진행상황 공유

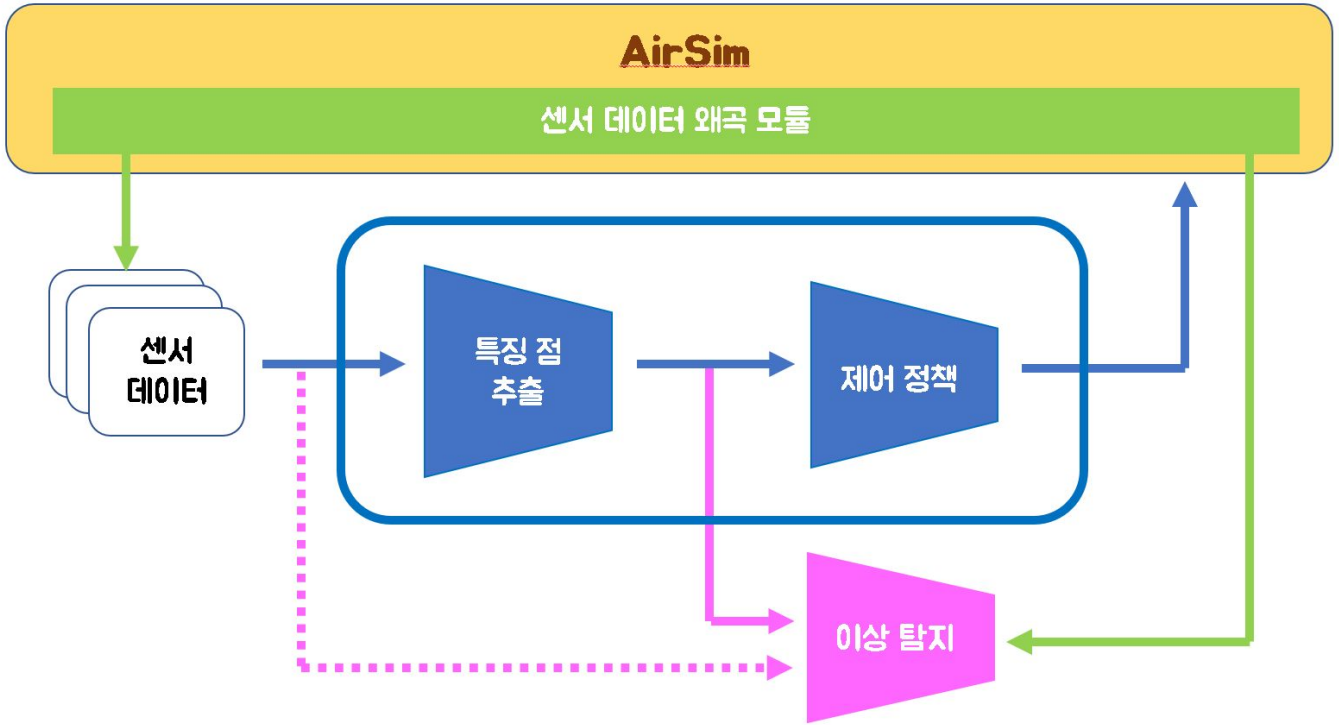
2023.6.28

연구 목표

- 관성 측정 장치의 (IMU: Inertial Measurement Unit) 센서 값 오류를 발생시켜 드론 제어를 손쉽게 무력화할 수 있으니 이에 대한 대응 방안 기술이 부족함.

- 본 연구에서는 강화학습 기반 센서값 오류를 재현시켜 오류를 빠르게 탐지하고 대응력을 향상시킴

학습 모델 개요



환경 소개

- AirSim by Microsoft Research
  - Unreal-Engine 4.27, Airsim 1.8.1
  - 드론 모델: DJI F450
  - 입력 데이터: 자이로 센서, 가속센서, GPS 위치

- 센서 값 왜곡 모듈 개발
  - 자이로센서 & 가속센서 값 왜곡

진행사항

1) AirSim 환경 구축 완료

2) AirSim 구조 분석 및 센서 데이터 왜곡 생성 모듈 개발 완료

3) 센서 데이터 왜곡 정도에 따른 규칙 기반 자세 제어 성능 도출 완료

4) 드론 자세 제어를 위한 강화학습 모델 설계 및 1차 학습 결과 도출

=> 아직 강화학습 모델 기반 성능이 부족함 (추후 개선)

센서 데이터 왜곡 생성 방식 (Gaussian noise)

➢ **IMU Sensor 정보에 왜곡 정도에 따라 일정 수치를 더해 왜곡을 표현**

IMU 각 센서값들의 min-max 범위 지정 (※실험 결과 및 드론 제원 참고)

1. **orientation(드론이 바라보는 방향) : -1 ~ 1**
2. **linear acceleration(선형 가속도) : -10 ~ 10 [m/s²]**
3. **angular velocity(각속도) : -3.14 ~ 3.14 [rad/s]**

해당 센서값의 크기에 왜곡 정도를 곱한 뒤 이를 표준편차로 하는 정규분포 생성

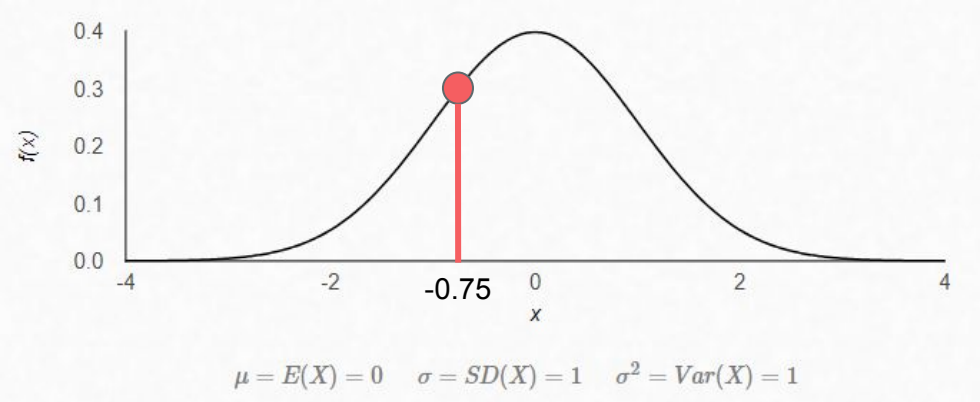
현재 드론의 센서 값에 정규분포에서 랜덤으로 추출된 값을 더해 왜곡 표현

