**Vehicle Control Examples**

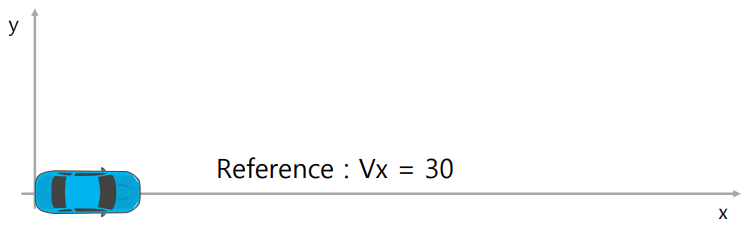
**Planning & Control**

**[1기] 조환영**

**1. Speed control**

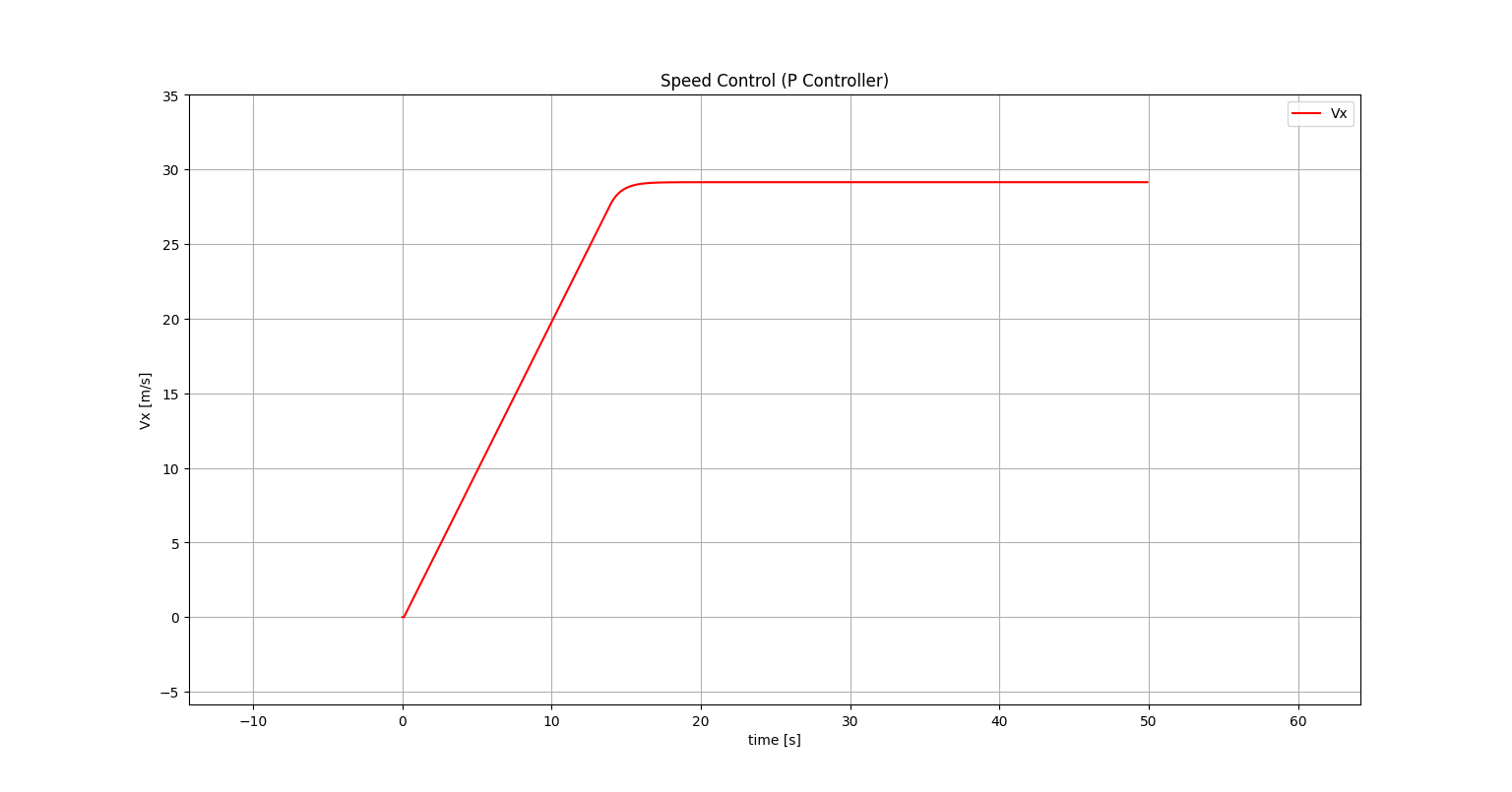
- 앞 차량과의 거리가 충분히 멀 때 작동하는 기능으로, reference velocity를 추종하면 된다.

- 해당 제어는 정상상태에서의 속도가 desired 값과 차이가 나지 않도록 설계하는 것이 중요하기 때문에, 정상상태 오차를 줄일 수 있는 **PI 제어기**를 사용하는 것이 적절하다.

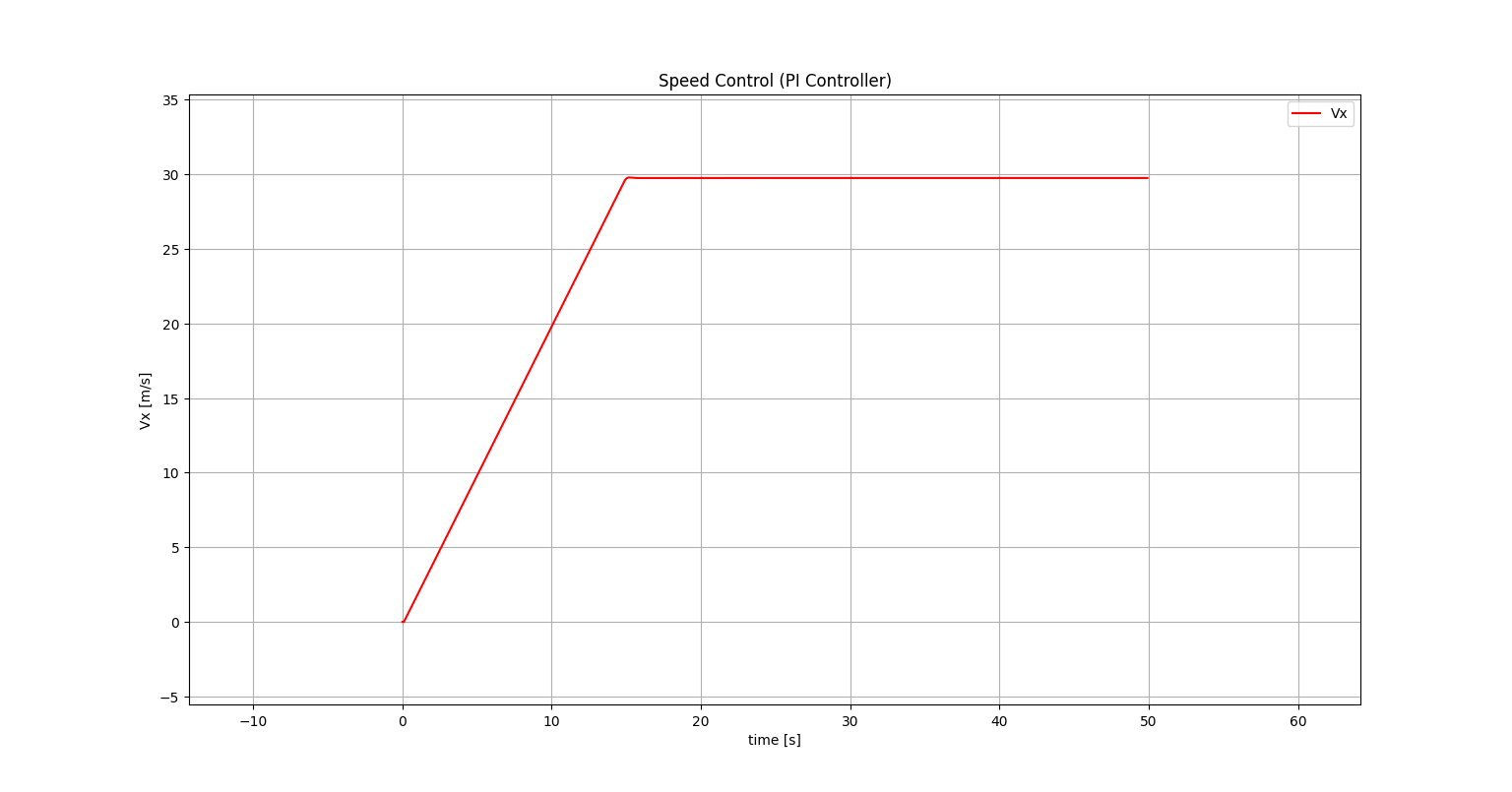


- 참고로 속도의 단위는 mps이다. **(30mps = 108kph)**

- 아래는 P 제어기(kp=1)를 사용했을 때의 결과이다.



- 아래는 PI 제어기(kp=1, ki=25)를 사용했을 때의 결과이다.

