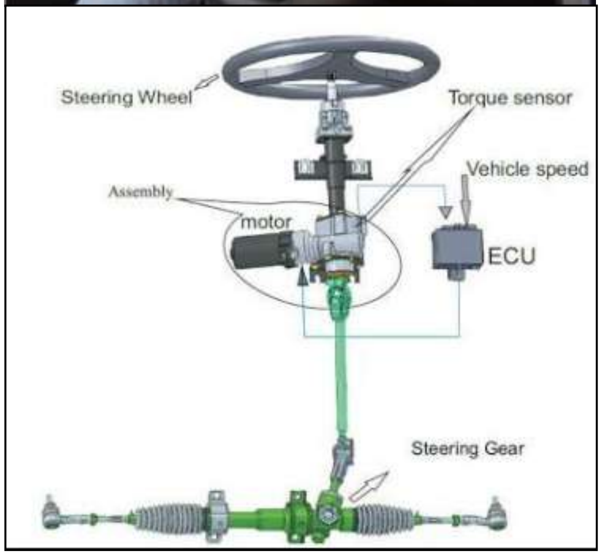
석사 졸업 과제

: 강화학습을 이용한 EPS system steering tuner 자동화 연구

문성주

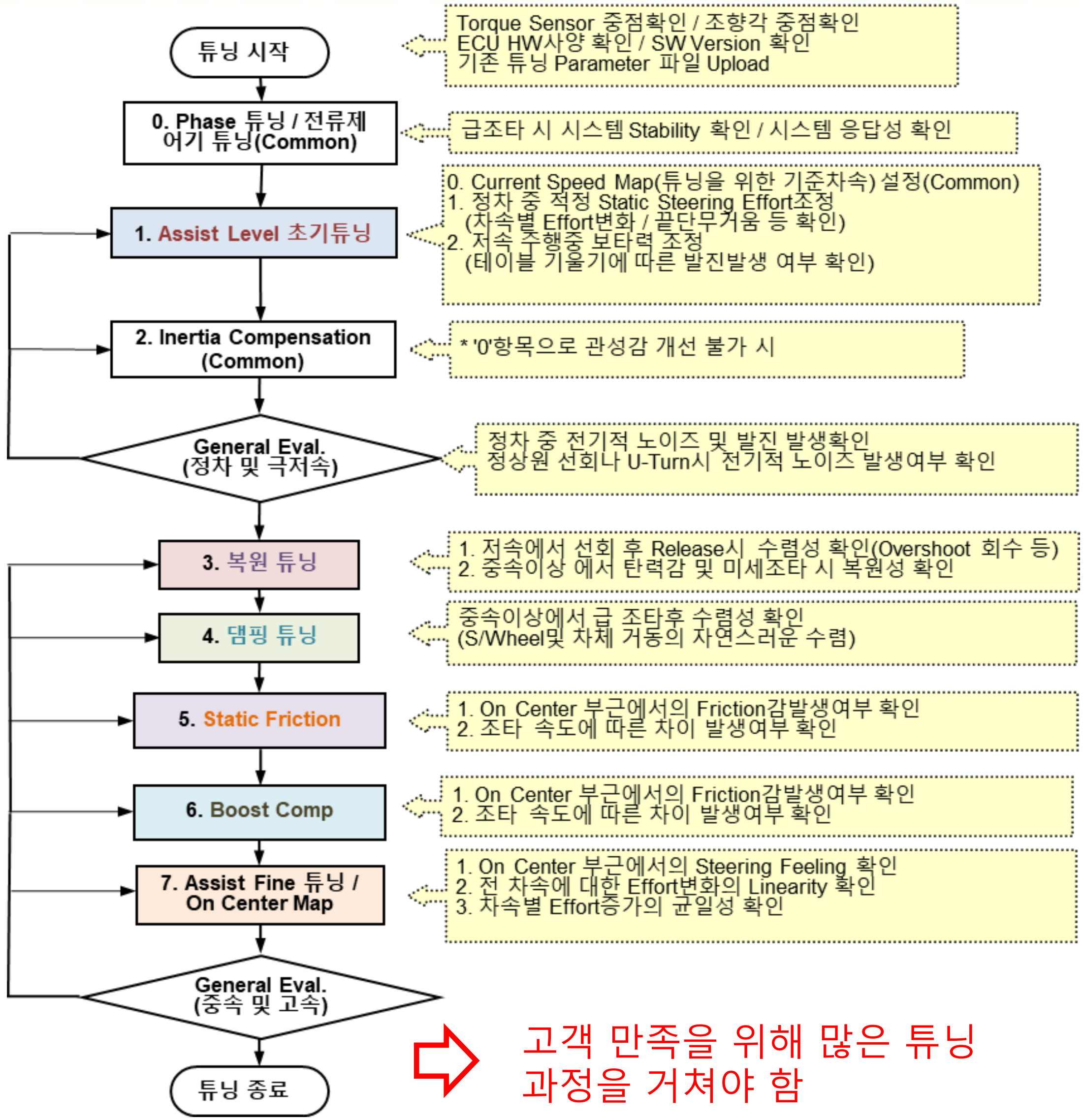
1. 석사 졸업논문 과제 – 문제 정의



<EPS(Electric Power Steering) System>

조타감

좋다
or
나쁘다



고객 만족을 위해 많은 튜닝 과정을 거쳐야 함

1. 석사 졸업논문 과제 – 문제 정의

제약조건 :
1. 실질적인 SW 개발 기간 :
   3~6 Month
2. 개발 인원 : 20명
3. 인당 Project 참여 : 10/인

해결 목표 : OEM 실차 테스트 Tuning시 한번에 Tuning 을 끝내고 싶다.

입력 : 1. Tuning Parameter가 각 기능별 존재
2. 실차 Test시, OEM Engineer와 APP SW engineer가 같이 배석해 튜닝 실시

<수단>

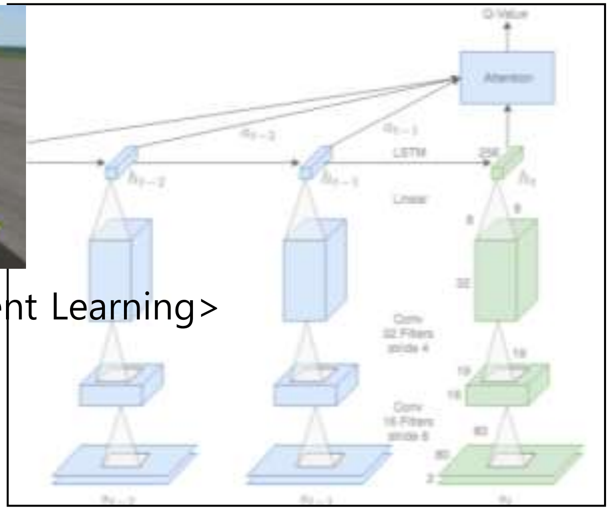
과정 : 실차 Test를 진행하면서, Test 종류 후, APP SW engineer 의 직관과 지식으로 Tuning Parameter를 Case-by-Case로 수정

<활동>

현상황 : 실차 Tuning이 한번만에 끝나지 않고 많은 시간이 걸려 APP SW 엔지니어가 부수적인 시간에 많은 시간이 뺏겨 실질적으로 Application algorithm 개발할 시간이 없다