센서 데이터 왜곡 생성 방식 (Gaussian noise)

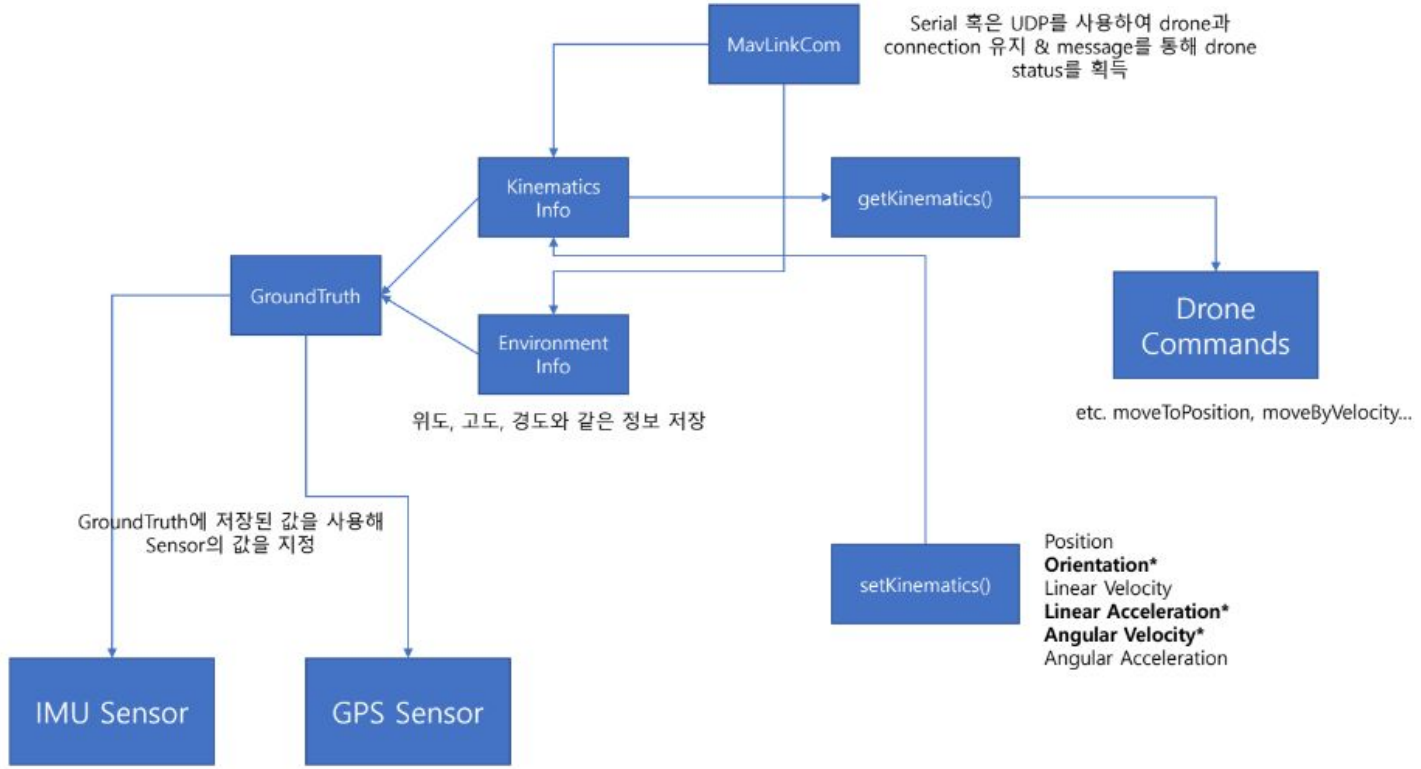
ex) 선형 가속도가 6m/s²인 드론에 10%의 왜곡이 주어졌을 때

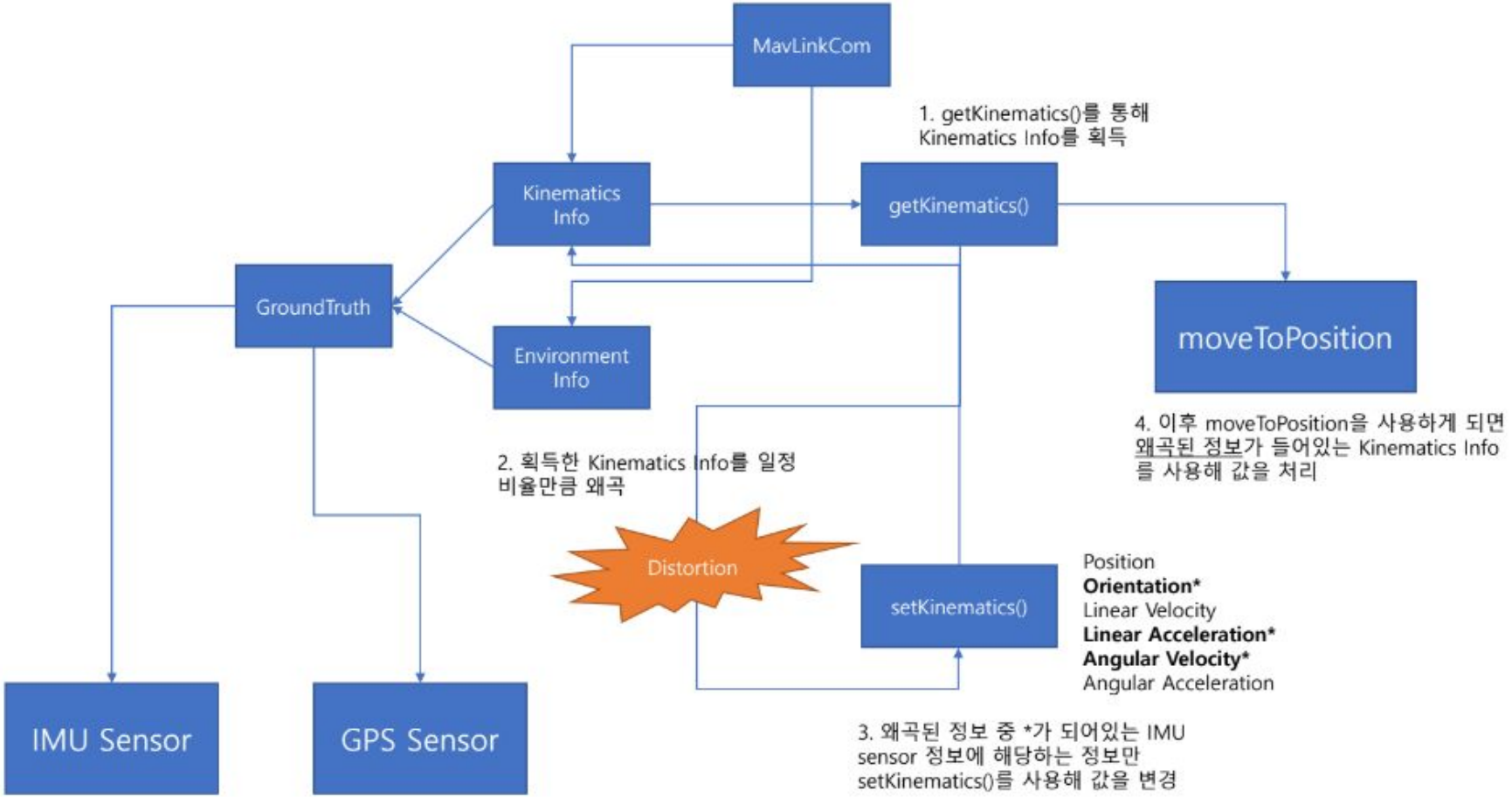
1. 선형 가속도의 크기인 10에 왜곡 정도 10% 를 곱한 값 계산 ⇒ 1 (→ 표준편차)
2. 위에서 계산한 값을 표준편차로 하고 평균이 0인 정규분포 생성
3. 해당 normal distribution에서 확률밀도에 따라 랜덤하게 x값 추출
4. 추출된 x값을 기존의 선형 가속도 6m/s²에 더해 왜곡을 표현 (bias)

➤ **왜곡 후 선형 가속도:** 6 - 0.75 = **5.25m/s²**



센서 데이터 왜곡 모듈 구현 (Airsim 으로 통합)




MavLinkCom
1. getKinematics()를 통해
Kinematics Info를 획득
Kinematics
Info
getKinematics()
GroundTruth
Environment
Info
moveToPosition
4. 이후 moveToPosition을 사용하게 되면
왜곡된 정보가 들어있는 Kinematics Info
를 사용해 값을 처리
2. 획득한 Kinematics Info를 일정
비율만큼 왜곡
Distortion
setKinematics()
Position
Orientation*
Linear Velocity
Linear Acceleration*