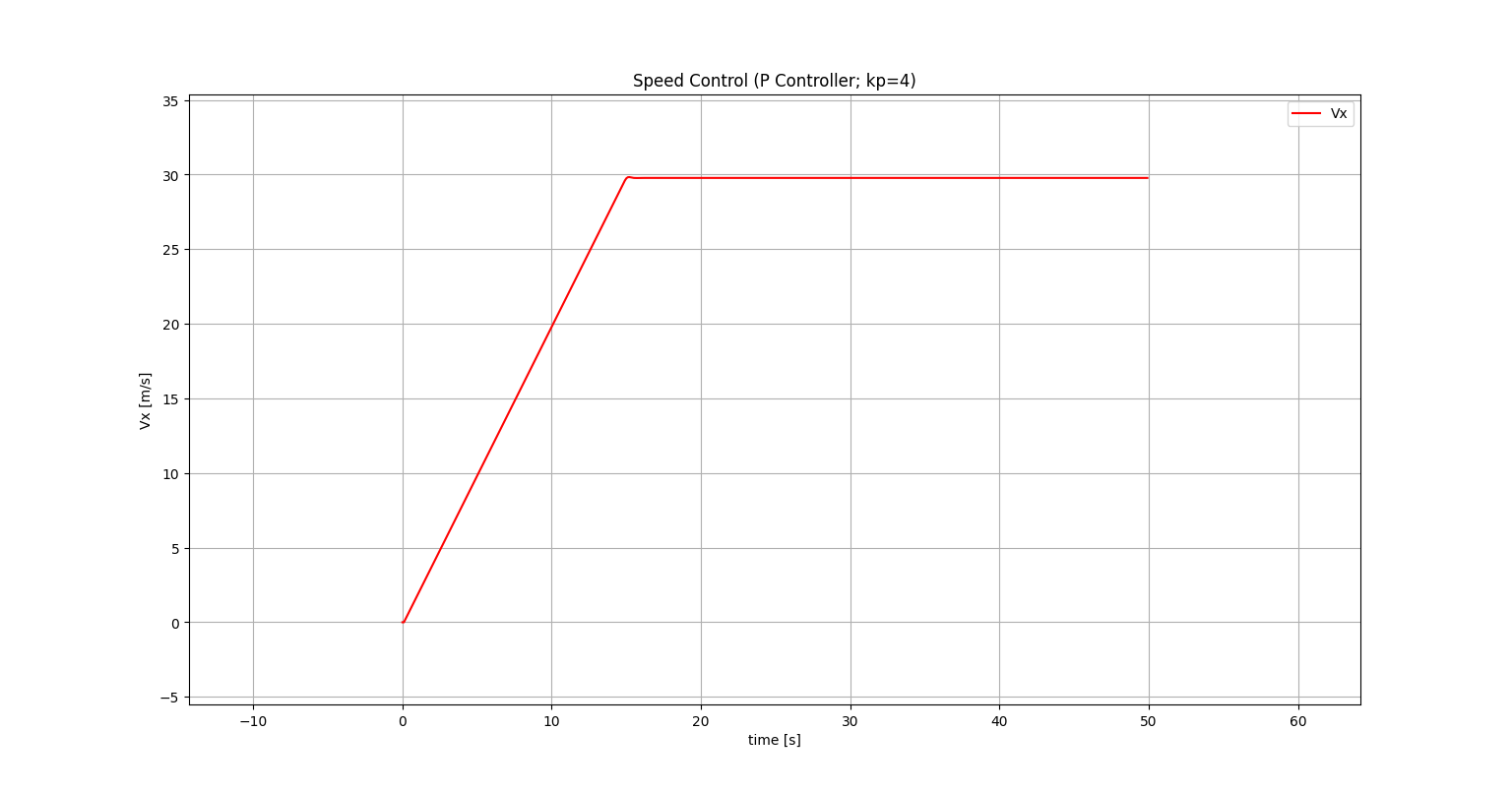


- 정상상태 오차가 개선되었음을 확인할 수 있다.

- Gain 값을 튜닝할 때, kp>kd>ki의 gain값이 부여돼야 안정적인 결과를 볼 수 있다고 해서, kp를 넘이 않는 ki값을 사용하려 했으나, 그랬을 경우 결과의 변화가 보이지 않아 ki값을 무리해서 키웠다. gain값의 편차가 (kp=1, ki=25) 이런 식으로 크게 발생해도 문제가 되지 않는지 궁금하다.

- 또 흥미로운 결과가 있다. 본 시스템에 한해 그런지 몰라도 I gain 없이 P gain만 크게 조절해 줘도 위와 같은 결과를 나타낸다.

- 아래는 P 제어기(kp=4)를 사용했을 때의 결과이다.



- 이런 현상이 발생하는 이유는 모르겠다… 하지만 경우에 따라서는 많은 제어기를 사용하는 것보단, 적절한 튜닝이 더 효율적일 수도 있다는 생각을 했다.

**2. Constant Spacing control**

- 차속과 상관없이 앞차와의 적정 거리(안전거리?)를 상수로 지정해 이를 유지하기 위한 시스템이다.

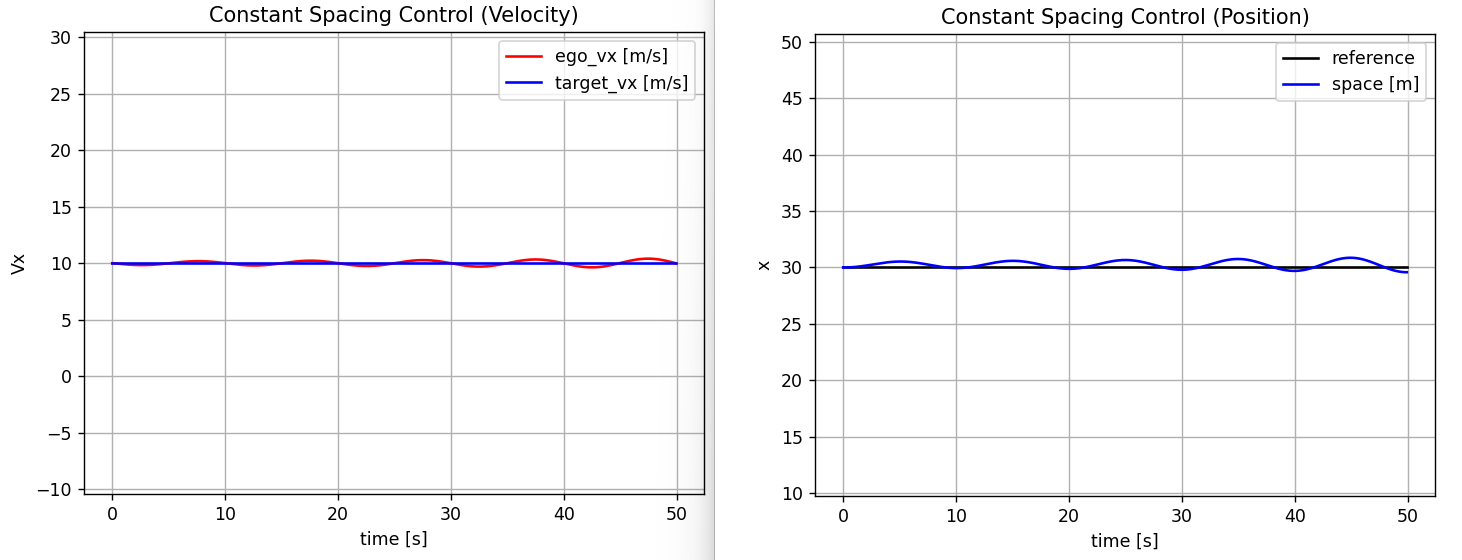
- 실제 도로에서는 더 유연한 상황에서의 거리 유지를 위해 적용되기 힘들 것이라고 생각된다.

- 해당 제어는 목표값으로 수렴하는 과정에서 급격한 속도 및 가속도의 변화가 최소가 되도록 즉, 수렴까지 속도의 진동이 생기지 않도록 제어하는 것이 중요하기 때문에 PD controller를 사용하는 것이 바람직하다.

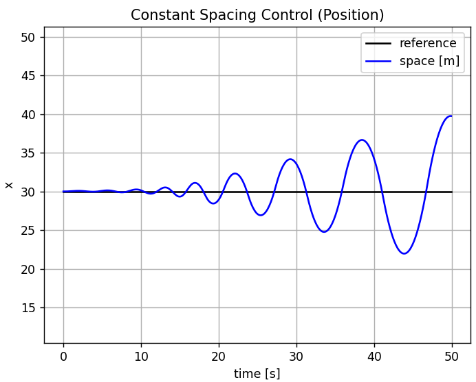
스크린샷, 도표, 라인, 텍스트이(가) 표시된 사진

자동 생성된 설명

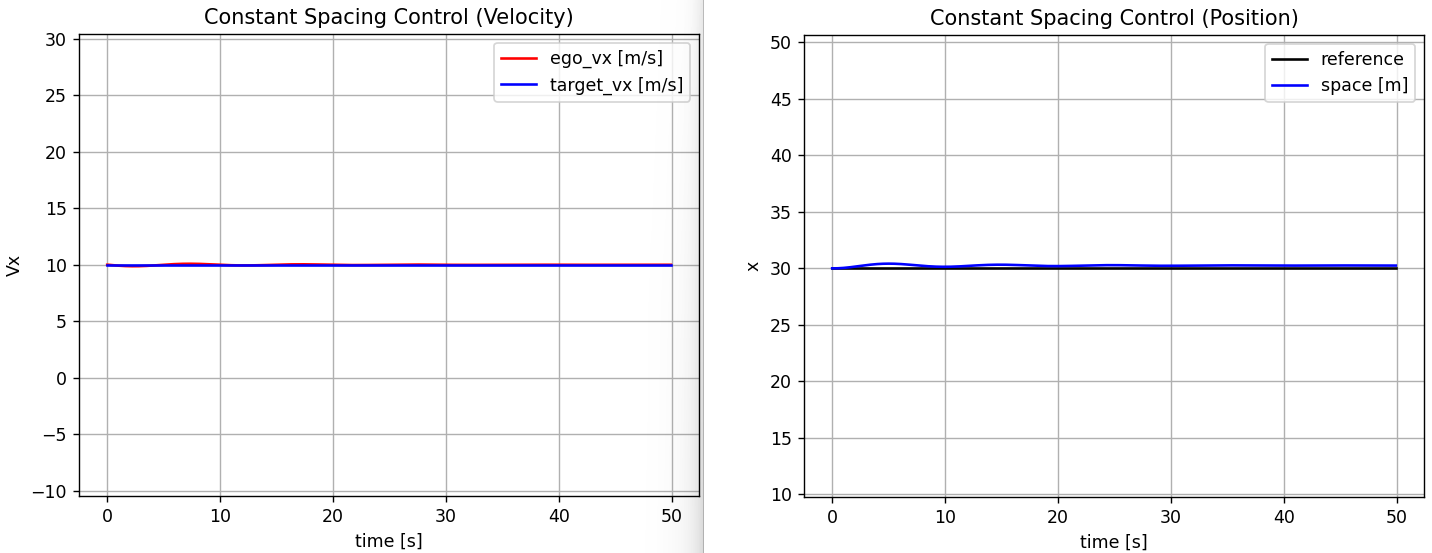
- 아래는 P 제어기(kp=0.4)만을 사용했을 때의 결과이다.