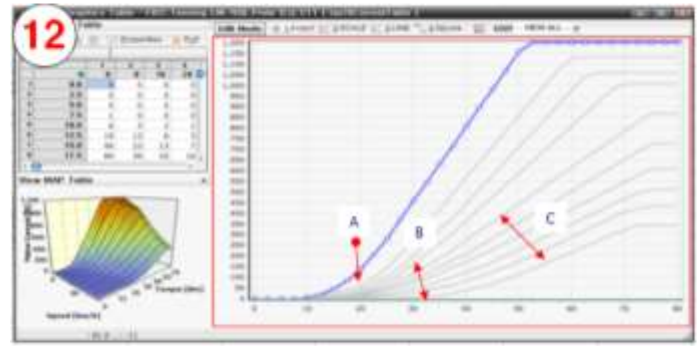
<결과>

1. 석사 졸업논문 과제 – Concept 정의



<Deep Reinforcement Learning>

<튜닝 파라미터
최적 자동화>

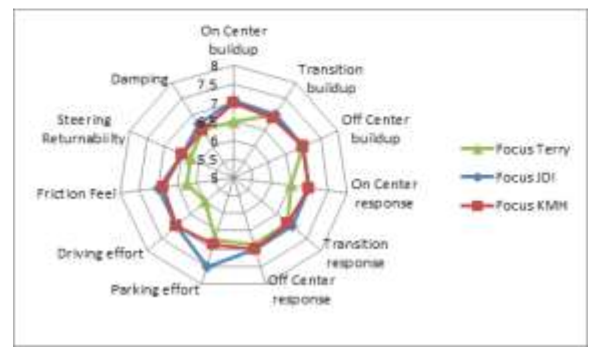


<보상값 선정>

<Big Data 강화학습 알고리즘 적용>

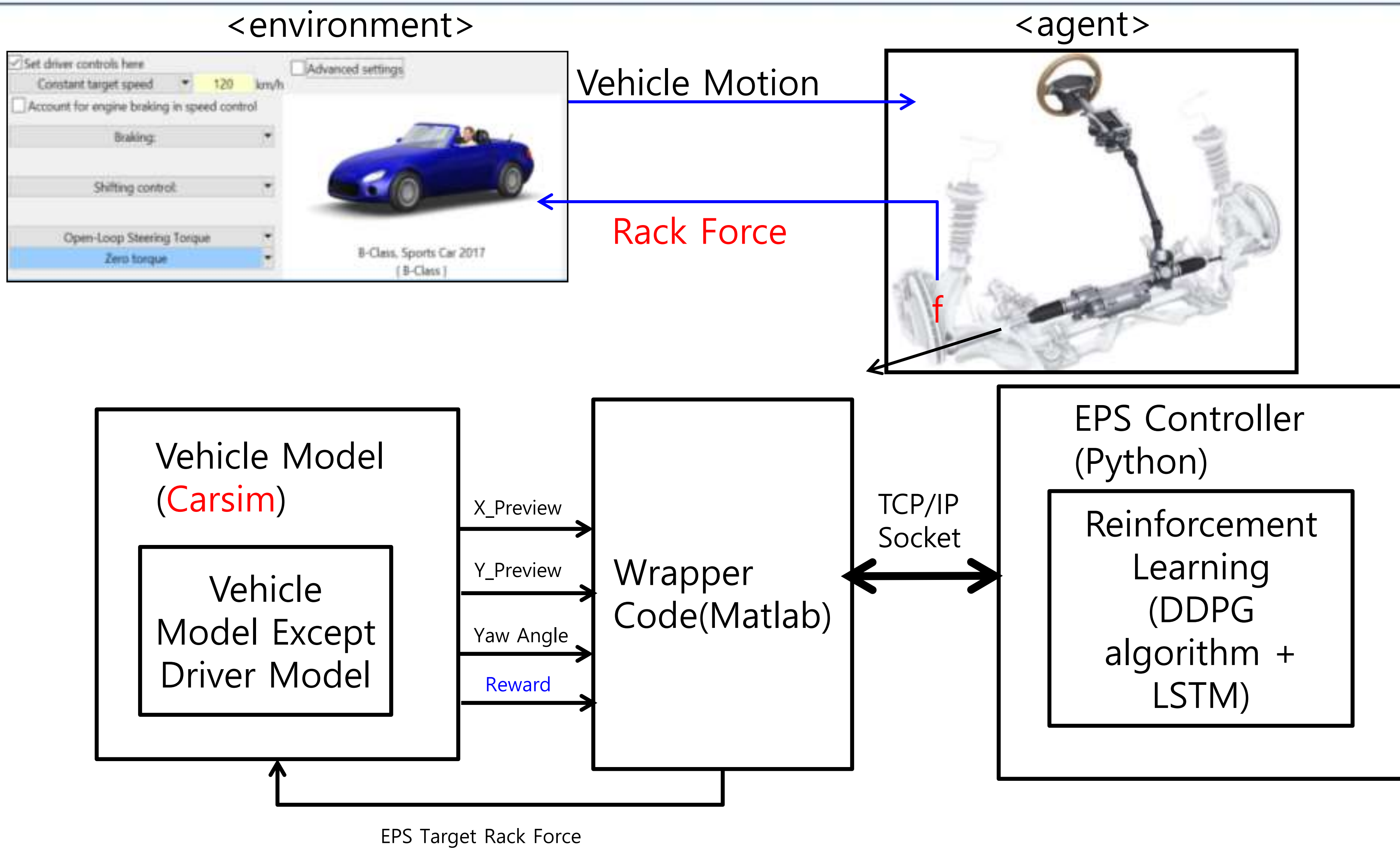
Time	SR Angle	SR Velocity	SR Torque	SR Column Torque	Speed	Lateral velocity	Lateral acceleration
s	°	°/s	Nm	Nm	km/h	m/s	m/s²
0.005	1.631305695	0	0	-0.07	120.5400009	-0.977500021	0.465000004
0.01	1.631644964	-0.200000003	-0.01	0.090000004	120.5299988	-0.975000024	0.485000014
0.015	1.628636718	-0.800000012	0.140000001	0.189999998	120.5299988	-0.975000024	0.485000014
0.02	1.625779629	-0.400000006	0.090000004	-0.02	120.5199966	-0.975000024	0.485000014
0.025	1.6262362	0.300000012	-0.079999998	-0.100000001	120.5199966	-0.975000024	0.215000004
0.03	1.626554608	-0.100000001	-0.150000006	-0.119999997	120.5199966	-0.975000024	0.215000004
0.035	1.62638557	0	-0.119999997	-0.159999996	120.5199966	-0.977500021	0.129999995
0.04	1.627439737	0.200000003	-0.209999993	-0.200000003	120.5100021	-0.977500021	0.129999995

Vehicle Motion BIG DATA
<Input>



Test Engineer Feeling 평가 값
<Output>

Carsim – Python Interface Implementation

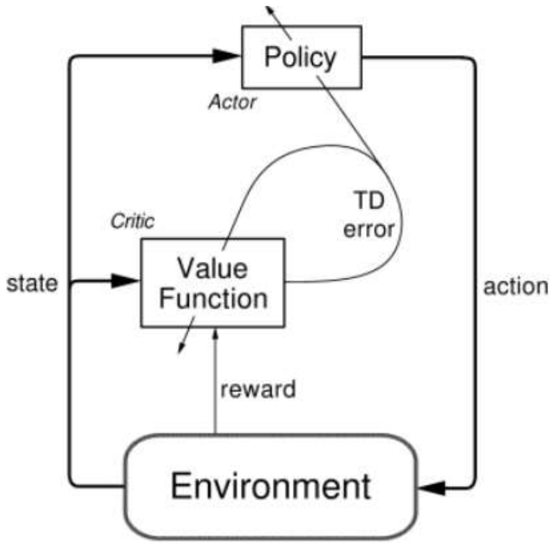


알고리즘 – DDPG (Deep Deterministic Policy Gradient)

