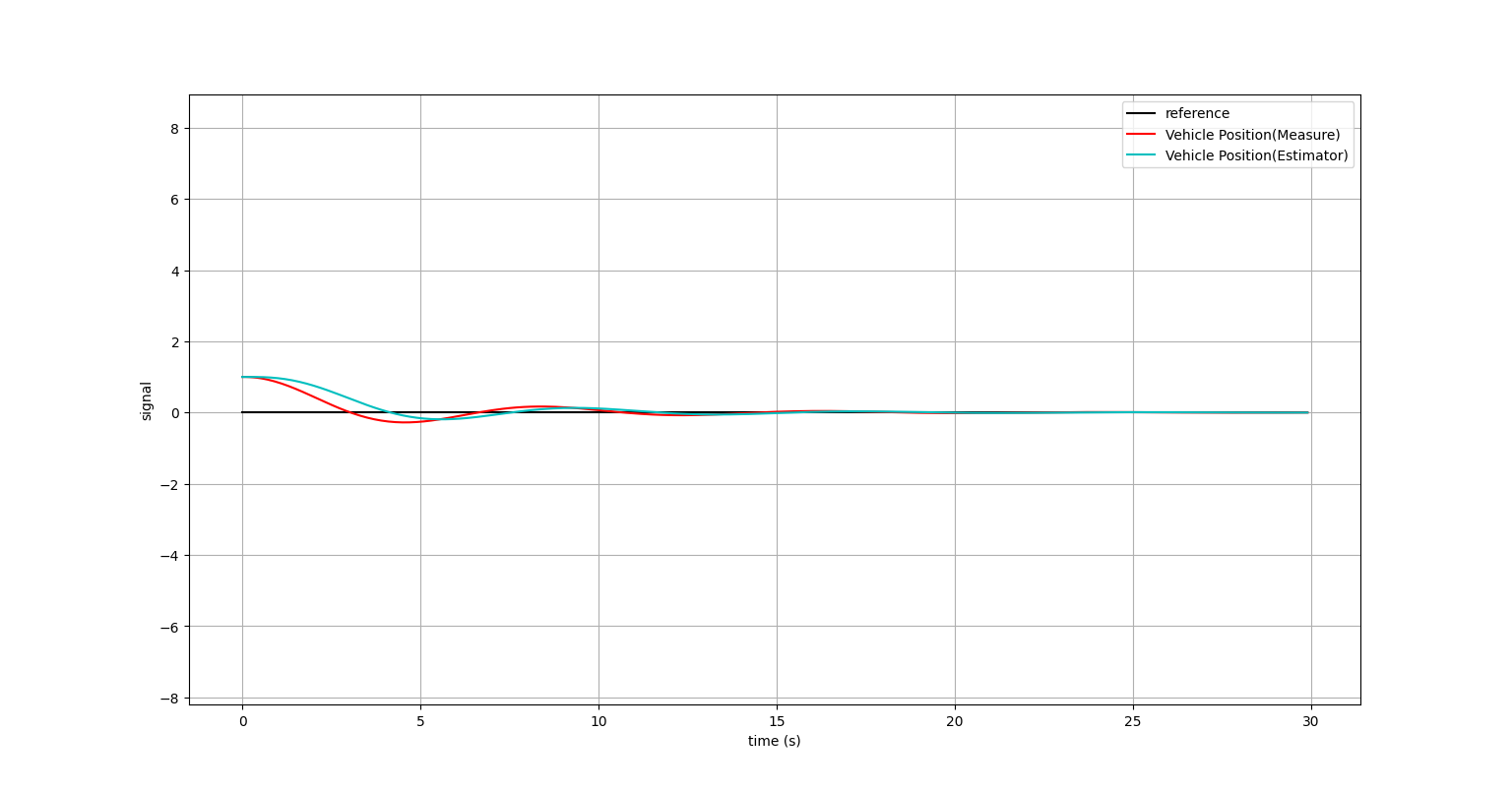
- 최종 결과는 다음과 같다.