- P gain값은 키우면 키울수록 진동 발산하는 경향을 보였다. 아래는 kp=3,0일 때이다.



- 아래는 overshoot을 개선하기 위해 D 제어기(kd=0.2)를 추가했을 때의 결과이다.



- PD 제어기만을 사용했음에도 불구하고 정상상태 오차도 거의 발생하지 않을 정도로 성능이 개선된 것을 볼 수 있었다.

- 10mps = 36kph의 속도에서 30m의 거리를 유지하는 것은 적절할 수 있으나, 속도가 빨라졌을 경우, 예를 들어 시속 100km의 차량이 30m의 거리를 유지하는 것은 운전자가 심리적으로 불안함을 느낄 수 있는 거리일 수도 있다.

- 반대로 고속에서 비교적 긴 차간거리를 유지하던 차량이 저속에서 불필요하게 공간을 낭비하는 현상이 발생할 수도 있다.

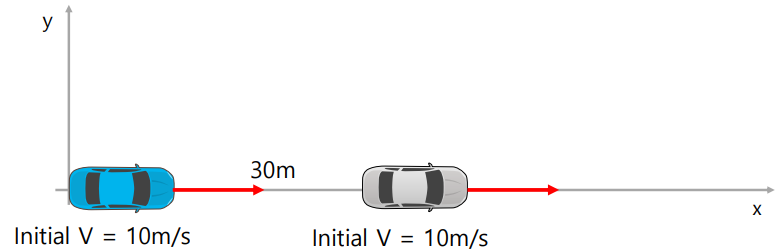
- 따라서 순항제어 시 단순히 정해진 차간 거리를 유지하는 것 보단, 속도에 맞게 차간거리를 유지하는 constant time-gap policy 방법을 사용하는 것이 좋다.

**3. Constant Time-gap Policy (CTP)**

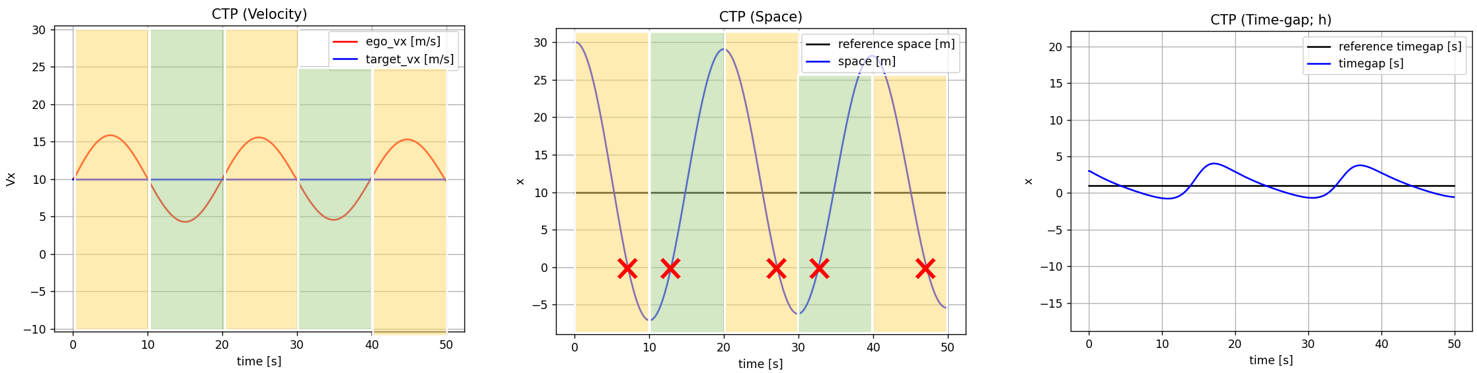
- 앞차와의 거리가 적정 기준 내로 가까워졌을 때, 차속에 따른 최적의 거리를 유지하는 것을 목적으로 하는 시스템이다.

- 실제 ACC에 적용되는 방법으로 속도에 따른 차간거리 유지가 가능한 실용적인 모델이다.

- 해당 제어도 spacing control과 마찬가지의 이유로 PD controller를 사용하는 것이 바람직하다.

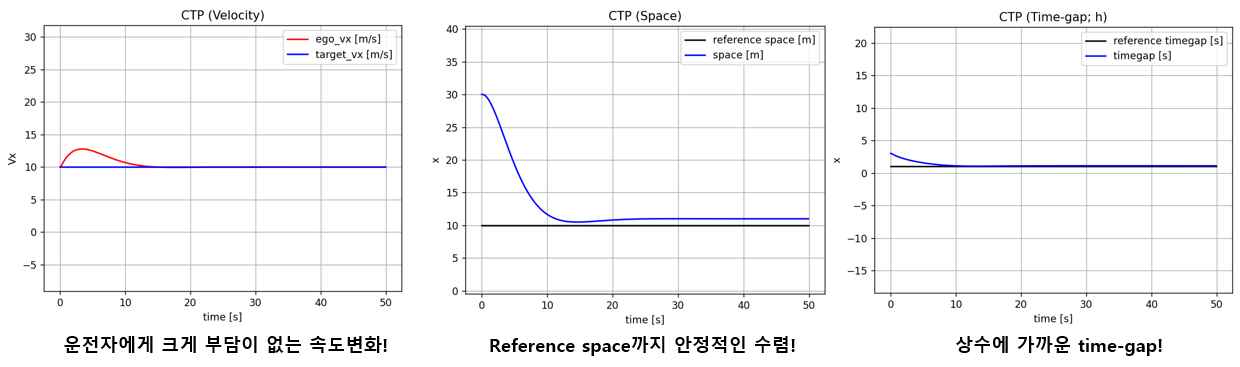


- 아래는 P제어기(kp=0.1)만을 사용했을 때이다.



- 속도가 reference 보다 큰 경우(노란색), 차간거리가 줄어들다가 결국 충돌하는 부분이 생기고, 반대로 속도가 reference 보다 작은 경우(초록색) 차간거리가 늘어나는 결과를 볼 수 있다.

- 충돌과 overshoot을 방지하고 time-gap h를 일정한 상수에 가깝게 개선해 주기 위해 D 제어기를 추가하겠다. 아래는 PD (kd=0.45) 제어기를 사용한 결과이다.



**4. Longitudinal Control: Control Policy (CSP vs CTP)**

스크린샷, 도표, 라인, 텍스트이(가) 표시된 사진

자동 생성된 설명

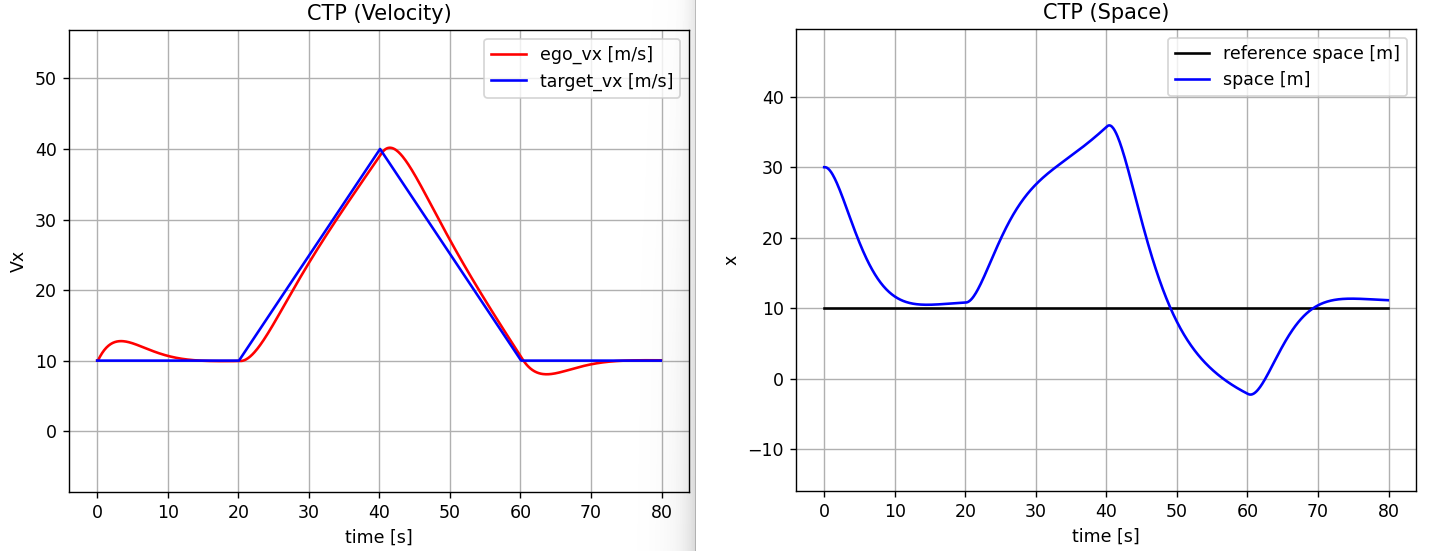
- 위의 2, 3번과 동일한 상황에서 감/가속을 한다는 상황 하에 에러의 측면에서 각 정책을 분석해보자. (간단히 CSP (Constant Spacing Policy)/CTP (Constant Time-gap Policy)라 명명한다.)

- 총 80초의 주행 중 20초 간격으로 “등속🡪가속🡪감속🡪등속”의 상황이 부여되었다.