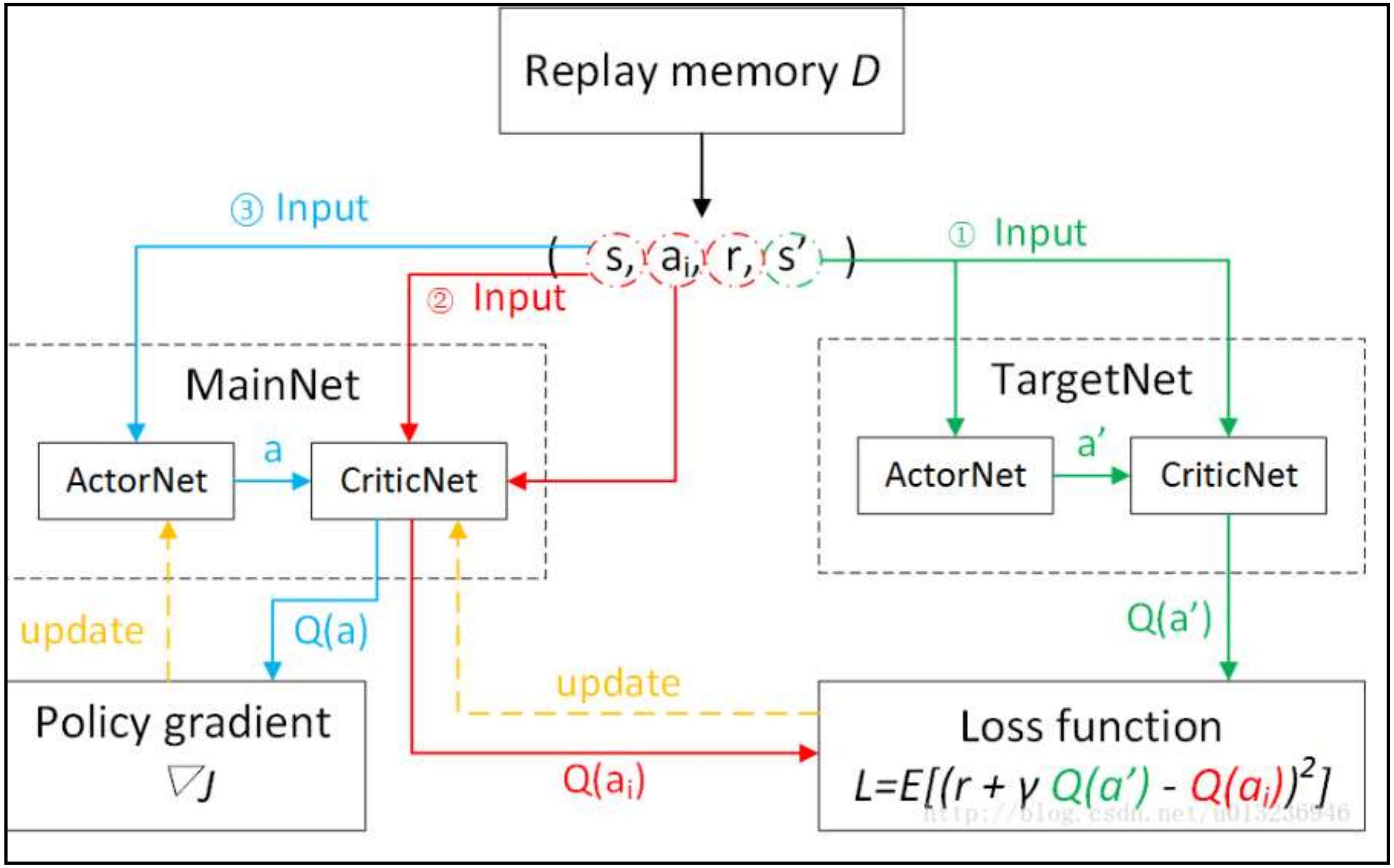
Stochastic Policy → Deterministic Policy (David Silver 2014)

$$\lim_{\sigma \downarrow 0} \nabla_\theta J(\pi_{\mu_\theta,\sigma}) = \nabla_\theta J(\mu_\theta)$$

: parameter 의 편차가 없으면(=0) 확률적 정책이 결정적 정책으로 수행 가능



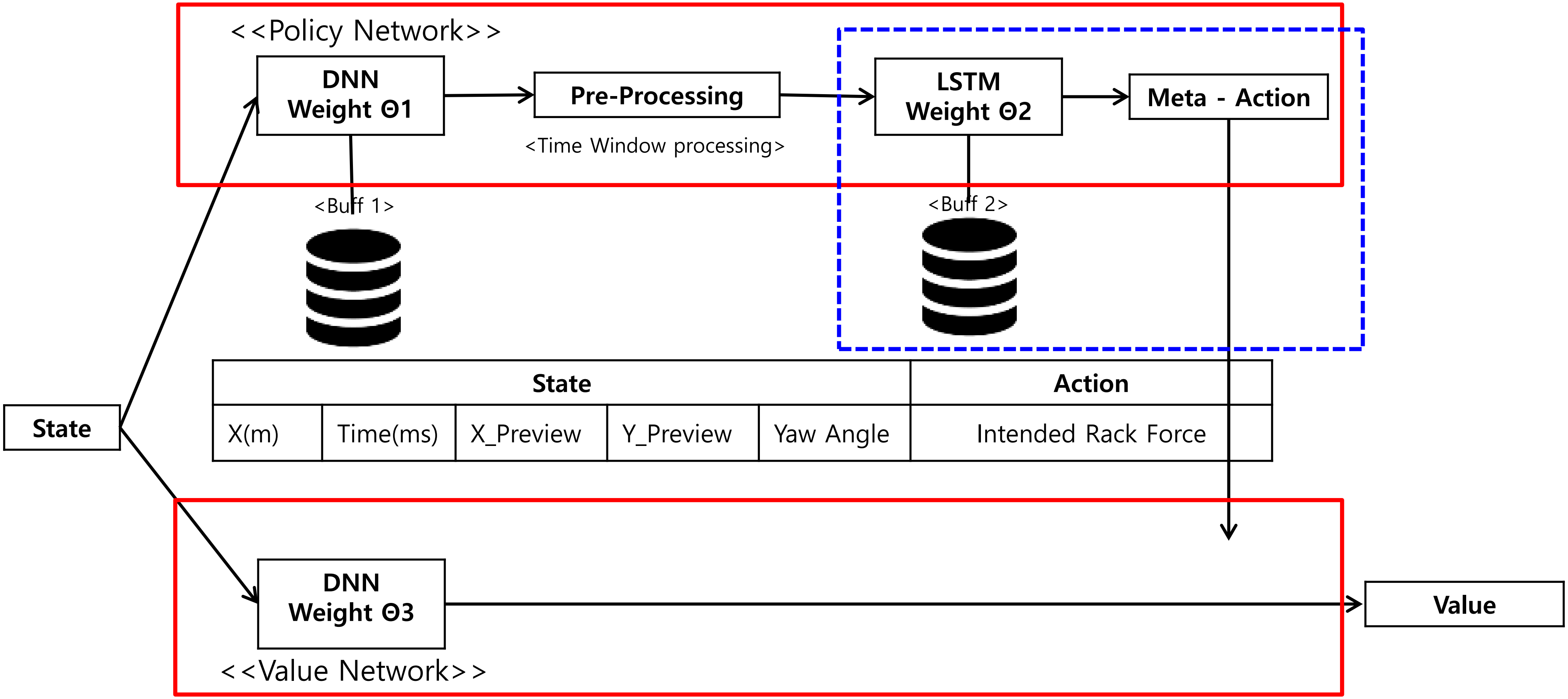
<Actor – Critic>



<DDPG>

알고리즘 – DDPG + RNN(LSTM)

