AGENT MCQUEEN

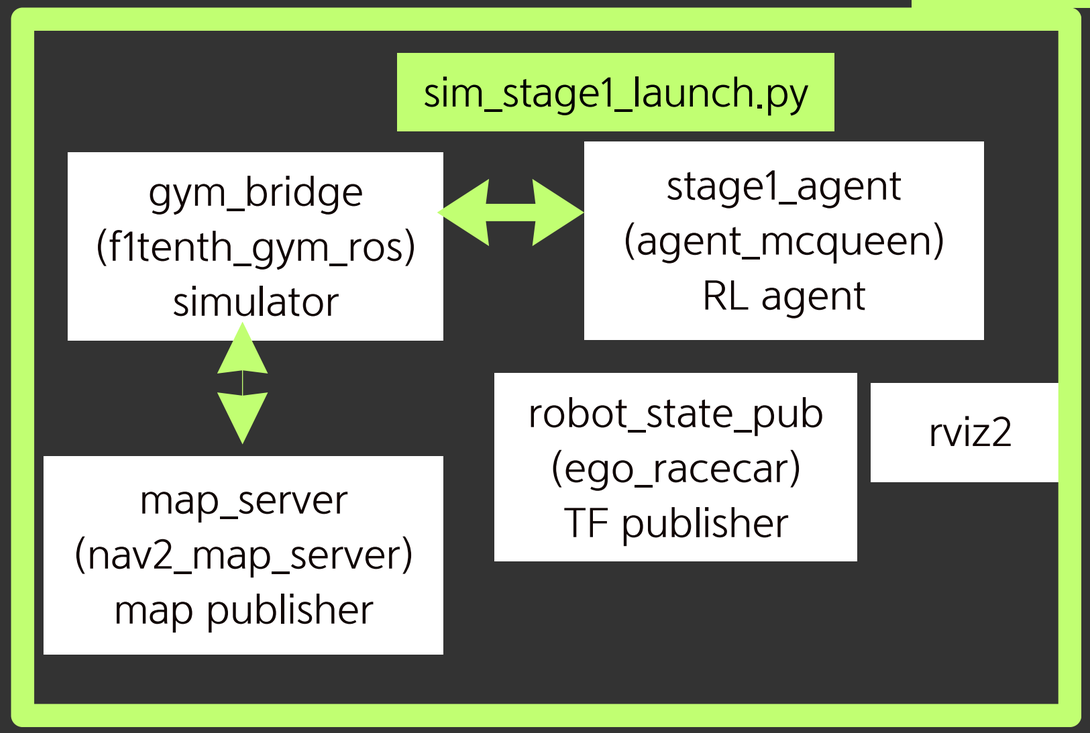
:residual learning with frozen agent

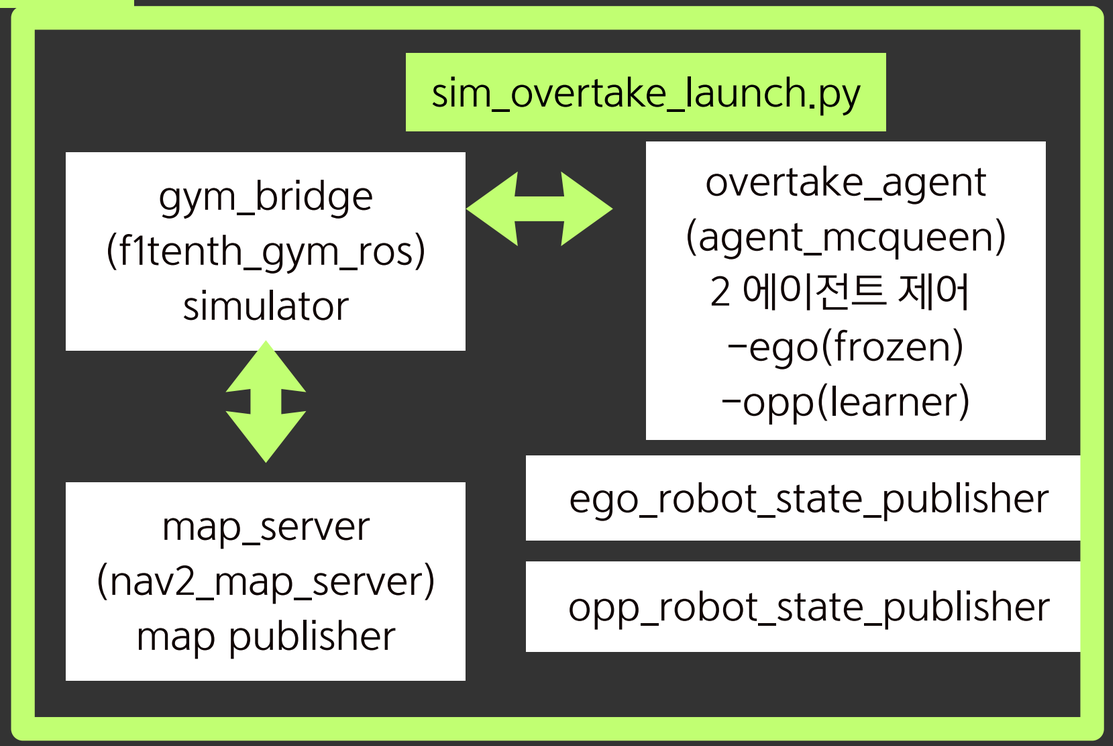
숭실대 ai융합학부 20211539 이솔규

SUMMARY

- stage1: ppo 로 트랙을 주행하는(평가에서 23개의 f1 트랙 들 중 21개의 트랙들에서 완주 성공) 모델 학습
- stage2:residual learning 으로 head-to-head 상황에서 적극적으로 추월 시도하는 모델 학습
- 이를 f1tenth gym 환경에서 렌더링함.
- f1tenth gym 환경에서 학습시킨 두 개의 학습 렌더링을 ros 2 python 패키지로 작성 후 f1tenth , forzaETH racetrack 연동 완료

node graph





F1tenth 시뮬레이터(연동 완료)
gym_bridge → stage1_agent/ overtake_agent
/sim/scan
/sim/ego_racecar/odom
/sim/opp_scan
stage1 agent/ overtake_agent → gym_bridge
/sim/drive
/sim/opp_drive

ForzaETH 시뮬레이터(연동 완료)
gym_bridge → stage1_agent/ overtake_agent
/scan
/car_state/odom_GT
/opp_scan
stage1 agent/ overtake_agent → gym_bridge
/drive
/opp_drive

3