**6. PID Controller with State Estimator: Low Pass Filter**

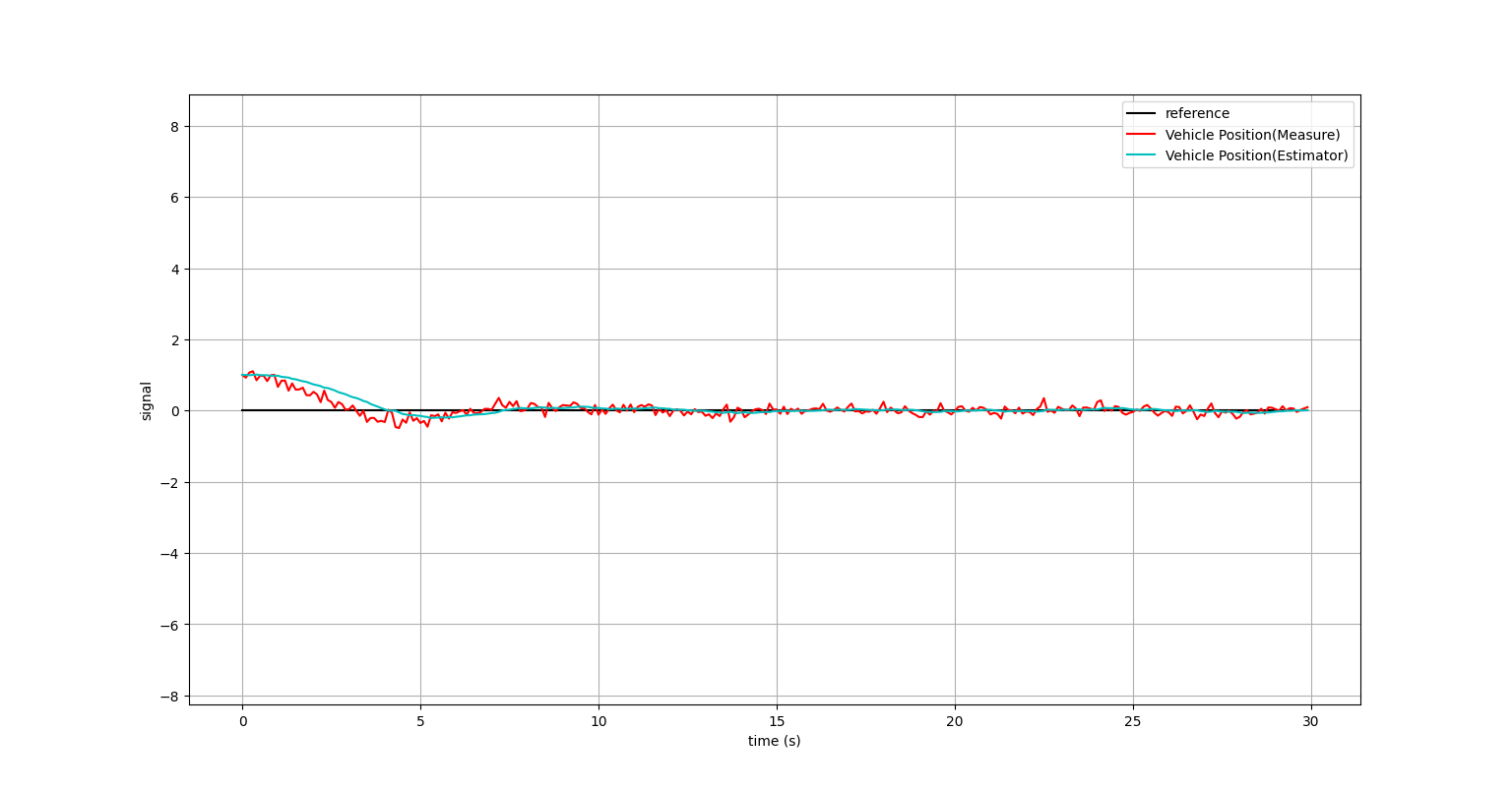
- 자동차의 위치를 알기 위해 state estimator로 Low Pass Filter를 사용해 기존 결과와 비교해 보자.

- 다음은 노이즈가 없을 때(R=0)일 때의 결과이다.