: 추가된 알고리즘



→ Deep Deterministic Policy Gradient 알고리즘에 time step action의 history 기억을 위하여 RNN 알고리즘 수행

알고리즘 – DDPG + RNN(LSTM)

Algorithm : RDDPG

Initialize critic network $Q^{\omega}(a_{h,} s_t)$ and actor $\mu^{\theta 1}(s_t)$, $\mu^{\theta 2}(a_{h-1})$ $with\ parameters\ w\ and\ \theta 1, \theta 2.$
Initialize target networks $Q^{w'}$, $\mu^{\theta 1'}$, $\mu^{\theta 2'}$.
Initialize replay buffer R1,R2.
for episodes = 1, M do
  Initialize action history $a_{h,}$
  for t = 0.0005, T do
    receive state s_t
    select current action a_t from DNN network
    store the $(s_{1,} a_1 \ldots s_t, a_t)$ in R1
    construct history of action
    compute meta action a_h from LSTM network
    select meta action using Orenstein – Uhlenbeck process (exploration)
    store the $(s_{1,} a_{h1}, r_1 \ \ldots \ s_t, \ a_h, r_t)$ in R2
    compute target value for each sample episode $(y^1{}_i \ldots . y^t{}_i)$:
      $y^t{}_i = r_t + \gamma \ Q^{w'} \ (s_t, \ \mu^{\theta 2'} \ (a_h))$
    compute critic by minimizing the loss :
      $L = \frac{1}{N} \sum_i (y^t{}_i - Q^W(a_{h,} s_t))^2$
    Update the current actor policy $\mu^{\theta 1}(s_t)$ using policy gradient
    Update the meta actor policy $\mu^{\theta 2}(a_{h-1})$ using policy gradient
    Update the target networks :
      $\omega' = \tau\omega + (1-\tau)\omega'$
      $\theta' = \tau\theta + (1 - \tau)\theta'$
end for

Predict current action

Pre-processing history of action

Predict final(meta) action

Update critic

Update actor

Update target

Data Structure

State								
X.Y (Location)	Time(ms)	Steering Angle	Steering Tq	Steering Speed	Vehicle Speed	YAWRate	LatG	…
X.Y(1)	0.0005	Ag1	Tq1	AgSpd1	VehSpd1	YAW1	LAT1	
X.Y(2)	0.001	Ag2	Tq2	AgSpd2	VehSpd2	YAW2	LAT2	
…	…	…	…	…	…	…	…	
X.Y(120)	T120	Ag120	Tq120	AgSpd120	VehSpd120	YAW120	LAT120	

Action
Steering Rack Force (N)
R_Force_t1
R_Force_t2
…
R_Force_t120

$$Reward(t) = V_x \cos(Yaw) - V_x \sin(Yaw) - V_x\ Lateral\ distance\ - V_x abs(LatG\ - 0.5),$$

Reward

차량 종가속도 추종

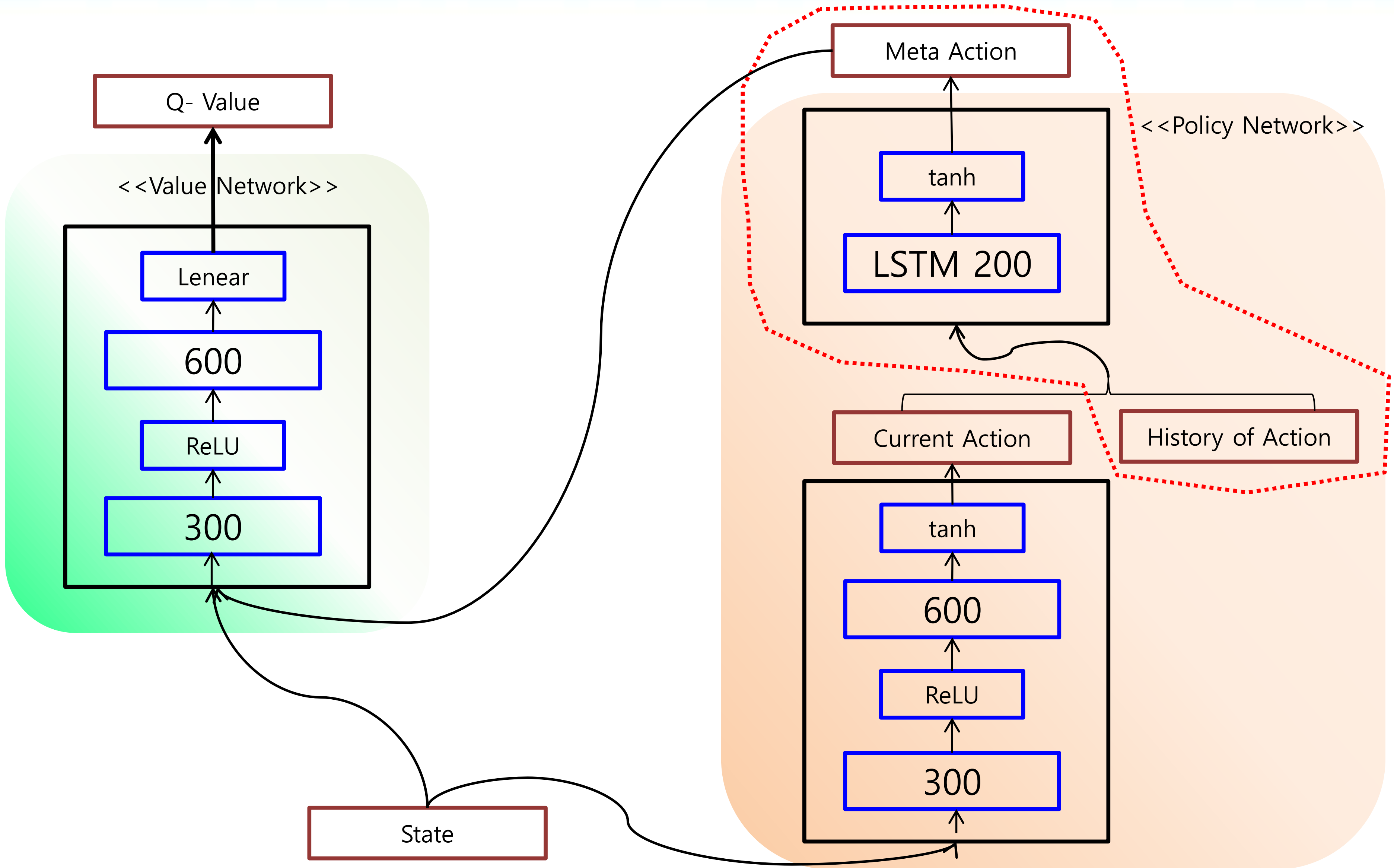
Reward(t) = Vx*cos(Yaw)
- Vx*sin(Yaw) - Vx*Lateral_Distance
- Vx* |Lat_Acc - 0.5|