AgentMcqueen	강화학습 PPO (Proximal Policy Optimization) + Self Play 방식은 신선하나, ForzaETH 스택 내부의 어떤 정보를 어떻게 이용해서 강화학습을 시킬지에 대한 구체성 부족

- 도커 환경에서 f1tenth_ws 내부에서 ros 브릿지로 바로 학습을 진행하려 했으나, 세팅에 어려움을 겪고, 학습이 느리다는 문제 발생 → 도커 환경이 아닌 호스트 머신에서 f1tenth gym 환경에서 강화학습 진행 후, 모델을 로드하여 사용. 1- ppo 로 트랙 완주하는 모델 2- residual learning으로 적극적인 추월을 시도하도록 학습된 모델 → 2 개의 모델을 사용
- 2번째 슬라이드에서의 forzaETH, f1tenth 의 노드 그래프와 하단의 메세지 흐름을 통해 내부 정보를 어떻게 사용하였는지 기술함. stage1_agent_node.py,

- mappo, self play 등의 기법을 사용하였으나, non-stationary 문제로 head-to-head 레이싱 상황에서 두 에이전트의 주행 능력을 상실되는 문제를 겪음(stage1 에서의 주행 능력을 지녔음에도) 문제 해결과 학습의 안정성을 위해 한 에이전트를 동결시키고, 다른 에이전트에 대해서 학습을 진행, 이때 여전히 non-stationary 문제가 발생하여, 상대 에이전트의 delta_s , delta_vs, ahead(상대의 거리, 상대의 속도, 앞/뒤 플래그) 을 observation으로 얻게 하고, 나머지 actor network 부분들(lidar_net,mean_head,log_std_head)은 동결을 시킨 상태에서 상대방의 정보를 처리하는 부분만 학습을 시키는 residual learning으로 문제 개선.