Angular Velocity*
Angular Acceleration
IMU Sensor
GPS Sensor
3. 왜곡된 정보 중 *가 되어있는 IMU
sensor 정보에 해당하는 정보만
setKinematics()를 사용해 값을 변경

사용한 모델

1. **룰베이스 알고리즘**
   high-level api인 moveToPosition에 목표 위치(goal location)를 넣어서 항상 목표 위치로 이동

```
self.client.moveToPositionAsync(0, 0, -20, 5, vehicle_name=self.drone_id)
```

2. **Position RL 모델**
   high-level api인 moveToPosition에 현재 위치로부터 변화량을 더해 변경된 위치로 이동

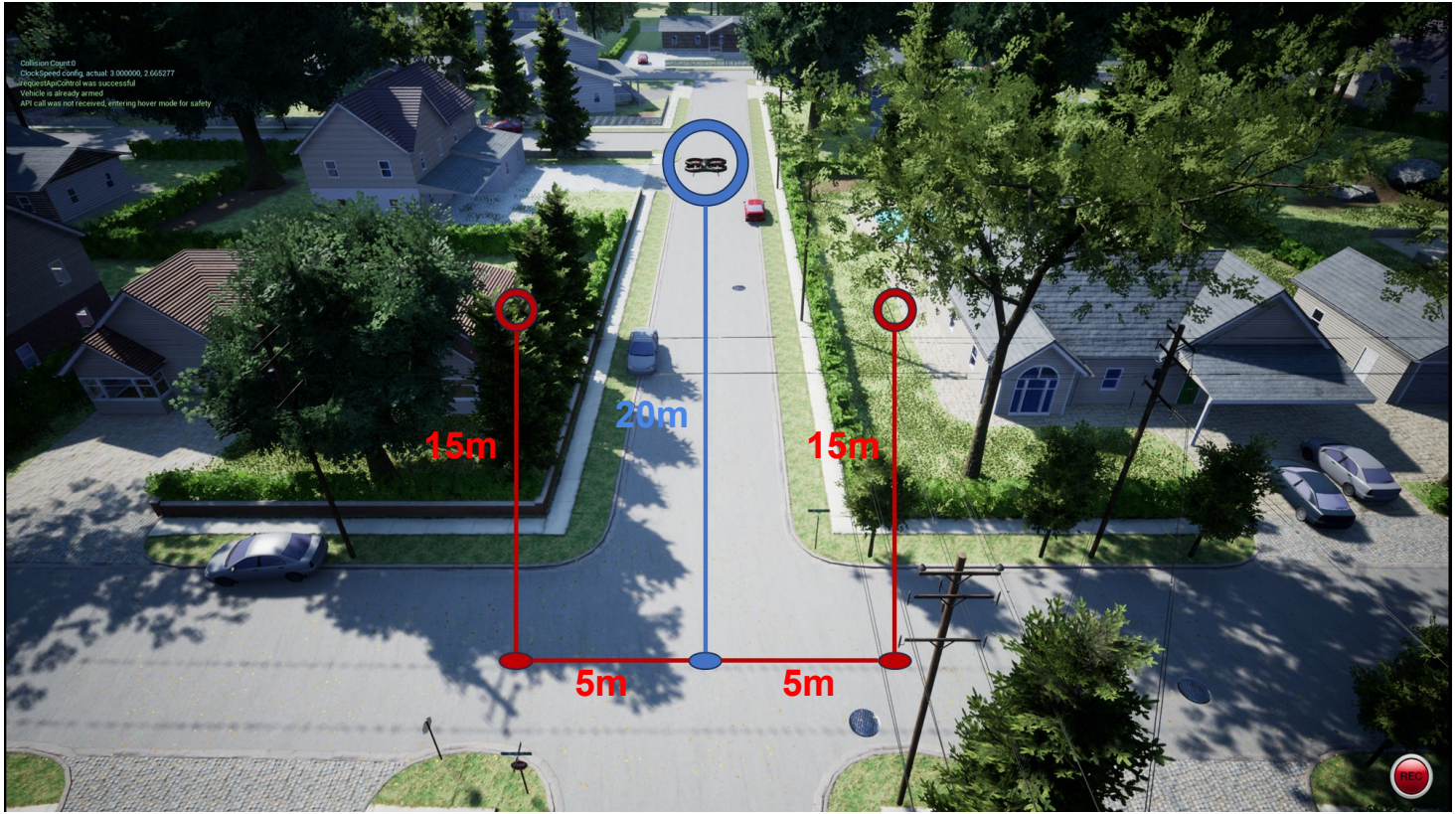
```
self.client.moveToPositionAsync((pos_info.position.x_val + float(action[0]) * 2),
                                (pos_info.position.y_val + float(action[1]) * 2),
                                (pos_info.position.z_val + float(action[2]) * 2),
                                5, vehicle_name=self.drone_id)
```