차량 도로 path 추종

운전자 조향 안정감

Network architecture

: 추가된 알고리즘



Exploration algorithm

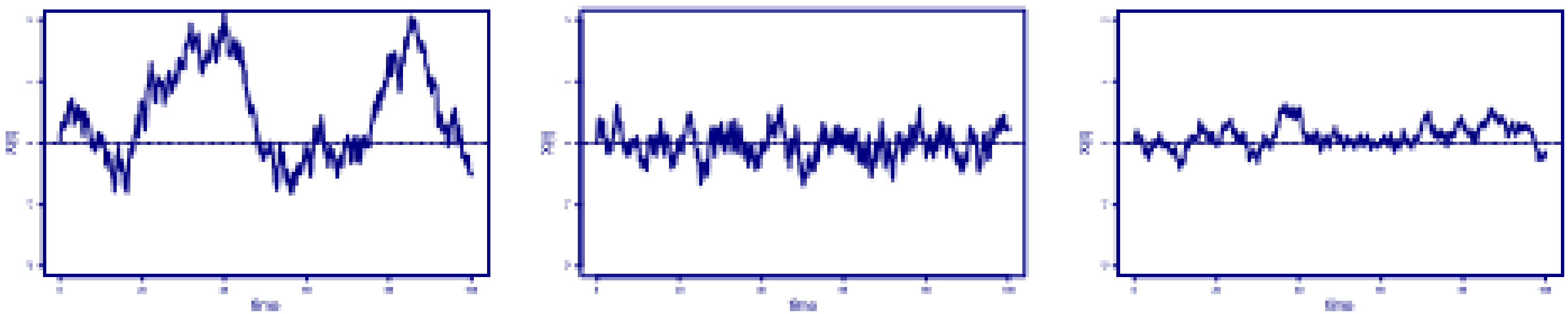
Exploration : the Ornstein-Uhlenbeck process
: 브라운 운동을 확률적으로 모사한 확률 프로세스. steering car motion에 적합하여 선정.

- Gaussian noise with moments

$$dX_t = -\beta(X_t - \alpha)dt + \sigma dW_t$$

β : decay rate or growth-rate
σ : variation or size of the noise
α : mean

≪ β ↓ σ ↑ : 값의 변동폭이 심함 > >

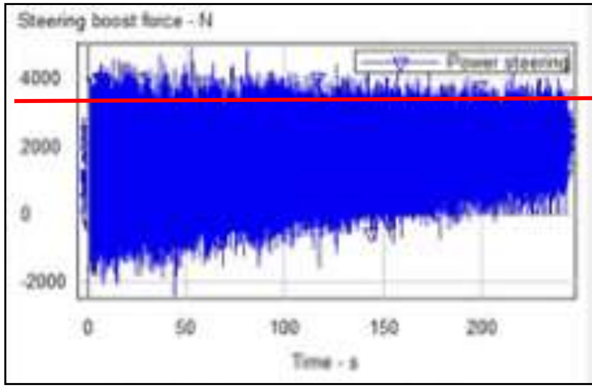


$\beta = 0.01, \sigma = 1$ $\beta = 0.1, \sigma = 1$ $\beta = 0.01, \sigma = 0.5$

Critic gradient inverting algorithm

Continuous action space의 action saturation 문제

: 연속된 action을 다룰 때, 쉽게 maximum action value boundary에 근접하여, 학습이 local optimization에 빠짐




```
s_t= [-0.60090699  0.29522567  0.11704854 ...,
0.09523867 -0.17911712
  1.00571467]
a_t_original= [[ 0.99993801]]
```

Action value 값이 쉽게 포화됨

→ Critic gradient inverting algorithm 기법 필요

- Critic gradient inverting algorithm (Matthew Hausknecht : ICLR 2016)

$$\nabla_p = \nabla_p * \begin{cases} p_{max} - p \;/\; p_{max} - p_{min} & : \text{ if } \nabla_p \text{ suggests increasing p} \\ p - p_{min} \;/\; p_{max} - p_{min} & : \text{ otherwise} \end{cases}$$