stage1_agent_node(내부)

input(subscribe)
- scan_topic
- odom_topic

processing
1. lidar 전처리: [0,1] 로 정규화
2. 속도 전처리: velocity/3.2 로 정규화
3. framestack: 4 frame 유지-> shape(4,1081): beam 1080 개와 velocity 1 이라 1081
4. ppo 모델 추론(stable_baseline3)

output(publish)
- drive_topic
  - steering_angle: float(rad)
  - speed: flaot(m/s)

→ stage1 의 모델 로드

overtake_agent_node(내부)

input(subscribe)
- ego_scan_topic
- opp_scan_topic
- ego_odom_topic
- opp_odom_topic

processing
- Agent 0(ego-frozen expert)
  - obs_dim:4324(1080+1)*4 frames
    - lidar scan(1080)/velocity(1)
  - actor_1(Actor Network)
  - Frozen-no learning
- Agent 1(oppo-learner)
  - obs_dim:4336(1080+1+3)*4 frames
    - lidar scan(1080)/velocity(1)/delta_s(1)/delta_vs(1)/ahead(1)
  - actor_2(Actor Network+WithOpponentInfo)
  - trainable-residual

문제 정의

레이싱을 즐겨 보시나요? 최근 F1 관련 영화가 큰 인기를 얻어 한국에서도 F1, 르망24, WRC 와 같은 에싱에 관심이 높아졌습니다. 하지만 유럽이나 미국, 일본 등에서 레이싱은 이미 큰 인기를 얻고 있는 스포츠죠. 많은 사람들이 이러한 레이싱에 열광하는 이유는 뭘까요?

제가 생각하기에는 최고의 자동차 기업들의 강력한 머신, 피트스탑 시간이나 여기에서의 협업 능력 등 다양한 매력적인 요소들이 있지만 그중에서 레이서들 간의 치열한 경쟁이라 생각합니다. 레이서 들 간 펼치는 심리전, 추월, 이를 방어하는 전략 등 이러한 DYNAMIC 레이싱이 핵심이라 생각하는대요. 저희가 지난 학기 동안 배운 알고리즘들로는 이러한 주행을 하기에는 제약이 있습니다.

아이디어 제안

TRIAL AND ERROR 로 학습을 하는 강화학습이라면 이러한 제어 알고리즘이 가진 수동적인 주행에서 벗어나 능동적인 레이싱을 학습하여 dynamic한 주행을 할 수 있지 않을까 라는 가정을 세웠습니다.

어떻게 이를 학습시켜야하나 에 대해 고민을 했습니다. 보통 강화학습에선 하나의 task 를 학습을 시킵니다. pick-up 이면 비전 모델로 이미 물체를 인식할 수 있는 상태에서 물체를 집는 걸 학습하고, 휴머노이드가 걷는 것이면 걷는 것을 학습시킵니다. 하지만 racing은 벽에 충돌하지 않고 주행하는 것은 물론이고, 상대와의 경쟁도 학습해야 합니다. 여러 가지 것들을 학습해야할 뿐 아니라 한 에이전트가 아닌 2개의 에이전트를 학습시키는 head-to-head 상황이기에 multi agent reinforcement learning을 이용해야 합니다. 그렇기에 racing 이라는 task 를 MARL 로 학습시키는 것은 현재의 저에게 있어 매우 어렵다는 판단을 내렸습니다.

그렇기에 2 가지의 stage 로 나누어 진행해보자는 생각을 하였습니다.
- -stage1: 하나의 에이전트가 충돌 없이 주행을 하는 것을 학습시킨다.
- -stage2: stage2 에서 학습시킨 모델을 2 에이전트에게 로드, actor network에 상대 에이전트의 정보를 처리하며 학습하는 부분들(opponent_net,opponent_adjustment_net)을 제외한 나머지 부분들은 동결시킨 상태에서 학습을 시켜, 적극적인 추월을 시도하도록 학습시킨다.