3. **Velocity RL 모델**
   middle-level api인 moveByVelocity에 현재 속도로부터 변화량을 더해 속도 변경

```
self.client.moveByVelocityAsync((pos_info.linear_velocity.x_val + float(action[0]) * 1),
                                (pos_info.linear_velocity.y_val + float(action[1]) * 1),
                                (pos_info.linear_velocity.z_val + float(action[2]) * 1),
                                1, vehicle_name=self.drone_id)
```

train & test 모두 센서 데이터 왜곡이 없는 경우

테스트 환경 설명



➤ **25만 스텝 학습시킴**

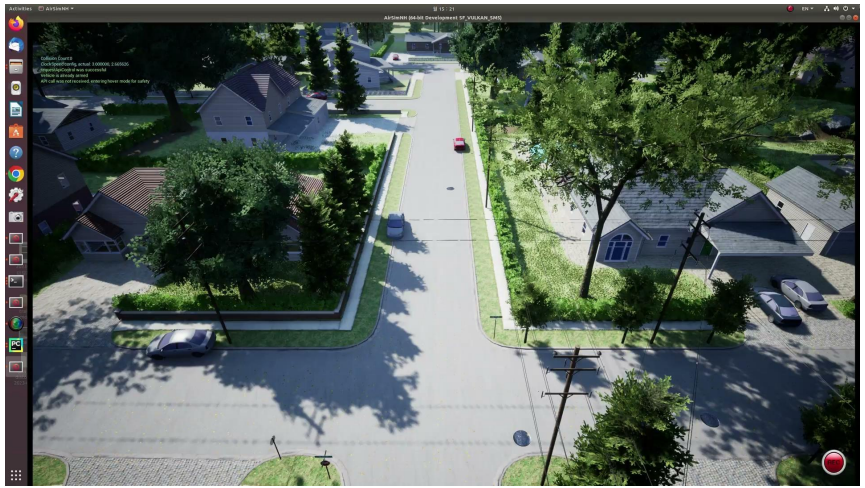
➤ **10 step 부터 왜곡 발생**

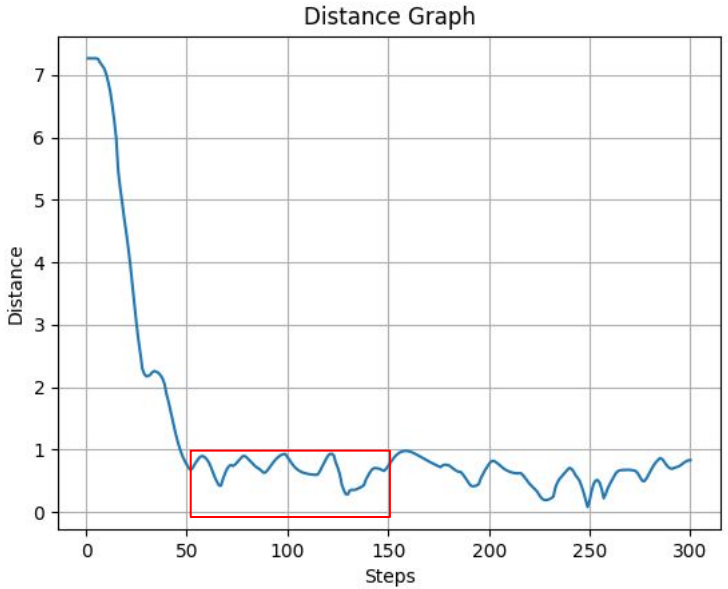
➤ **단위는 미터(m)임**

Start location

Goal location

Position RL 모델 (왜곡 없이 hovering)