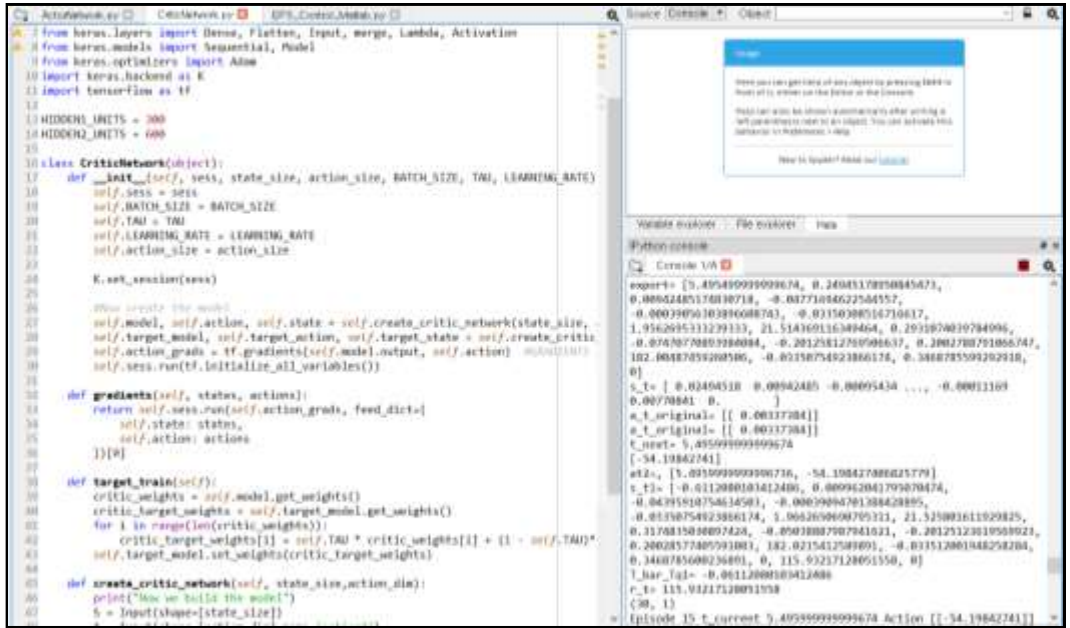
∇_p: critic gradient

$[p_{max}, p_{min}]$: boundary of action

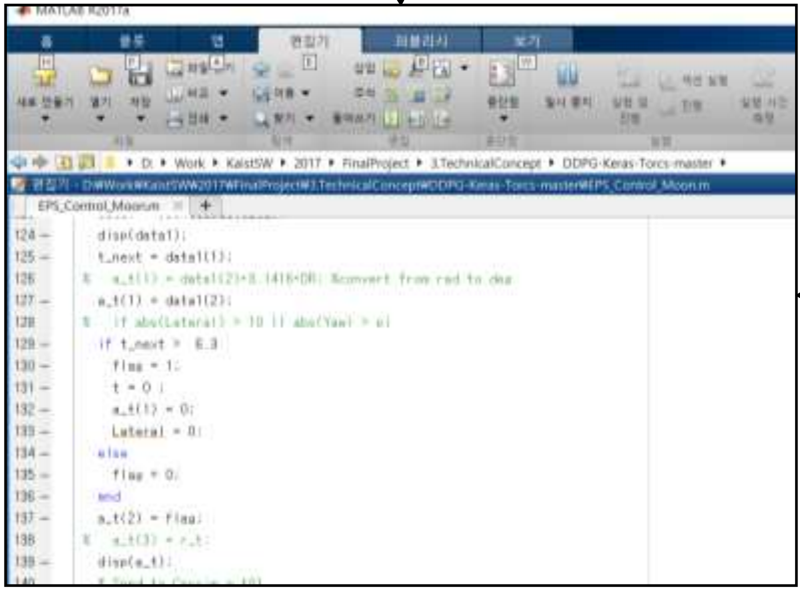
p : action value

Implementation

<ENV – Carsim>



<RL Machine – Python>



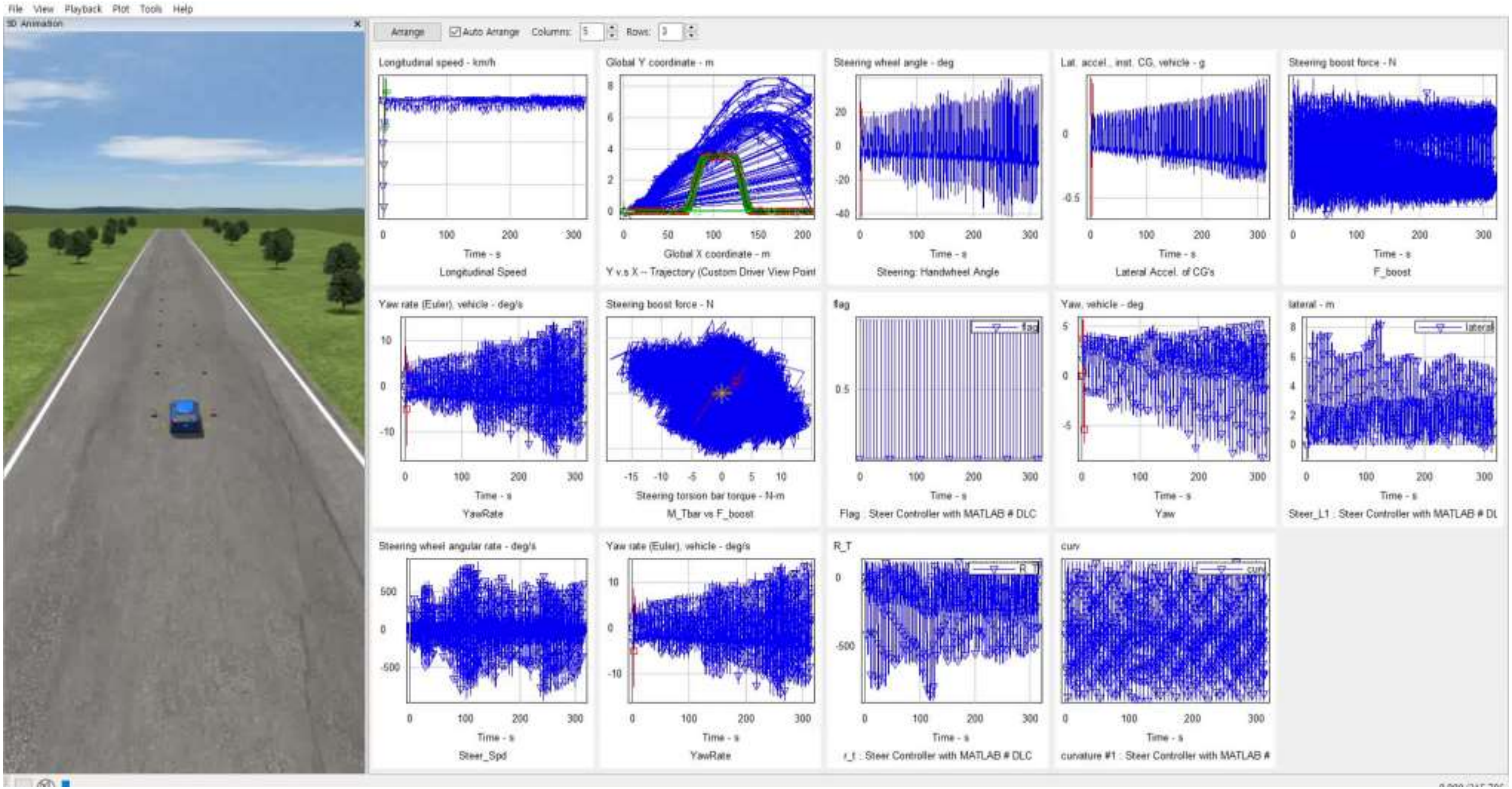
<Interface code – Matlab>

Implementation – tuning

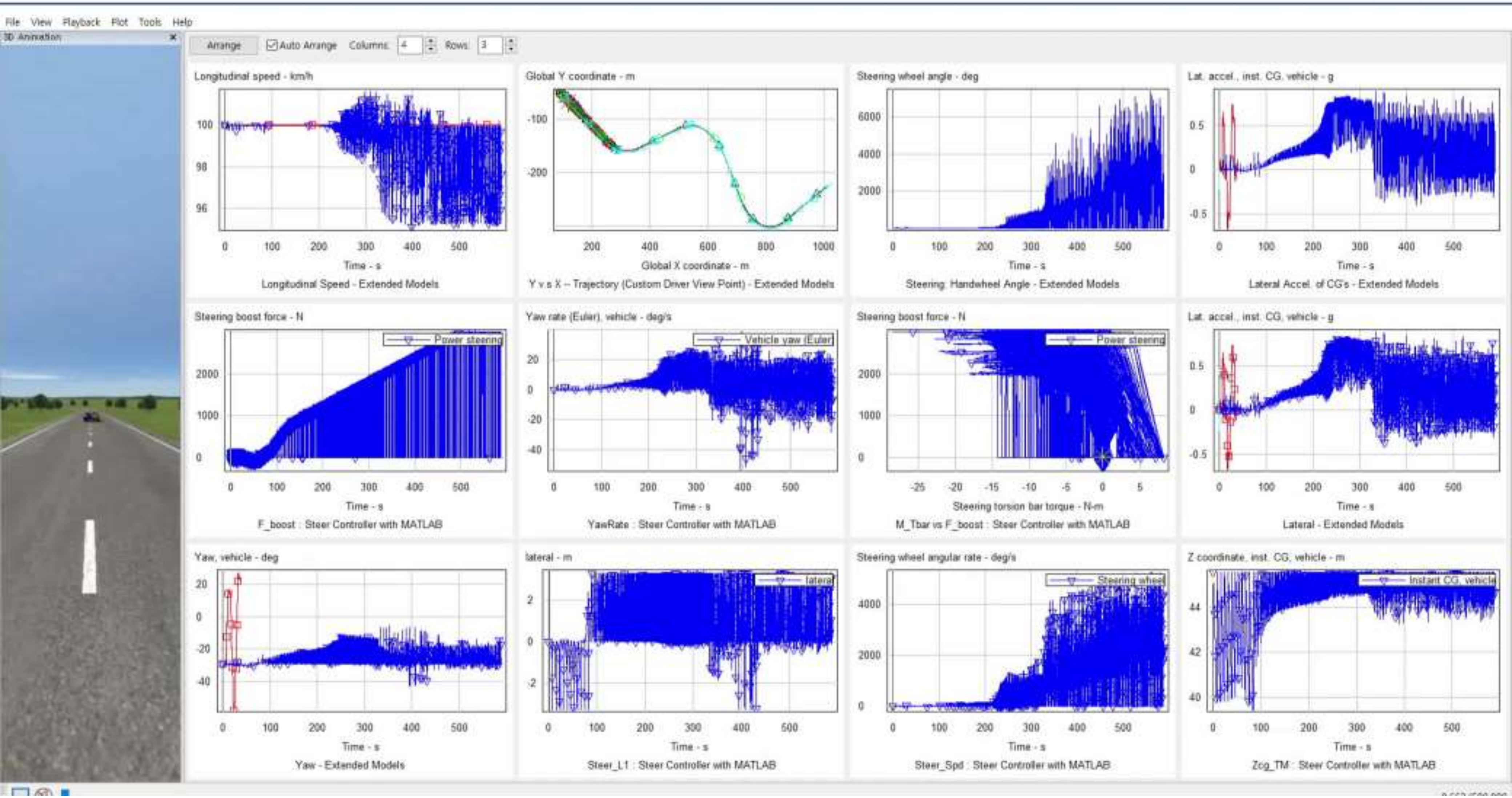
- Important stuff (출처 github_cgel) :