이 프로젝트는 결국 만든 것을 f1tenth , forzaETH와 연동을 해야했기에 f1tenth_ws 에서 바로 학습까지 시켜 모든 것을 진행하려고 하였으나, 앞서 언급한 문제점들로 f1tenth_gym 환경에서 진행하기로 하였습니다.

PPO

Proximal Policy Optimization, PPO 알고리즘은 로보틱스에서 굉장히 흔히 쓰이는 알고리즘입니다. 그만큼 단순하면서 강력한 알고리즘입니다. ppo는 우선 actor-critic 구조와 on-policy를 사용합니다. actor 와 critic 이라는 두 가지 네트워크를 사용하며, 말 그래도 배우가 연기를 하면 비평가가 이것이 좋은 연기인지 아닌지를 평가하며 더 좋은 연기를 하도록 학습을 진행하게 됩니다.

policy 는 강화학습에서 에이전트의 뇌라고 생각할 수 있습니다. 이때 이는 목표하고자 하는 target policy와 현재 정책인 behavior policy로 나뉘는데, on-policy는 이 두 개가 동일한 경우를 뜻합니다. 반면 off-policy는 target policy와 behavior policy가 다른 경우로, DQN처럼 experience replay buffer의 과거 경험을 재사용합니다. Sample efficiency는 좋지만 과거 데이터와 현재 policy 간 괴리가 발생할 수 있습니다. On-policy가 로보틱스 연속 제어에서 선호되는 이유는 on-policy는 actor network가 직접 연속적인 action distribution(보통 Gaussian)을 출력하므로 연속 제어에 자연스럽게 적합하기 때문입니다.

PPO의 전신인 TRPO는 KL divergence constraint로 policy 변화를 제한했지만 구현이 복잡했습니다. PPO는 이를 단순한 clipping으로 대체합니다. 새 policy와 이전 policy의 확률 비율을 $r_t(\theta)=\pi_\theta(a_t \mid s_t)/\pi_{\theta old}(a_t \mid s_t)$를 $[1-\varepsilon, 1+\varepsilon]$ 범위로 제한하여, advantage가 양수인 좋은 action에서 ratio가 너무 커지거나, 음수인 나쁜 action에서 ratio가 너무 작아지면 gradient를 차단합니다. 이 clipping 덕분에 on-policy임에도 수집된 데이터를 여러 epoch 재사용할 수 있어 sample efficiency와 stability를 균형 있게 달성하며, 구현 단순성, 하이퍼파라미터 robustness, 안정적 학습으로 로보틱스에서 사실상 표준으로 자리잡았습니다.

θ (theta): Policy network의 파라미터
ε (epsilon): Clipping 범위를 결정하는 하이퍼파라미터
a: action
s: state

stage 1

진행 과정

HTTPS://GITHUB.COM/MERACCOS/F1TENTH_REINFORCEMENT_LEARNING?TAB=README-OV-FILE

- 한 github 의 프로젝트를 참고하여 개발 → 이 프로젝트는 맵의 초기화 부분이 미완성이라 완성시키고, f1tenth, forza 에 연동하기 위해 환경이 달라 이에 용이하게끔 깃헙을 참고하여 프로젝트 개발.
- ppo(proximal policy optimization) 으로 학습
- 450개의 트랙과 centerline들을 생성하고 매 에피소드마다 랜덤으로 이들 중 하나를 골라 학습을 진행
- agent 의 pose 와 가장 가까운 centerline way point 를 찾고, 이를 프레넷 좌표로 변환 후 s,d를 기반으로 보상 계산 → 이때 에이전트는 자신의 보상 구조에 대해 모름. 하지만 학습하며, centerline 에 가깝게 달릴 수록 보상이 커진다는 경향성을 학습하게 됨.
- 일반화를 위해 domain randomizaiton을 통해 무작위로 생성된 고정된 장애물들에 충돌하지 않도록 학습을 진행하였습니다.