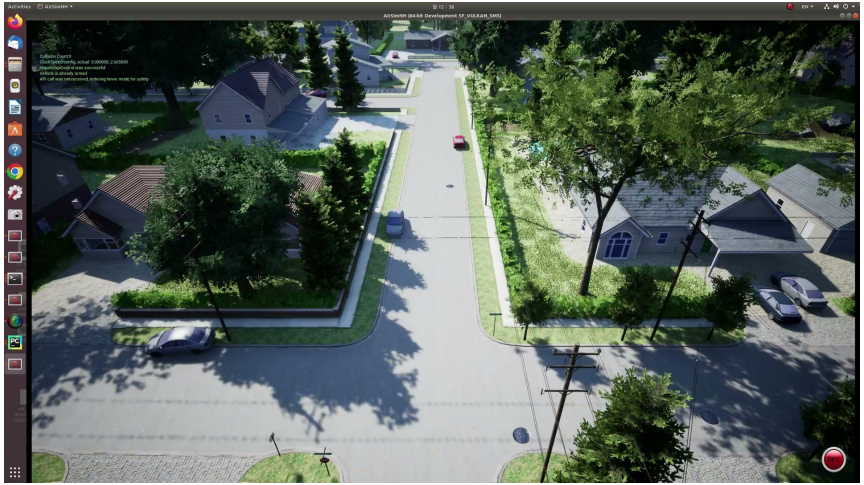


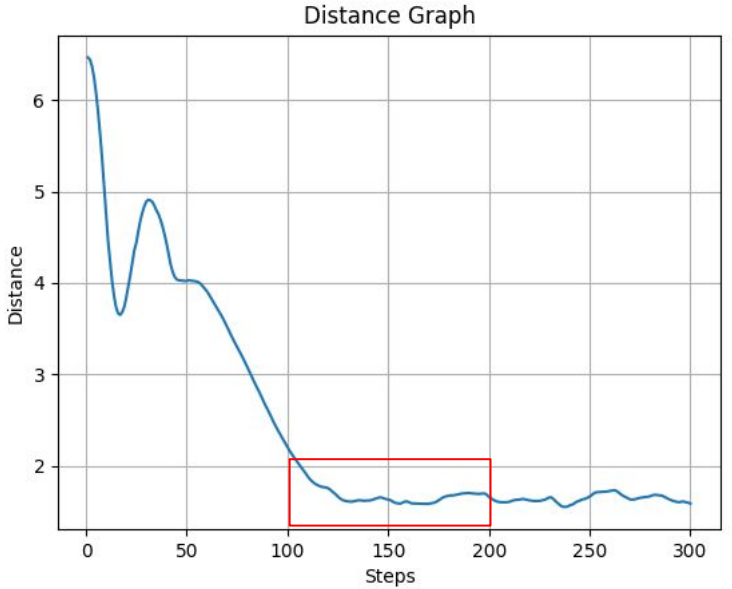


➢ **distance**란 목표위치인 **(0,0,20)**에서 현재 **drone**의 위치 사이 **Euclidean distance**를 의미함.

학습안정화 이후 **100steps**
average distance : 0.7

Velocity RL 모델 (왜곡 없이 hovering)





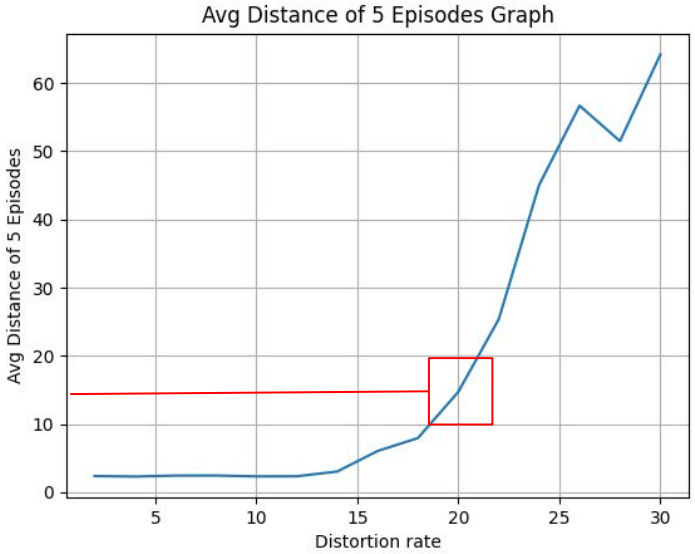
➢ **position과 velocity 모두 안정적으로 학습 완료.**

학습안정화 이후 100steps
average distance : 1.5

test 과정에서만 센서 데이터 왜곡을 넣은 경우

1) rule base
2) train + no distortion
3) train + distortion

룰베이스 알고리즘의 왜곡에 따른 성능 관측 결과



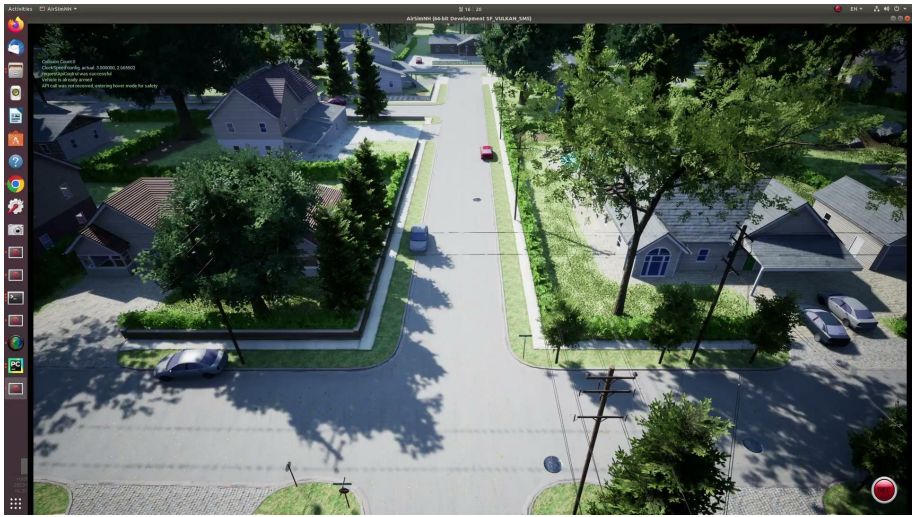
➤ 왜곡을 **2% ~ 30%**까지 점진적으로 **2%**씩 증가시키며 해당 왜곡 값에 따른 학습 안정화 이후 **100 step**동안의 **distance** 평균을 구함.

➤ 각 왜곡마다 **5**번의 **episode** 평균을 구했음.

➤ 룰베이스의 경우 **14%**부터 점차 불안정한 모습을 보이다가 **20%** 이후로는 급격히 불안정함.