- 먼저 기존 튜닝 그대로 둘의 차이를 비교해 보면, (CSP: kp=0.4, kd=0.2) (CTP: kp=0.1, kd=0.45)

**텍스트, 라인, 도표, 그래프이(가) 표시된 사진

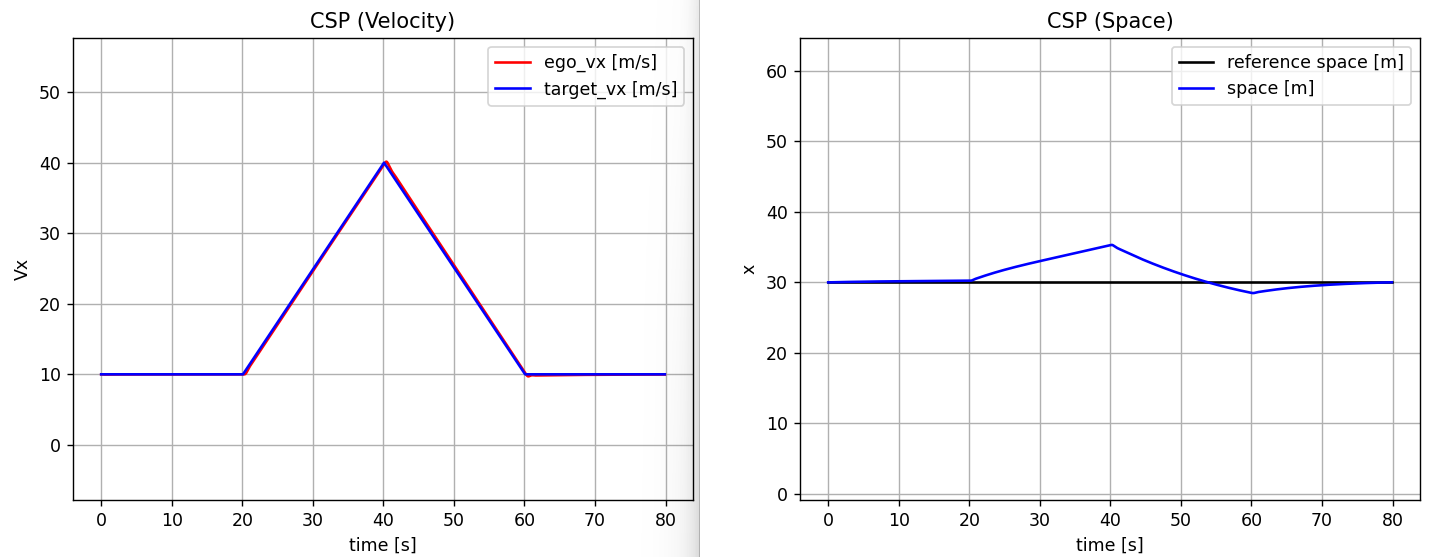
자동 생성된 설명**



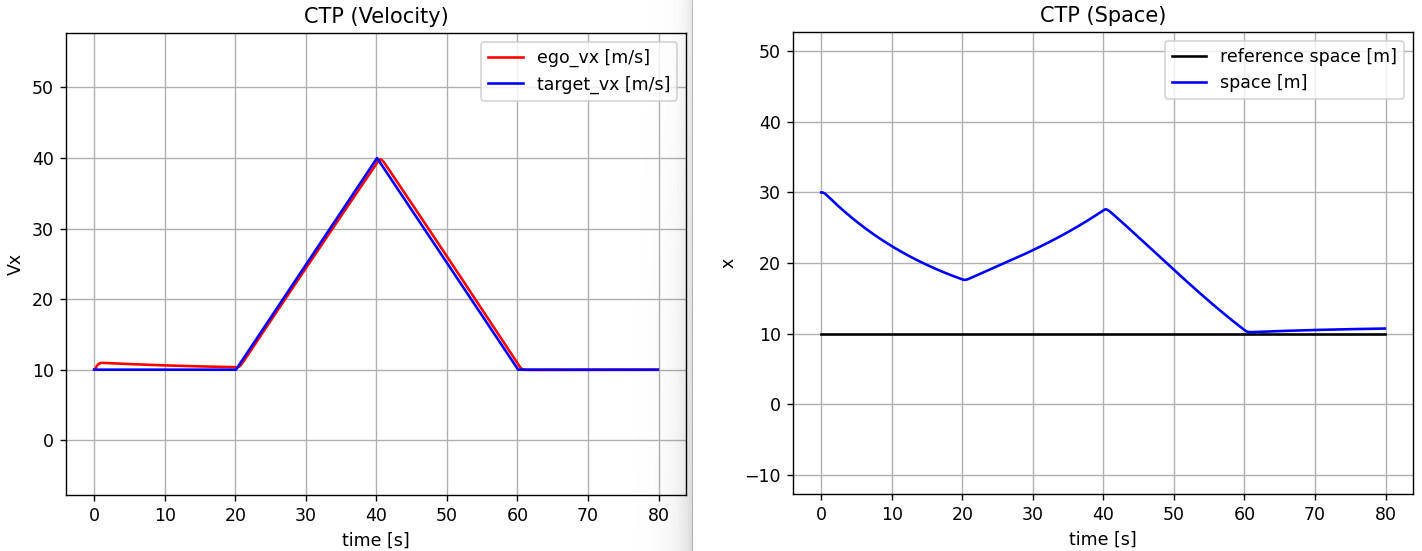
- target의 속도가 줄어들기 시작하는 peak 점부터 ego의 속도가 target의 속도를 넘어서는 40~50초 구간이 충돌 위험구간으로 보인다.

- 이를 개선하고자 1) d Gain 값을 키워보고, 2) I-Controller를 추가로 사용하는 방법을 생각했다.

- CSP (kp=0.4, kd=4.0)

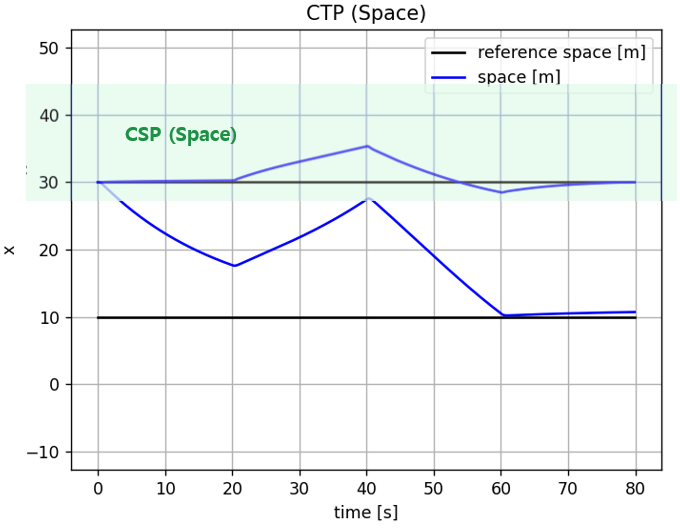


- CTP (kp=0.1, kd=1.9), I-Controller는 굳이 사용하지 않았다!!



- 이렇게 그래프로만 보면 CSP가 더 안정적인 것 아닌가? 라고 생각할 수 있지만, 실제 상황에서는 조금 다를 것이다.

- 두 그래프를 합쳤을 때, space의 scale을 동일하게 맞춰 놓고 보면 아래와 같다.



- 속도가 낮은 저속구간에서는 작은 space를 추종하고, 가속구간에서는 속도에 비례한 space 증가, 감속구간에서는 속도에 비례한 space 감소를 유연하게 적용시킬 수 있는 CTP방법이 실제 도로 상황에서 더 융통성 있는 방법이라고 생각된다.

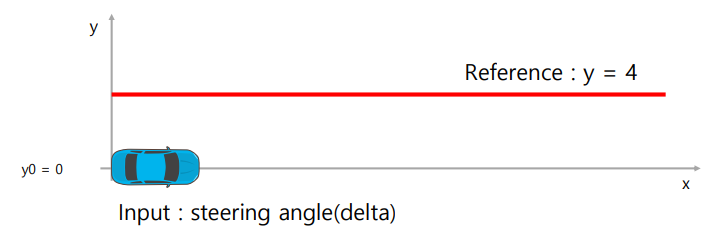
- 위 그래프의 CSP처럼 계속 30 이라는 상수로 지정된 reference space만을 따른다면, 시골 귀향 길 고속도로에서 다른 운전자들의 눈총을 살수도…?

**5. Kinematic model in Global Frame**

- Lateral vehicle dynamics에서 Kinematic Model은 아주 쉽게 유도된다.