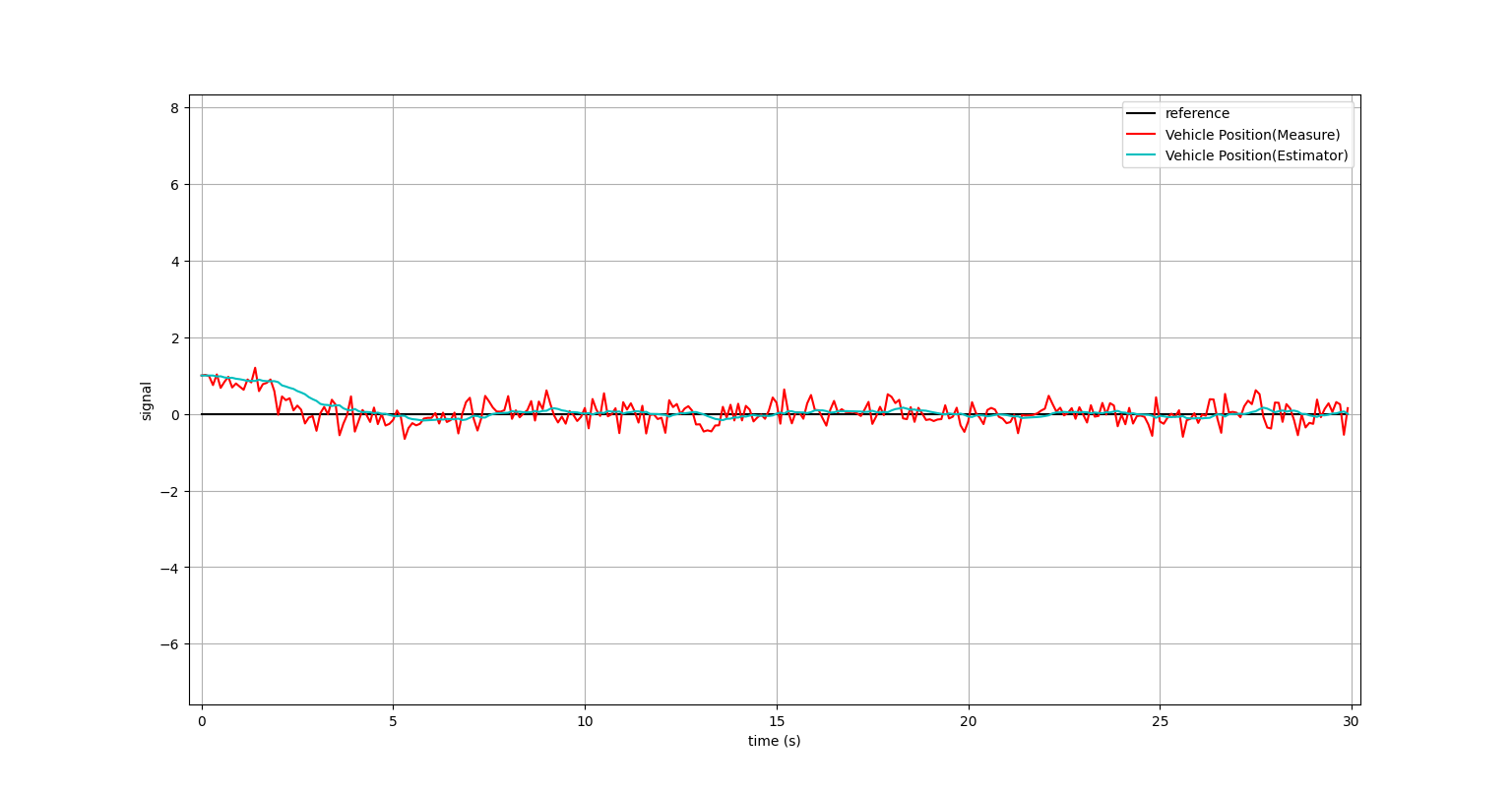


- 차량 모델링으로 인한 약간의 위치 오차가 있지만 전체적인 양상이 비슷한 걸로 봐서 상태 추적기가 잘 작동하고 있음을 알 수 있다.

- 이제 measurement에 노이즈가 있는 실제 상황을 가정하여 노이즈의 표준편차를 R=0.1로 두고 결과를 보았다.