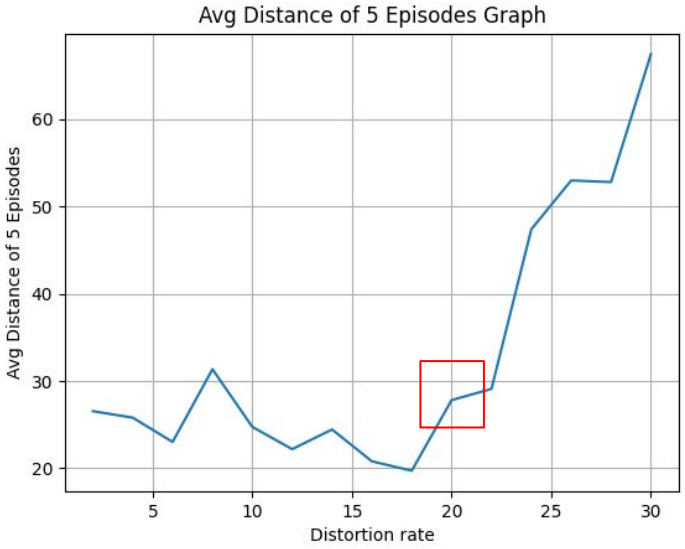
그러므로 **20%**를 기준으로 **rulebase**와 **RL** 모델을 비교하고자 함.

룰베이스 성능 예시 (20% distortion)



➤ **20%** 왜곡시 **rulebase**의 영상

Position RL 모델의 왜곡에 따른 성능 관측 결과



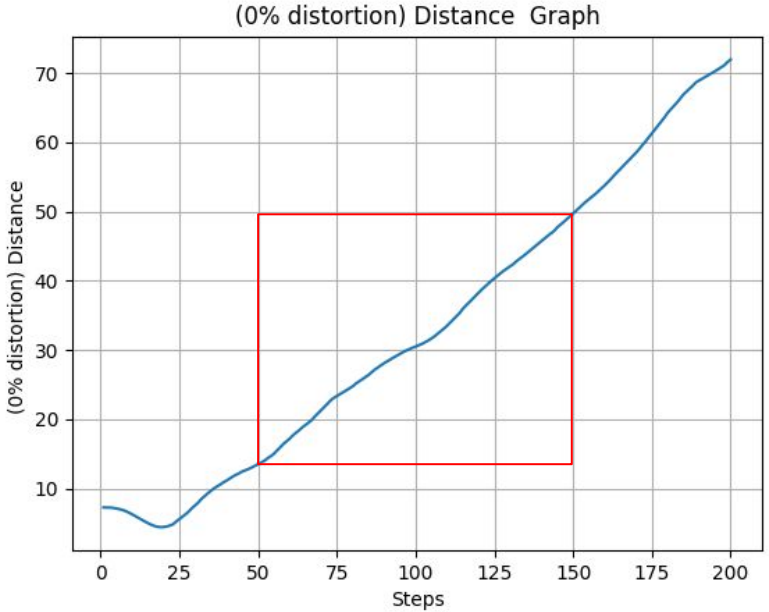
train with no distortion

➤ 룰베이스와 유사한 방식으로 왜곡을 **2% ~ 30%** 까지 점진적으로 **2%**씩 증가시키며 해당 왜곡 값에 따른 학습 안정화 이후 **100 step**동안의 **distance** 평균을 구함.

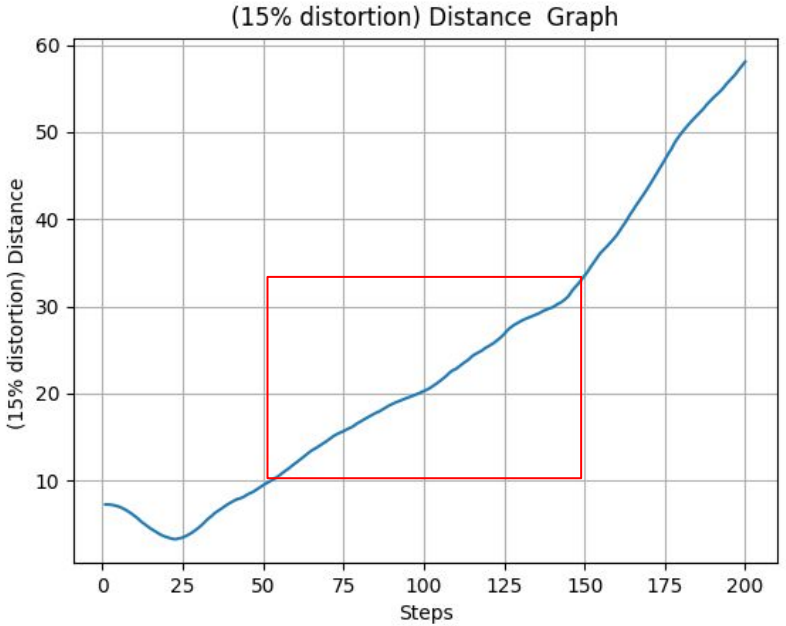
➤ 각 왜곡마다 **5**번의 **eposode** 평균을 구했음.

➤ **15%** 의 왜곡을 주고 학습시킨 모델을 테스트함.

Position RL 모델 성능 예시(with 20% distortion)

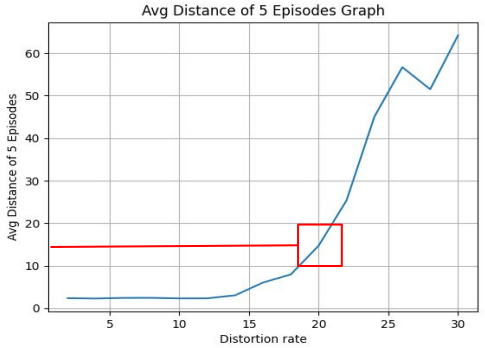


(W/O Distortion in Training)
Average Distance : 30.2

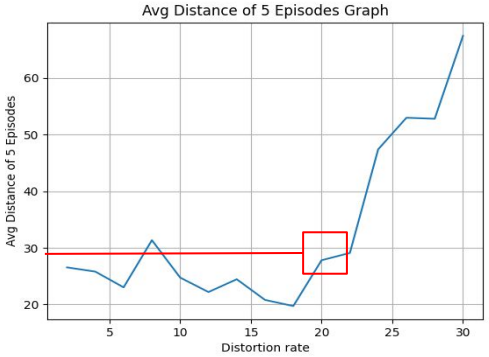


(W 15% Distortion in Training)
Average Distance : 19.1

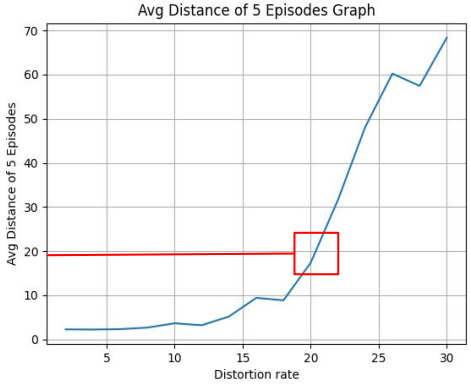
결과 비교(rulebase와 position RL)