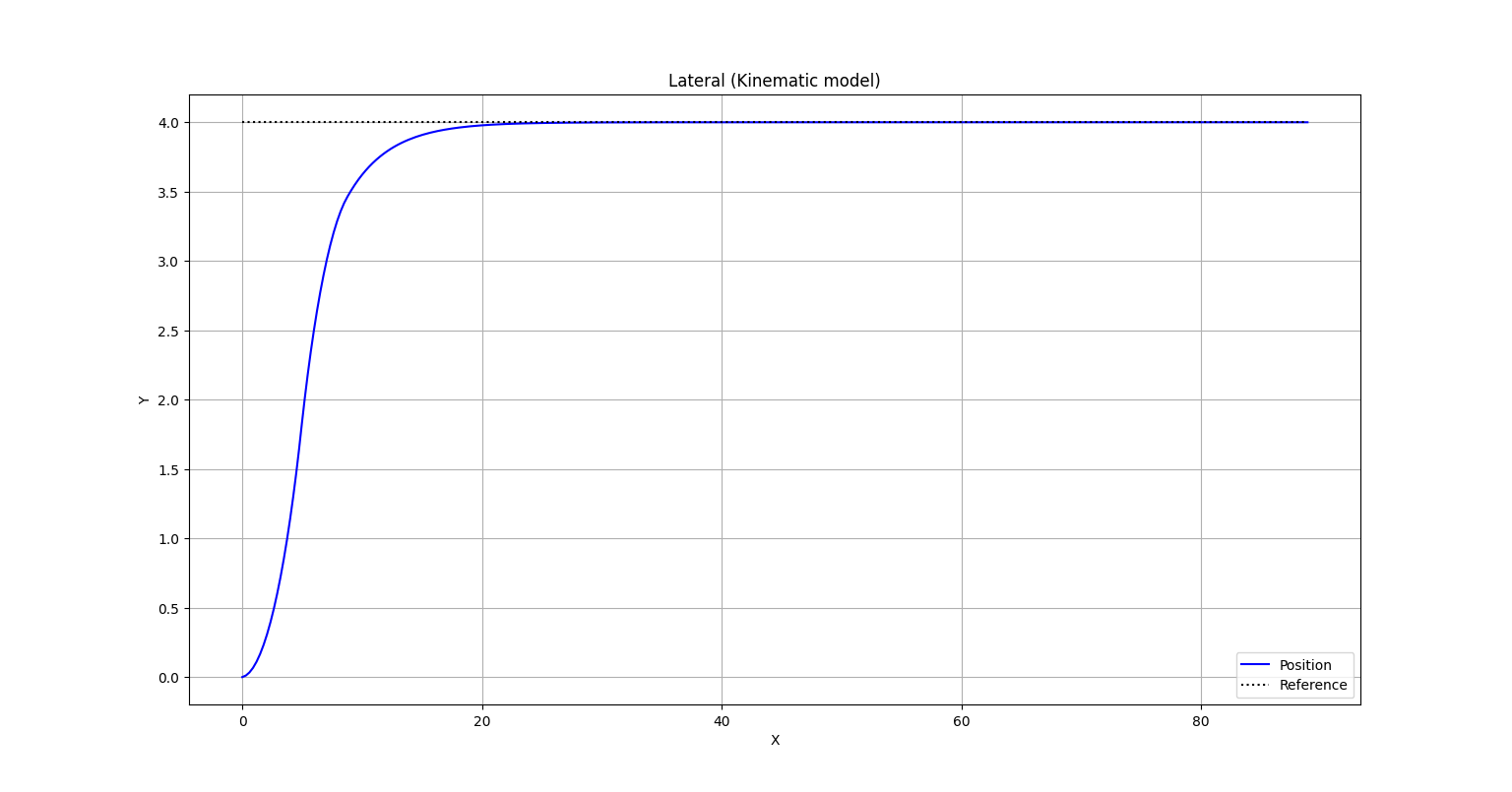
- TIL 기록용 velog에 정리해 놓았으니 여기선 따로 수식을 정리하지 않겠다.

- Kinematic Model을 이용하여 아래의 path를 따라가는 제어기를 설계해 보자.



- 이 kinematic Bicycle Model은 tire slip이 발생하지 않는 저속 구간에서 사용할 수 있다. (3 m/s)

- 아래는 PD 제어기(kp = 5.0, kd = 6.0)를 사용한 결과이다.



- delta의 한계를 np.clip(delta, -0.5, 0.5)로 제한하여 약 -28.65 ~ 28.65 degree 사이의 조향만 가능하도록 모델링 되어있었다.

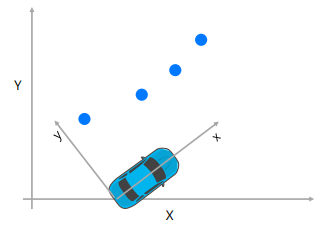
- 만약 이 한계를 더 늘린다면, 더 빠르게 수렴할 순 있지만, 운전자 입장에서 너무 급격한 핸들링을 요구하게 되는 것 일 수도 있다.

- 일단은 주어진 제한조건으로 과제를 수행했다.

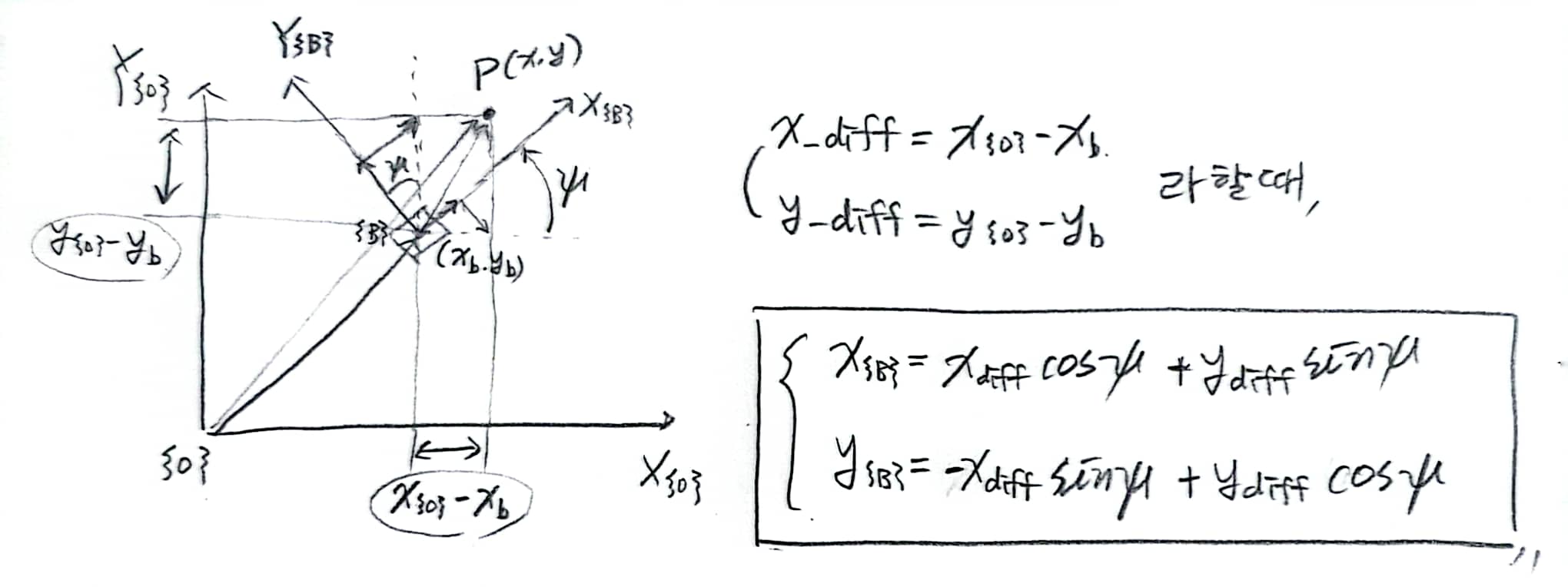
**6. Global Frame to Local Frame**

- 차량의 위치와 Reference를 Global Frame에서 표현하는 일은 매우 어렵기 때문에, Local Frame으로 변환해야 한다.

- 주어진 Point들을 Local Frame으로 변환하고, 3차 차선 방정식으로 Fitting해보자.



- 먼저 coordinate 정의를 먼저 하면, 아래와 같다.



- 회전변환 부호가 헷갈려서 직접 좌표계를 그려봤고, 해당 body frame에 대한 fitting을 진행했다.

- 코드 순서

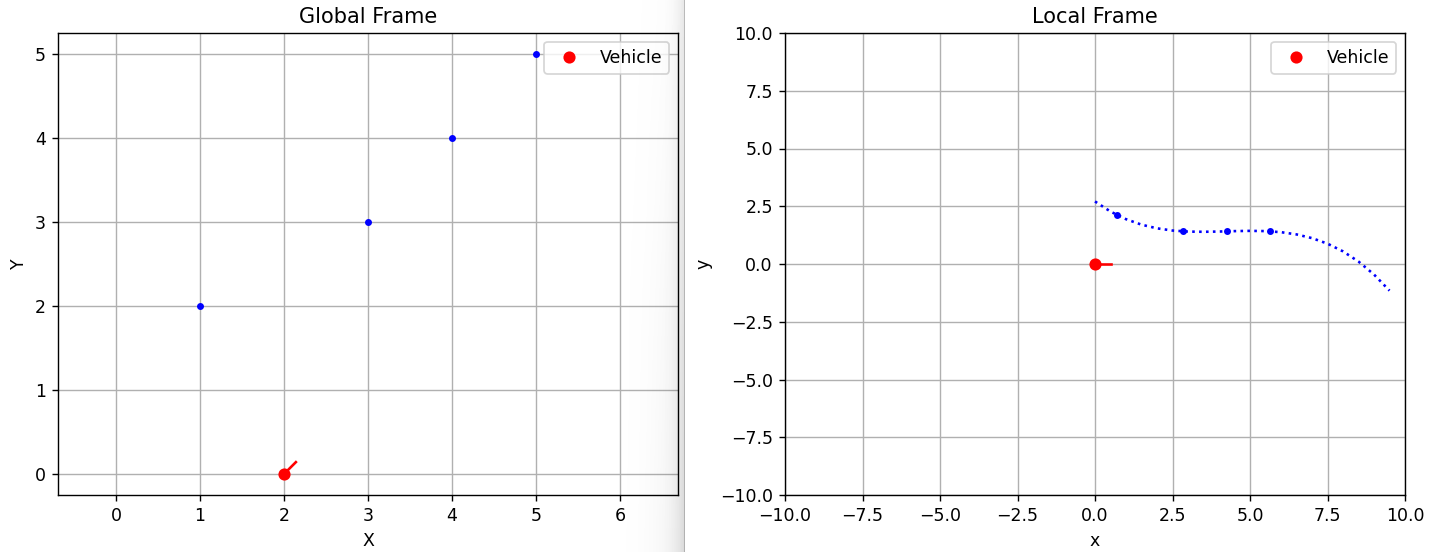
1) 글로벌 (1,2), (3,3), (4,4) (5,5) 좌표를 로컬 좌표 기준으로 변환 🡪 Global2Local 클래스

2) 변환된 로컬 좌표로 측정값과 후보 곡선 사이의 직선거리 에러의 제곱합이 최소가 되는 최적 곡선 방정식의 계수를 구함 🡪 PolynomialFitting 클래스

3) 계산된 곡선 방정식의 계수를 local\_x 값과 곱해 최종 계산된 local에서의 (x, y)\_local을 계산 🡪 PolynomialValue 클래스

- 역시 처음 짜는 코드였기 때문에 .py 파일 주석을 활용해 코드를 한줄한줄 모드 뜯어보았다…

- 아래는 fitting 결과이다.