1. Normalize input [0, 1] : ✓ Done
   → Input feature / $Value_{max}$
2. Clip reward [0, 1] : ✓ Done
   → reward term / $term\ value_{max}$
   → Inverting gradient
3. Don't use tf.reduce_mean the loss in the batch. Use tf.reduce_max : ✓ Done
   → use tf.reduce_max
4. Initialize properly the network with xavier initialize. : ✓ Done
   → use initializer glorot_uniform
5. Use the optimizer that the paper uses. : ✓ Done
   → try to use various optimizer (Adam/SGD , etc)
6. Don't use various constraints for reset (action saturation) : ✓ Done
   → use only time step constraints

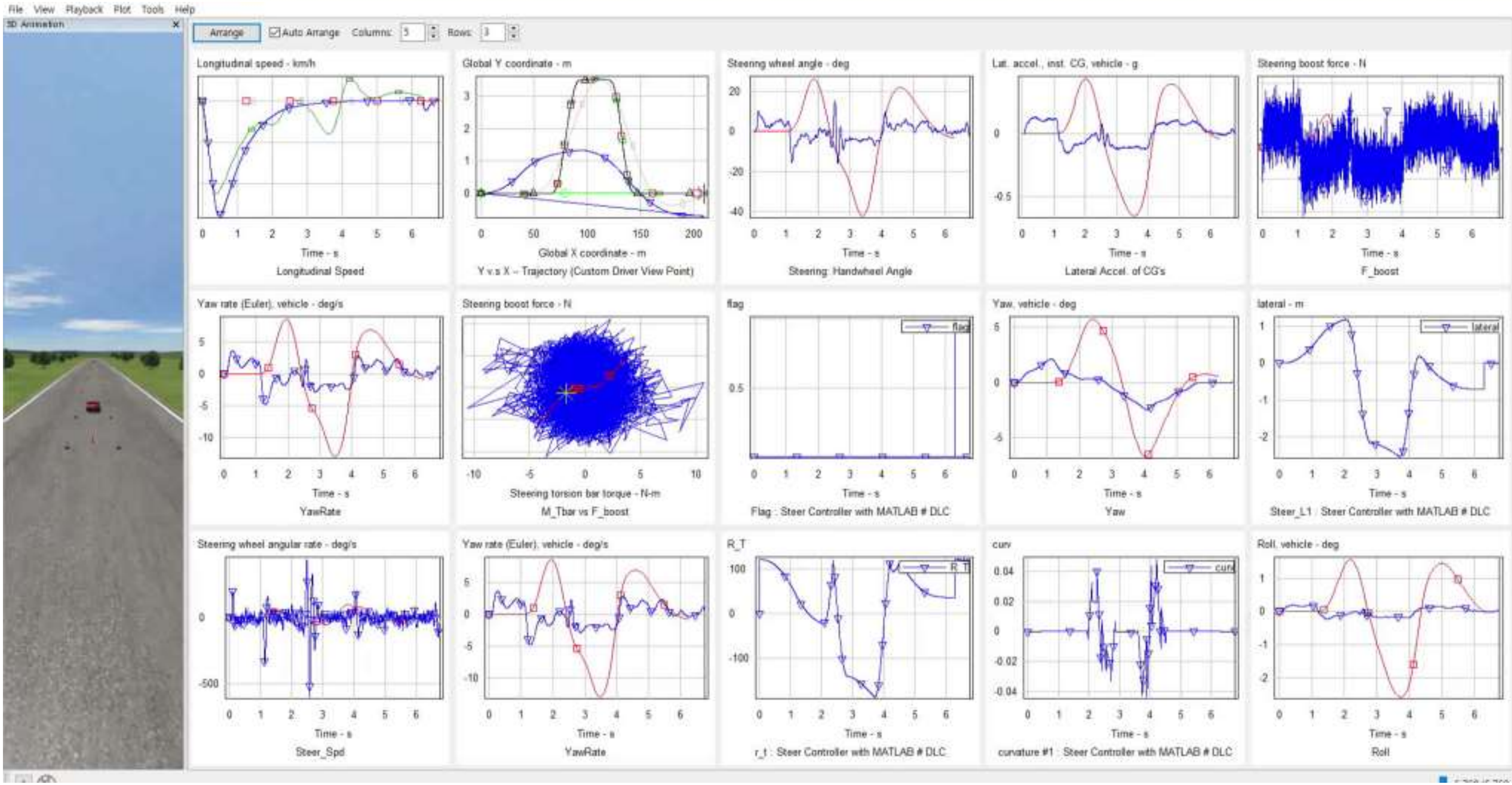
Implementation (training)



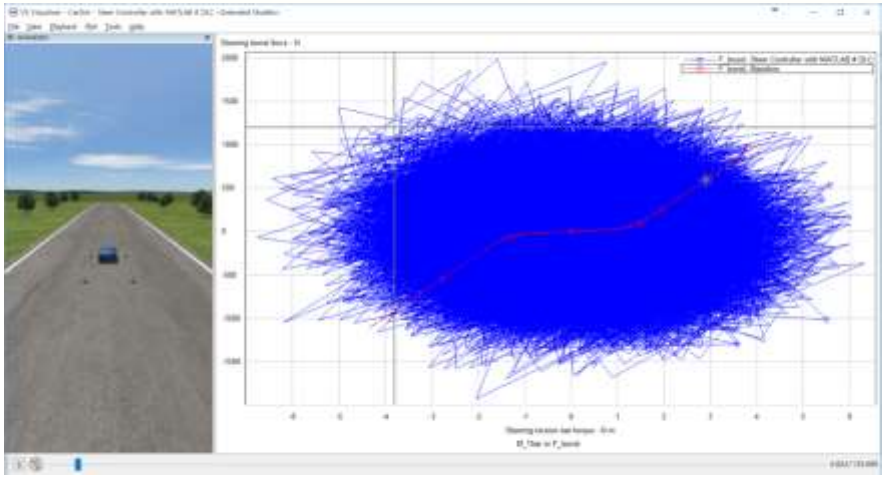
Implementation (training)

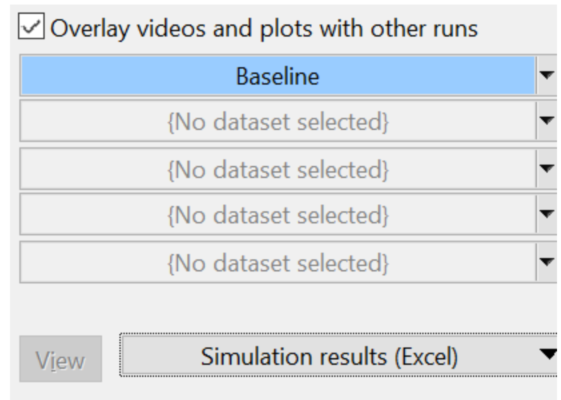


Implementation (trained _ EP 100)



Implementation (Tuning Parameter 추출)