rulebase



no distortion



15% distortion

	20% distortion
Rulebase	**14.7(±0.15)**
Position no distortion	28.1(±0.25)
Position 15% distortion	19.9(±0.2)

train때 적절히 왜곡을 경험한다면,
test에서 더욱 강건한 모습을
보여줄 수 있을 것으로 기대

추후 계획

1) 일정 길이 입력 데이터를 묶어서 사용

2) 학습 중 왜곡 정도를 점진적으로 높이는 커리큘럼 방안 적용

3) 지도학습 기반 왜곡 유무/정도를 추론하는 이상 탐지 모델 개발

4) 이상 탐지 모델과 강화학습 제어 모델을 통합 설계 및 개발

Appendix

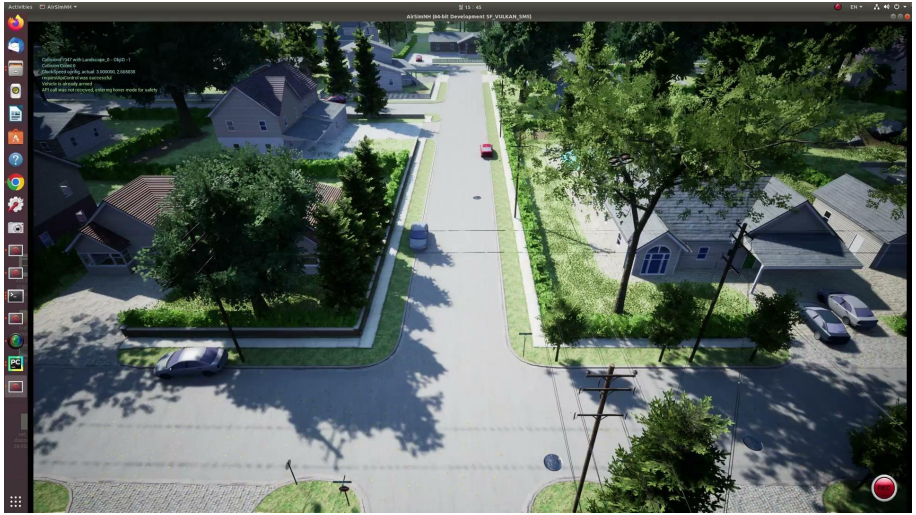
```
model = SAC(
    'MlpPolicy',
    env,
    verbose=1,
    buffer_size=200000,
    learning_rate=0.0003,
    batch_size=512,
    gamma=0.95,
    tau=0.005,
    ent_coef='auto',
    target_entropy='auto',
    train_freq=1,
    gradient_steps=1,
    seed=random_seed
)
```

➢ **Position, Velocity RL 모두 SAC를 사용하고 있음.**

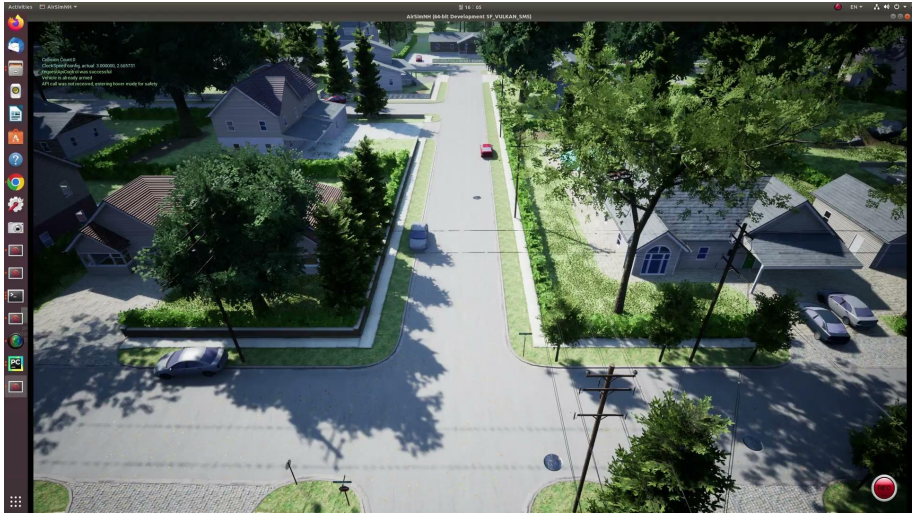
➢ **buffer size는 20만개, learning rate는 0.0003을 사용중임.**

➢ **추가적으로 Airsim simulator는 RHS 좌표계를 사용하며 전후를 x축, 좌우를 y축, 상하를 z축으로 함.**

Position RL 모델 성능예시 (with 20% distortion)