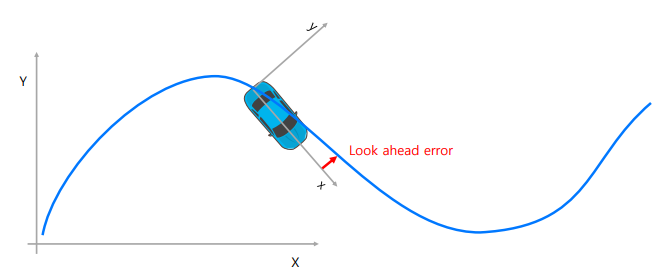
- 빨간색 작은 막대는 local frame의 x 좌표 단위벡터 방향을 나타내는 것 같다.

- 우려했던 좌표계 변환 시 부호가 틀리는 이슈가 발생하지 않고 잘 변환됐고, 동시에 fitting또한 잘 이루어진 것을 볼 수 있었다.

- 시간이 남는다면, 최소제곱해 coefficient를 구할 때 np.linalg.lstsq() 메서드를 사용하지 않고 직접 로 구현해 보는 것도 좋을 것 같다. 시간이 있으면..ㅎㅎ

**7. Lateral Control on Local Frame: PID Control**

- 6번의 Local Frame 변환 코드로 reference를 따라가는 제어기를 사용해 보고 성능을 개선시키자.



(Look ahead distance에서 error를 구하는 개념을 적용해보자) (속도 = 3m/s)

- 이 실습을 하면서 이걸 왜 하는지 알았다.

- **내가 지금 사용하는 kinematic model은 y 값의 error 즉, CTE를 기반으로 조향각 delta를 도출하는 모델이다.**

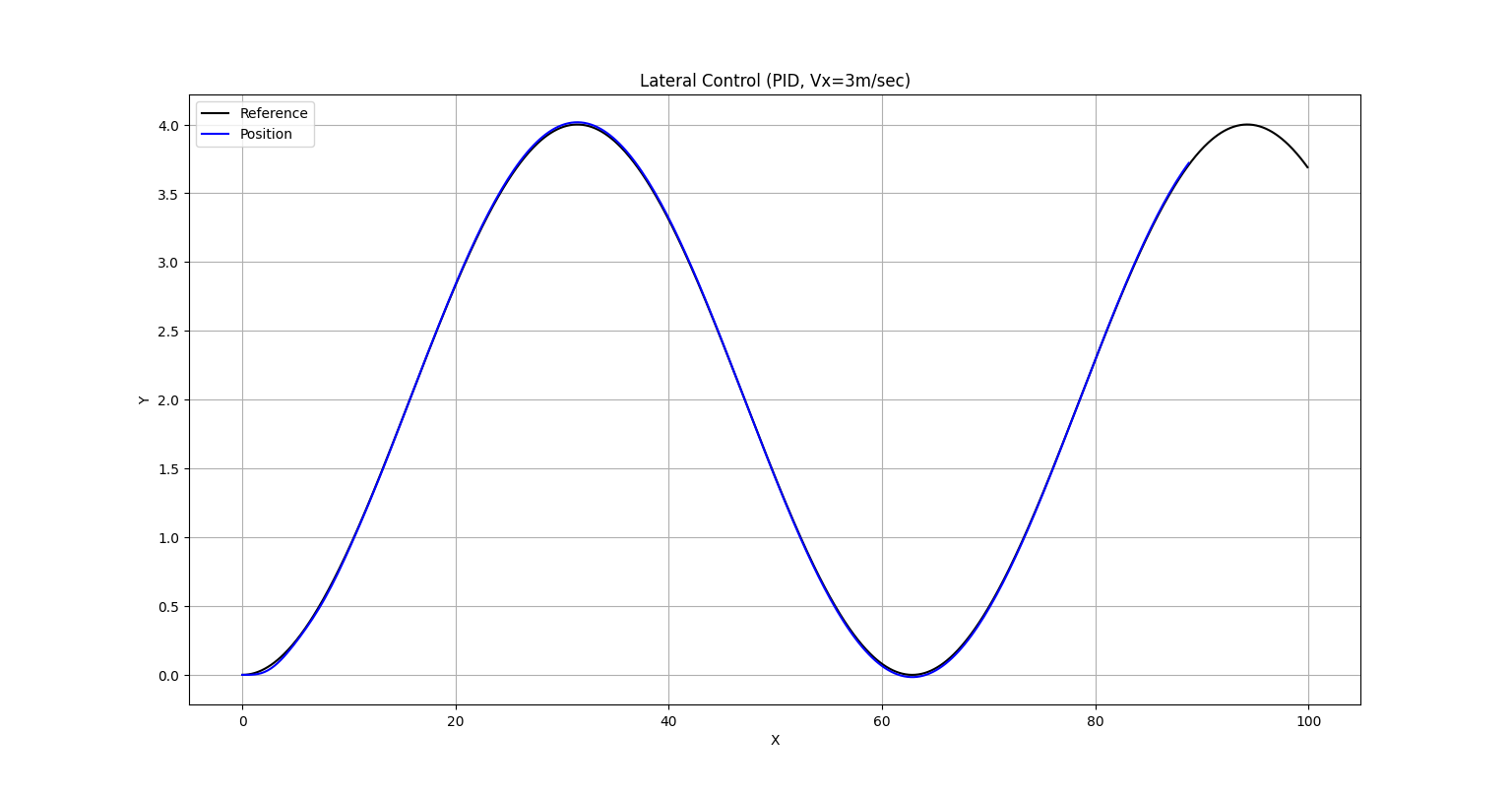
**하지만 글로벌에서는 차량의 자세를 고려함과 동시에 이 CTE를 계산해 내는 것이 까다롭다.**

**그래서 차체의 현재 위치를 로컬 좌표계로 변환하면,**

**차의 현재 위치를 0, desired path의 로컬 y 좌표를 reference 값으로 놓을 수 있고,**

**단순히 y\_ref 값 자체가 CTE가 되므로 보다 손쉽게 조향각 delta를 구할 수 있다.**

- 간단히 PD(kp=5, kd=0.5)제어기만을 사용했음에도 아래와 같은 좋은 결과를 얻을 수 있었다.



**8. Lateral Control on Local Frame: High Speed**

- 다소 고속 상황에서 (20m/s = 72km/h) 타이어의 횡 방향 슬립을 고려하여 Feed Forward 제어 term을 추가한 kinematic model을 사용하여 Reference를 따라갈 수 있도록 제어기를 설계해보자.

도표, 라인, 그래프이(가) 표시된 사진

자동 생성된 설명

- 먼저 고속에서의 CTE, heading angle error는 steady state steering angle error를 가진다. 이 delta는 정상상태 등속 선회 운동을 하는 차량의 neutral, over, under steer 거동을 표현하는 지표이기도 하다.

- 사실 cos 함수 특성 상 x에 대한 곡률이 모두 다르지만, 이를 수학적으로 표현하기는 아직 어려워서, 회전반경 R을 함수의 평균 곡률로 대체하였다. (R=1/0.019177)

이 때문에 이 모델은 모든 구간에서 neutral, over, under 중 한가지 특성만을 동일하게 표현하게 된다.

- 속도 Vx=20m/s로 기존 속도보다 6배가량 증가함에 따라, 시뮬레이션 시간을 1/6 배로 감소시켜 X\_ref 와의 scale을 맞춰주었다.

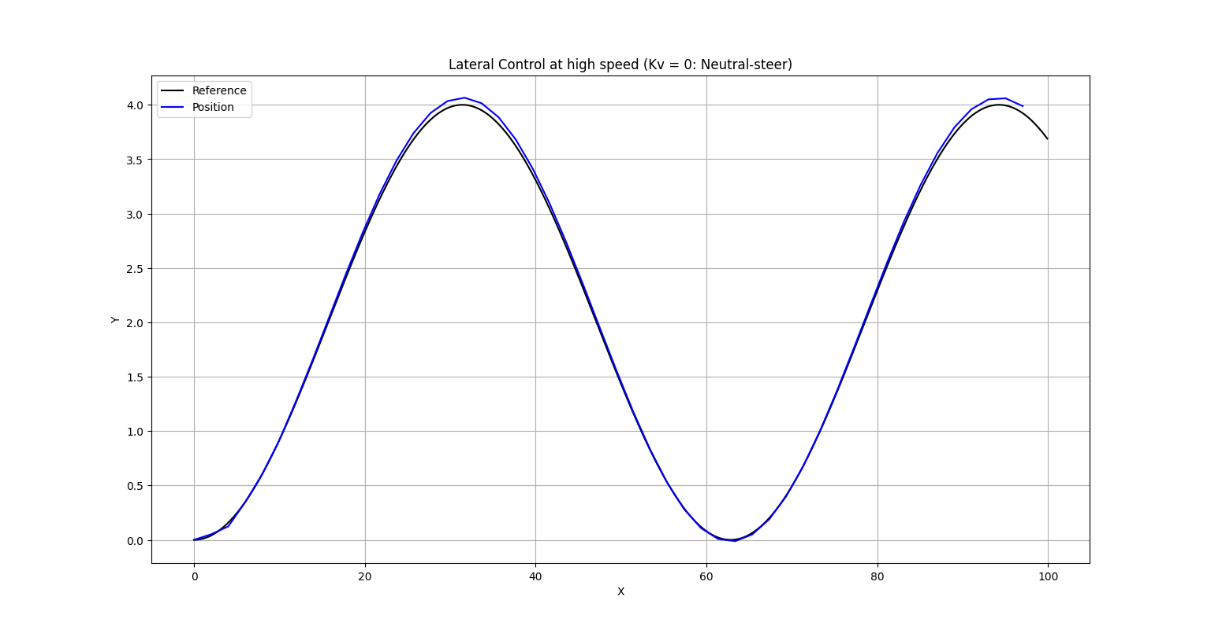
- Understeer, Oversteer라고 칭하기도 민망할 정도의 결과이지만, 설명의 편리함을 위해 목표조향 보다 덜 틀어진 경우(Kv>0)를 Under, 더 많이 틀어진 경우(Kv<0)를 Over라고 부르도록 하겠다.