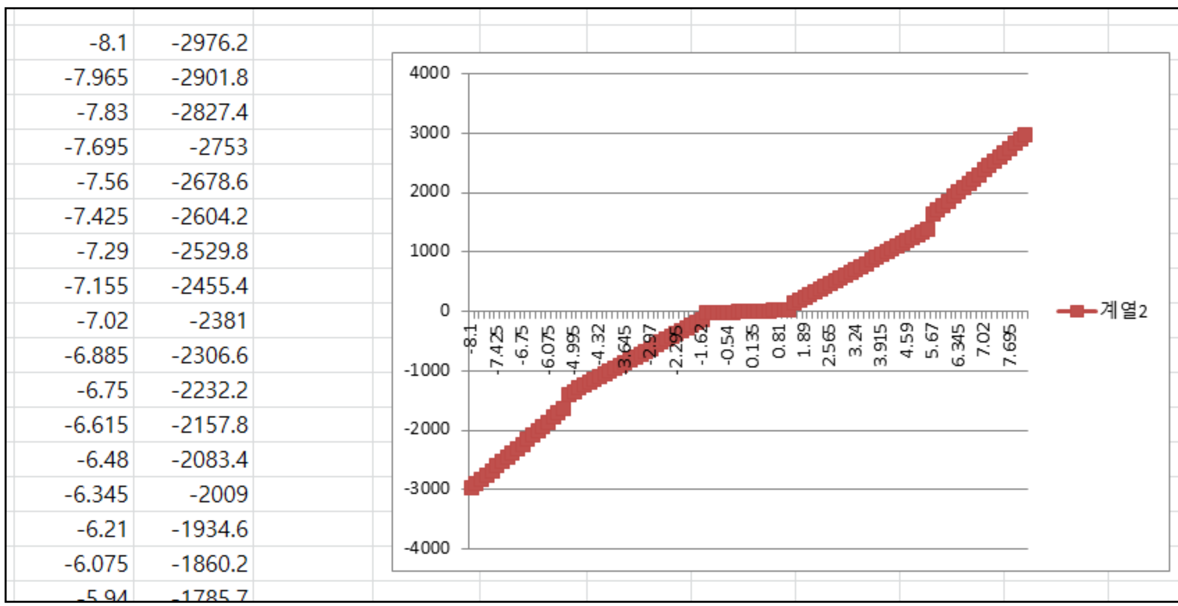




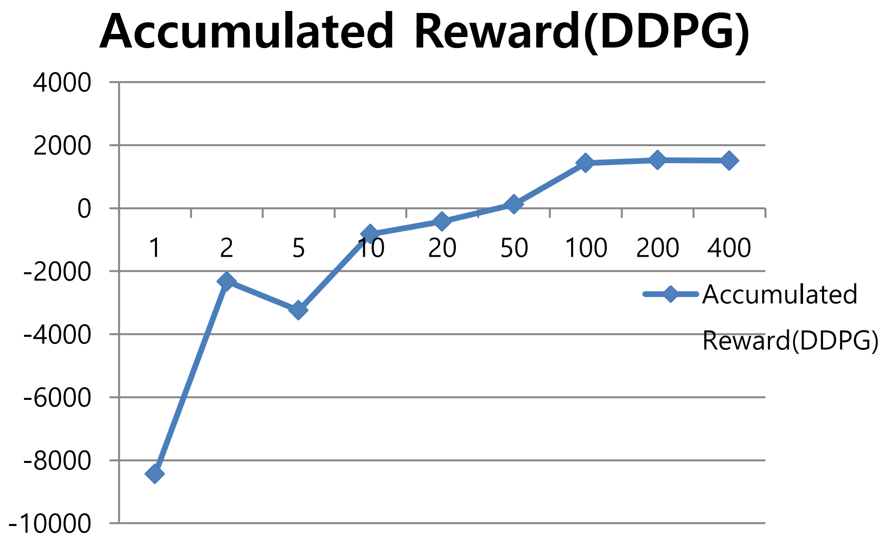
<<Data 추출>>

<<signal processing>>

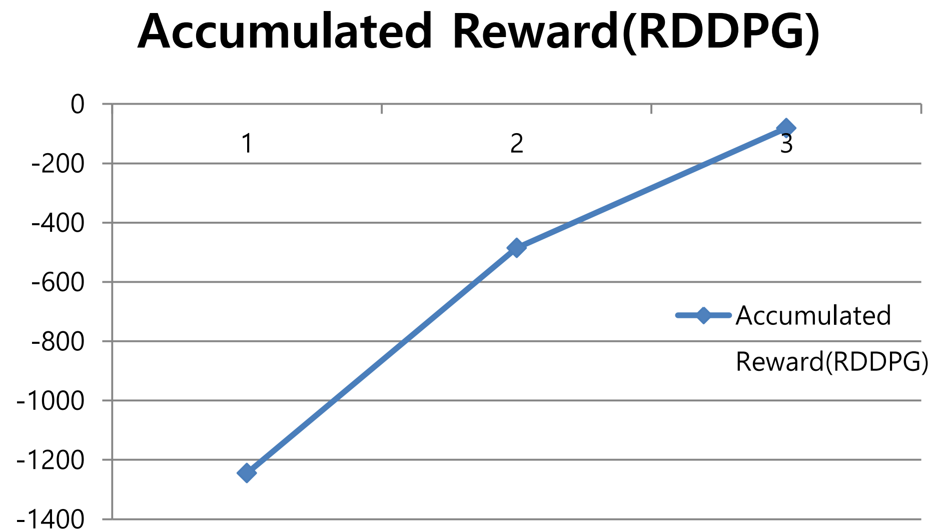
<<Steering Tuning Parameter 지표화>>



Results



<DDPG algorithm>



<RNN-DDPG algorithm>

RDDPG : Hyper parameter tuning 및 학습 시간 더 필요 (future work!)

Contribution

Academic

- 기존 DDPG 알고리즘에 action에 메타러닝 알고리즘을 추가
  → 초기 학습 속도 ↑
  → 유사한 문제에 메타 모델 weight 사용가능 (모델 재사용성 증가)
  → action의 history를 기억하고 있기 때문에 action saturation 방지효과

- **But,** RDDPG 알고리즘의 tuning 과 더 많은 학습 시도가 필요

Industry

- 상용 Tool(carsim)에 steering tuner를 자동화
  → 초기 실차 튜닝 없이 pre-tuning 가능
  → 초기 code 구현 후, 튜닝 과정을 simulation가능
  → 엔지니어링 Effort 감소

→엔지니어 3인 1달 절감 효과 기대 (2400만원/프로젝트 기대)

- **But,** 더 많은 Test case 개발로 실차와 가까운 환경 구성
  네트워크 최적화 및 Computing power 보강으로 학습 시간 단축 필요
  (EP1(약 10초) 학습 시, 약 6시간 소요)

Future Work

- Future Work
  - 실제 차량 데이터를 근거로 한 상용 car simulator와 RL machine 실제 구현

<Double Lane Change>

<Circle Road>

<Rolling Over Stability>

<Sine Dwell>

<u>Total reward</u> 학습으로
steer comfort & stability 확보

- Open Issue
  → 기존 DDPG 알고리즘의 경우에도 일정 EP 경과 시(EP>150),
  action saturation 문제 해결 필요
  → reward clipping 과 critic network inverting gradient로 일부 해결했지만
  최적의 학습 상태는 아님

  → RDDPG 알고리즘의 경우, memory resource 할당 및 학습 속도 최적화 필요
  (현재 10초(time step = 0.0005) 시뮬레이션 약 6시간 소요)

느낀점

- frankly speaking ...

  \- 실제 차량 Test를 하면서 Tuning 하는 것보다 RL machine parameter tuning 시 초기 man power 투여가 더 많이 소요 되는 것 같다.
  → 더 많은 학습 및 know-how 확보로 pre-trained model 확보 필요

  \- 생각보다 학습이 잘 안 된다.
  EP 150 초과 후에는 action 값이 포화 되어 더 이상 학습이 진행 되지 않음
  → 현재 몇 가지 기법과 논문에 소개된 알고리즘을 도입하였으나, 실제로 현업에 쓰이기 위해서는 더 많은 노력이 필요

  \- 그럼에도 불구하고,
  실제 차량 car simulator와 RL agent를 구현함으로써, 학습 모델만 구축 된다면 많은 man power 와 시간 절감(약 한달/인당) 효과가 있을 것으로 예측.
  상용화된 RL Steering Tuner를 만들고 싶습니다.