- 노이즈를 조금 더 강하게 줘보자. (R=0.25)



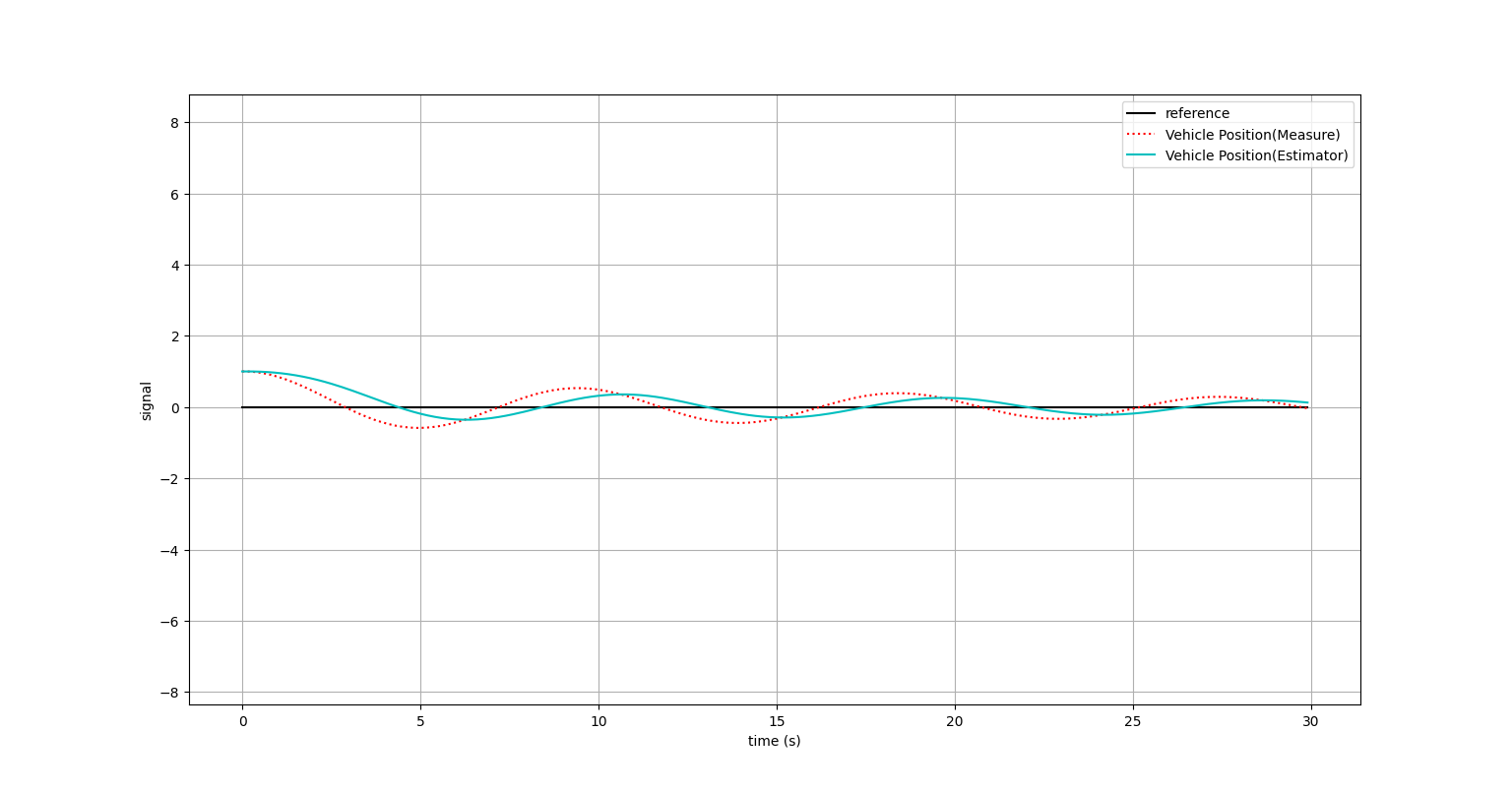
- 성능 좋은 저주파 필터라도 강한 노이즈에는 아직 취약한 모습을 보인다.

- 하지만 위 실습을 통해, 노이즈 때문에 실제 데이터를 측정하는데 어려움을 겪을 수 있는 상황에서 low pass filter를 적용시킨 state estimator가 시스템 상태 파악에 확실히 도움을 준다는 것을 눈으로 확인할 수 있었다.