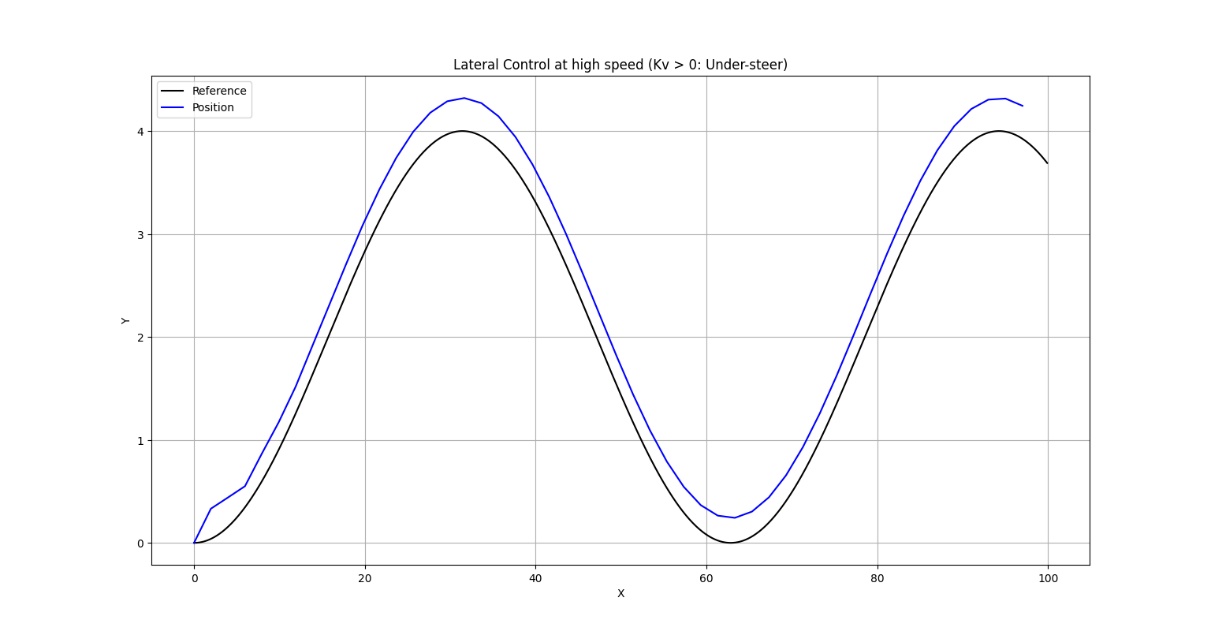
- PD 제어기(kp=3.0, kd=0.1)를 사용했으며, 타이어 슬립으로 인한 차체의 거동을 확인하기 위해 understeer gradient Kv = 0, 0.1, -0.1를 각각 사용하였다.

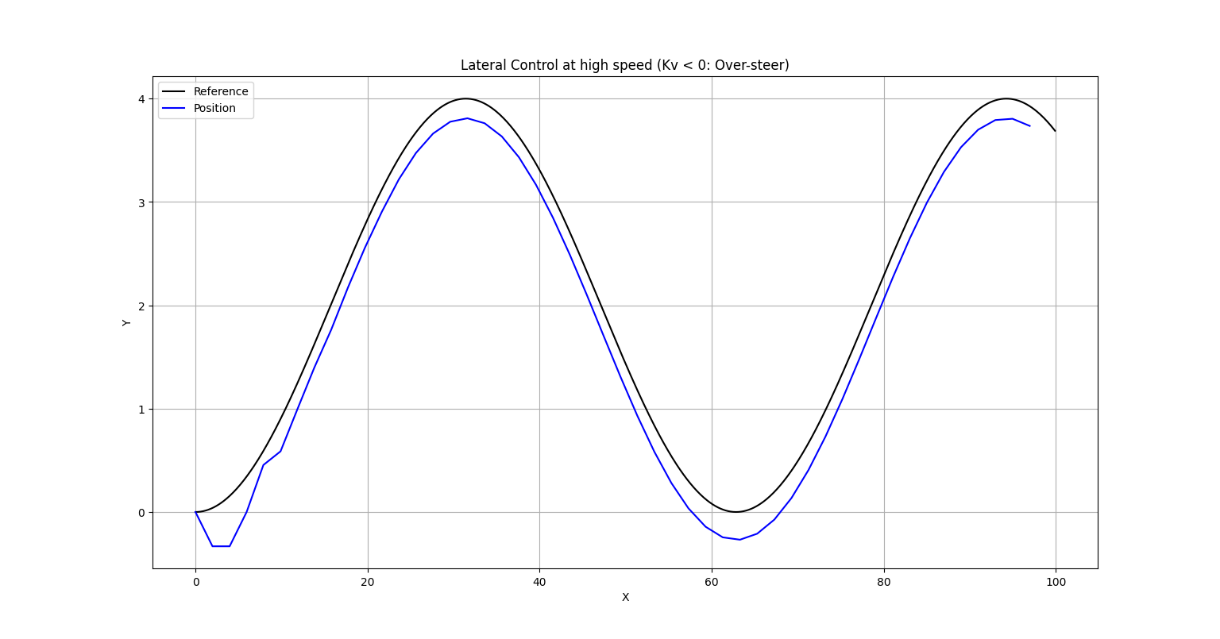
- 추후 회전반경(R)을 위치에 따라 변화하는 동적인 모델로 설계한다면 실제 이상적인 차량의 거동과 유사해질 수 있다.

- 나중에 생각한 대로 코딩할 수 있을 정도로 실력이 올라가면, 도로 경로 생성 모델로 배운 Clothoid의 위치 혹은 시간에 따른 곡률을 적용시킨 횡방향 제어를 실습해 보고 싶다.

- 아래는 차례대로 Kv가 0, 0.1, -0.1인 neutral, under, oversteer인 경우의 제어 결과이다.





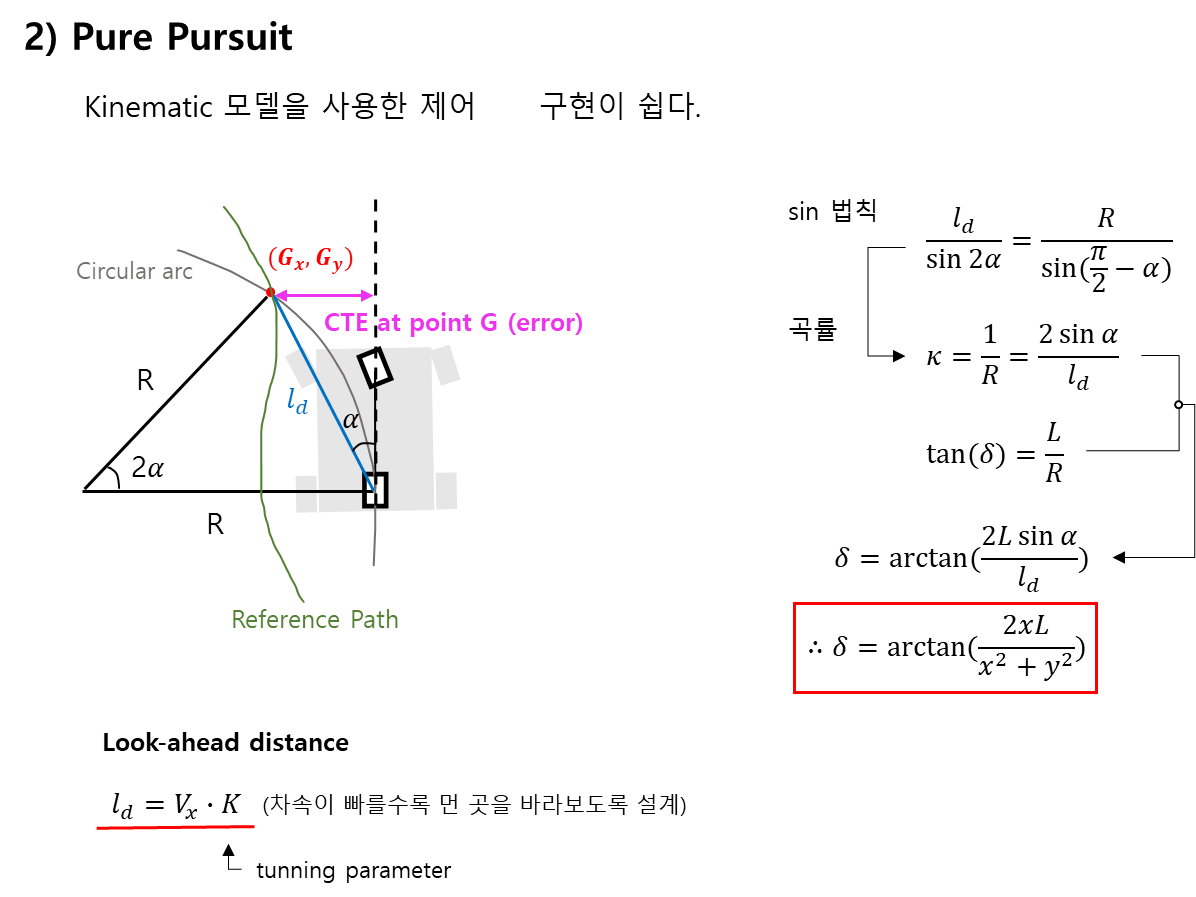


**9. Lateral Control on Local Frame: Pure pursuit**

- Pure Pursuit을 사용하여 reference를 간단히 따라갈 수 있는 제어기를 설계해 보자.