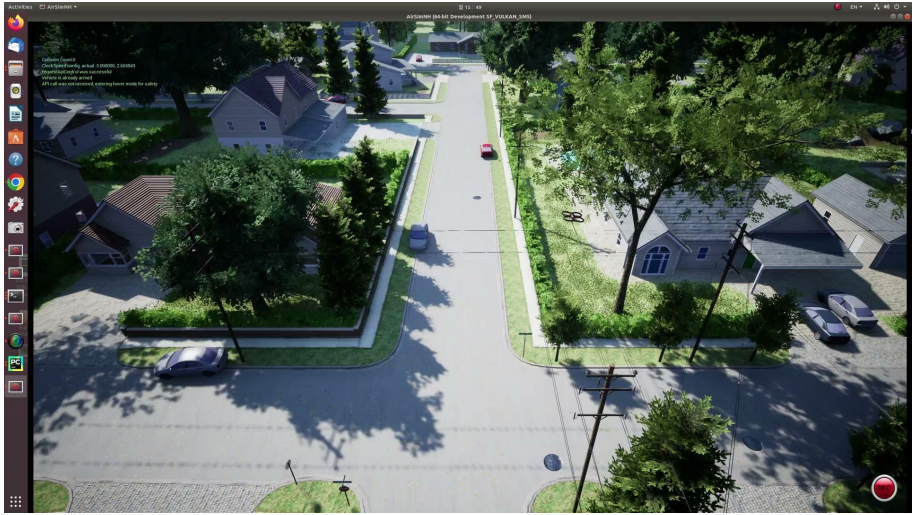


Train model without distortion

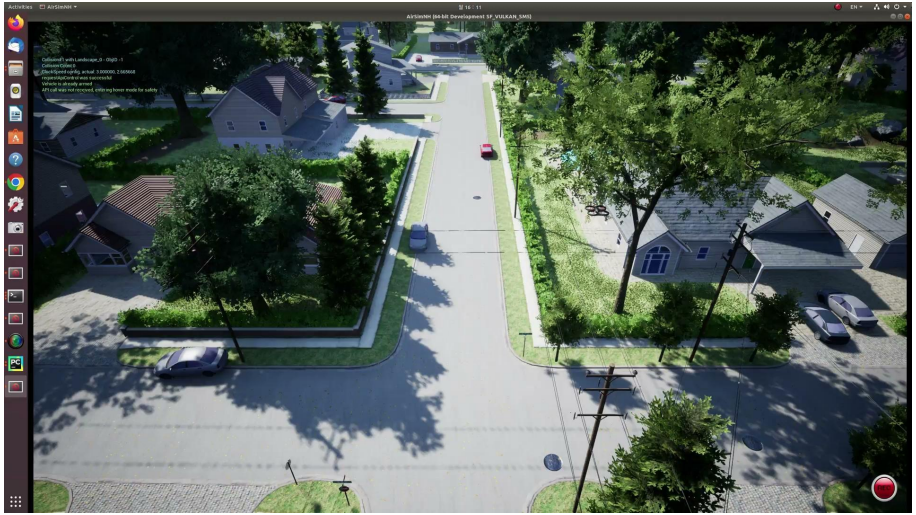


Train model with 15% distortion

Velocity RL 모델 성능 (with 20% distortion)

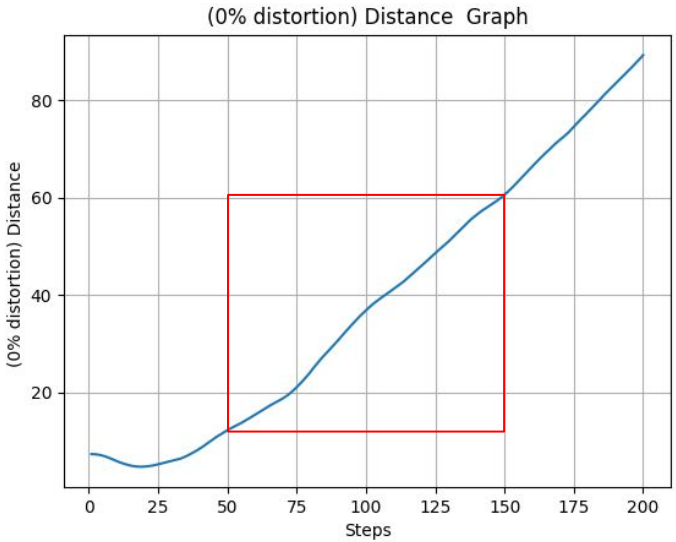


Train model without distortion

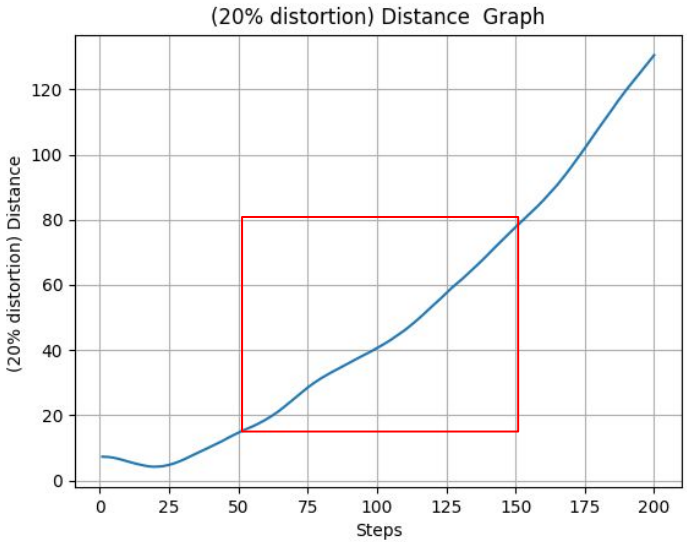


Train model with 15% distortion

Velocity RL 모델 성능 (with 20% distortion)



(W/O Distortion in Training)
Average Distance : 35.2



(W 15% Distortion in Training)
Average Distance : 44.2

Rulebase와 RL 모델 비교 및 추후 발전 방향

➢ **Position RL**의 경우 **distortion**을 늘려 학습시킬수록 **Rulebase**에 근접해가는 것을 확인함.

➢ **Velocity RL**의 경우 **distortion** 없이 **test**했을 때도 **Position RL**에 비해 안좋은 성능을 보였던 것처럼 **Error**에도 강건하지 못함을 확인함.

➢ **Velocity RL**의 경우 추가적인 **hyperparameter tuning** 등의 방법이 필요해보이고, **Position RL**의 경우
State를 **Stacking**하거나 **Auxilary task**를 이용하여 **Distortion** 정보로 **loss**를 추가로 **update**하는 방향으로 진행할 예정.

	No Distortion	15% Distortion
Rulebase	**14.7**	**14.7**
Position	28.1(±0.25)	19.9(±0.2)
Velocity	35.2(±0.5)	44.2(±0.65)