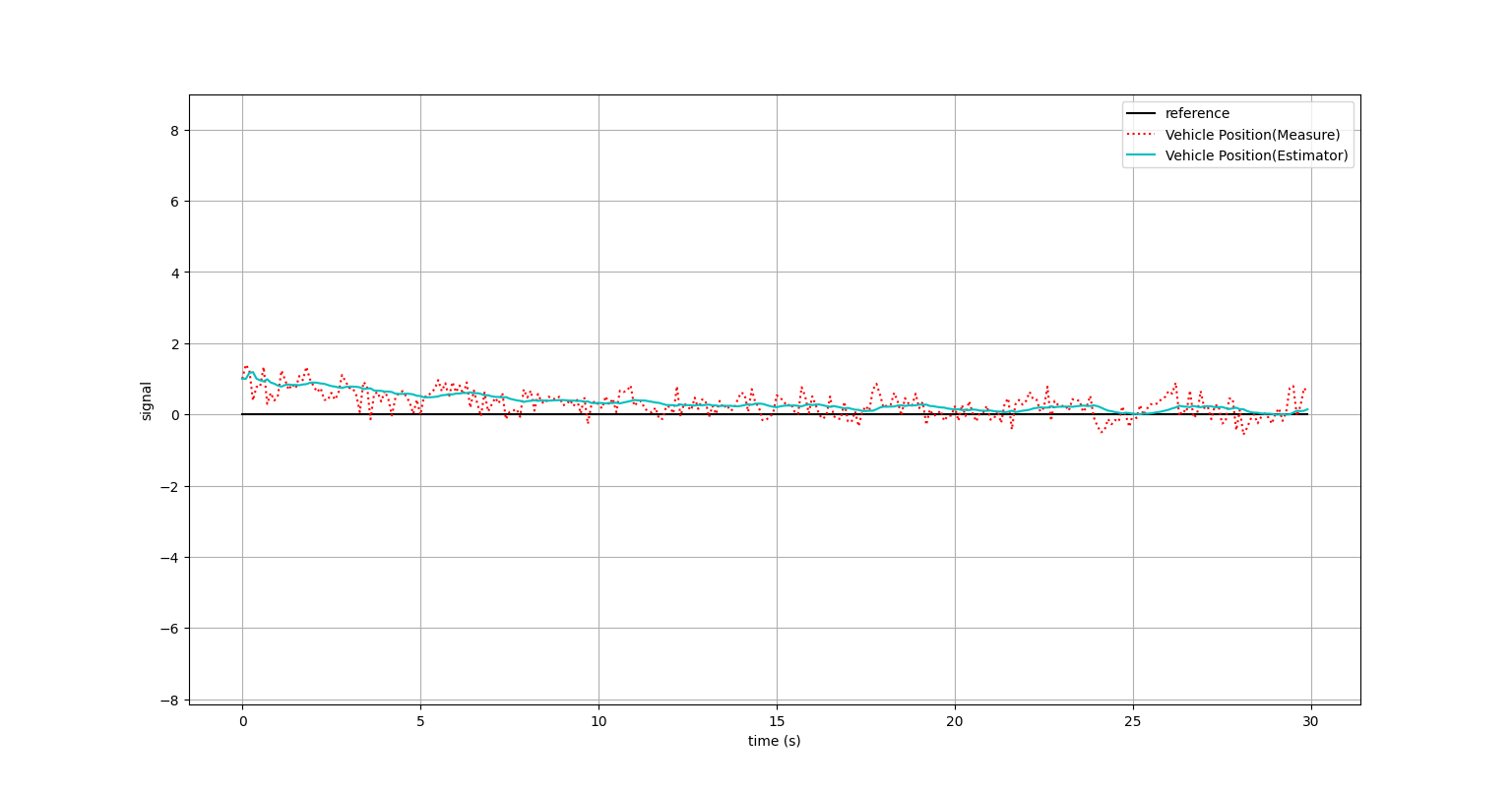
**7. PID Controller with State Estimator: Kalman Filter**

- 이번엔 state estimator로 Kalman Filter를 사용해보자.

- 먼저 노이즈가 없는 경우에 상태 추정기가 measure값을 잘 따라가는지 살펴보자.



- 노이즈 표준편차는 제공된 R=0.25로 사용했고 최적의 결과를 얻기 위해서, P, I, D gain을 튜닝했다.



- 노이즈가 있는 상황에서, kp=0.5, kd=4.0으로 튜닝했고 I 제어기는 사용하지 않았다.

- I 제어기를 사용하는 순간 오차 누적으로 인한 oscillation이 발생해 불안정한 결과가 발생했다.