도표, 라인, 그래프이(가) 표시된 사진

자동 생성된 설명



- Pure pursuit의 조향각 생성 모델을 사용해 보았다.

- ld = K \* Vx의 형태로 설계하고 싶었으나, ld에 있는 desired-path의 좌표 (Gx, Gy)를 구하는 방법을 몰라서, 로컬 origin 기준으로 속도에 비례한 x값을 Xp=K\*Vx라 두고, ld를 상수로 지정한 뒤, waypoint의 좌표를 예상하여 제어해 봤다.

- 근데 잘 안됐다… 이건 어떻게 해야 되는지 모르겠다. 그냥 고민해 본 코드만 첨부하겠다.

**10. Lateral Control on Local Frame: Stanley Method**

- Stanley Method를 사용하여 reference를 간단히 따라갈 수 있는 제어기를 설계해 보자

도표, 라인, 그래프이(가) 표시된 사진

자동 생성된 설명

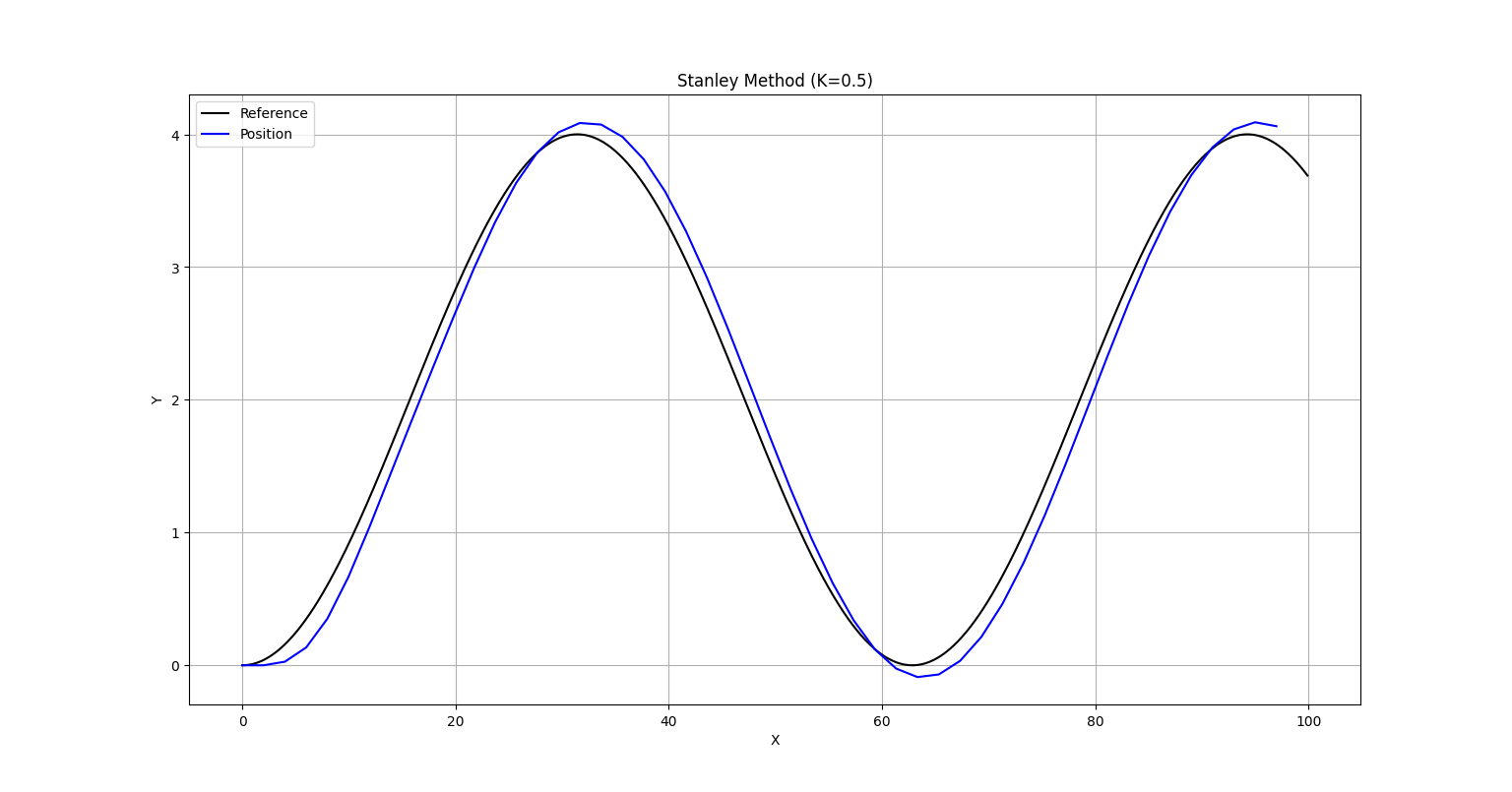
스크린샷, 도표, 라인, 직사각형이(가) 표시된 사진

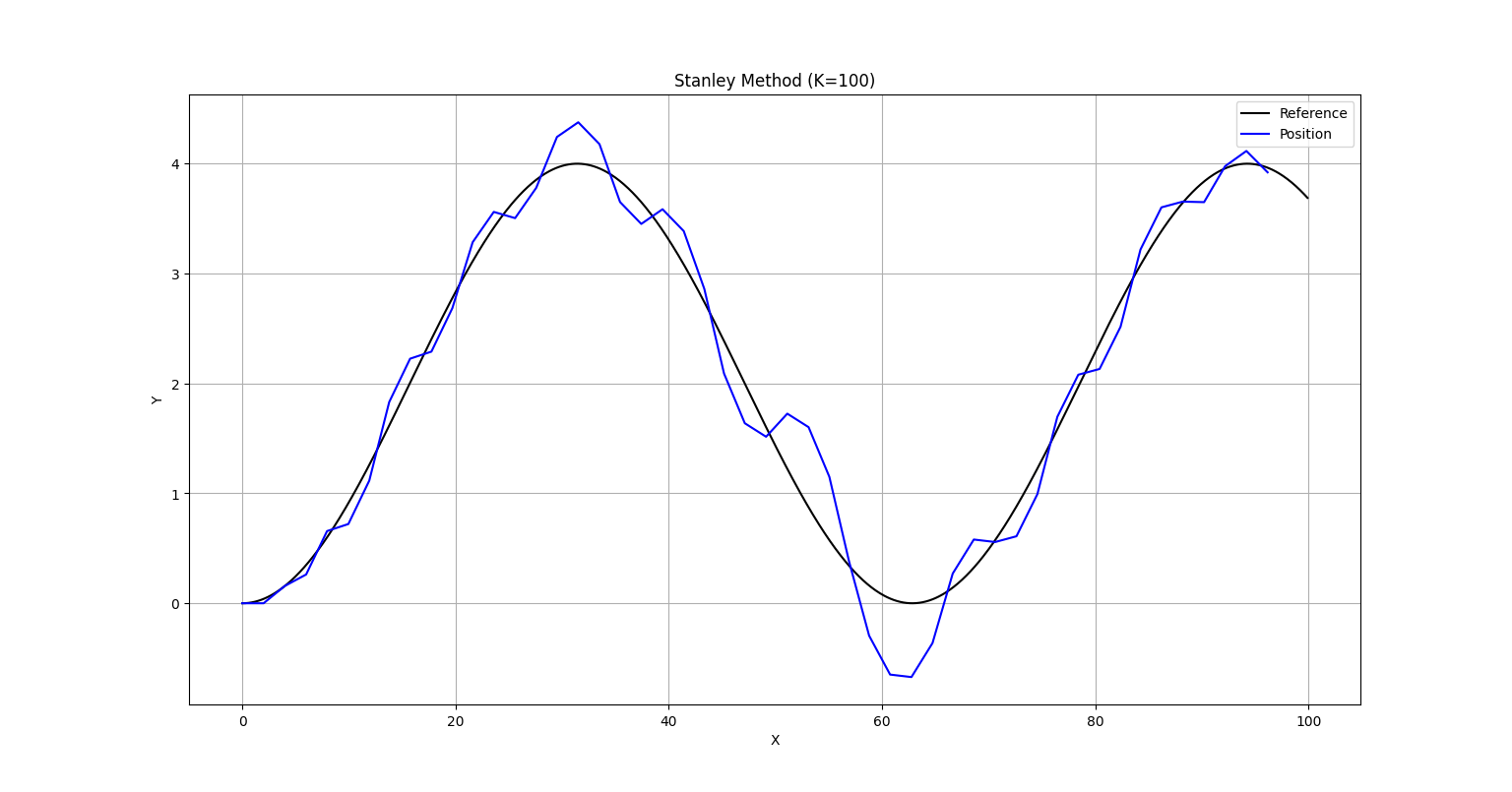
자동 생성된 설명

- 난 개인적으로 stanley가 더 간단하고 쓰기 좋은 것 같다. 물론 적절한 k값을 선택하는 것이 많이 어렵다고 하지만, 상수로 놓고 썼을 때 가장 구현이 쉬웠고 직관적인 것 같다.

- k gain 값이 클수록 차량의 반응이 민감하다고 하는데, 통상적으로 많이 쓰인다는 0.5 기준으로 했을 때 100정도의 드라마틱한 변화를 줘야 눈에 좀 보이는 것 같다.

- 아래는 순서대로 k=0.5일 때, k=100일 때의 결과이다.





<과제 후기>

한 문제 한 문제 최선을 다해 풀었고 코드를 정말 씹어 먹어보려고 많이 노력했다.

9번 문제를 못 풀어서 아쉬운데, 고민은 많이 해봤으니 다음에 만나면 해결법을 찾을 수 있을 것이라고 생각한다.

더 많은 것을 시도해 보고 싶은 부분도 있었지만, 한정된 시간 관계상, 또 다른 과제와 실습들을 진행해야 하는 일정상 아쉽게 여기서 마무리하겠다.