- 보상 설계

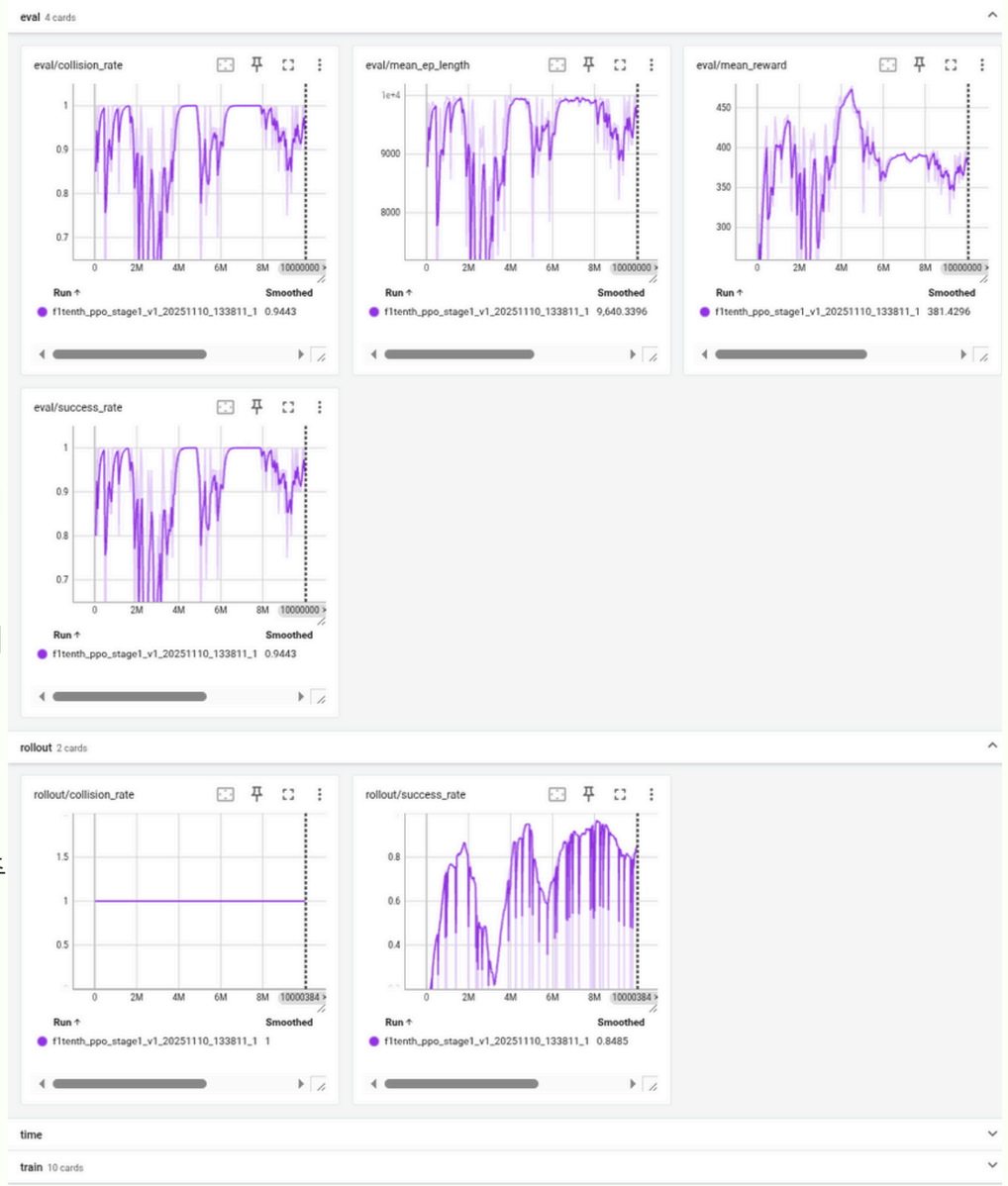
positive rewards
-Survival bonus: +0.01 (매 스텝)
-Forward progress(vs) : +1.0 × vs (트랙 따라 전진 속도)

negative rewards
-Stop penalty: −2.0(속도 ≤ 0.25 m/s일 때)
-Lateral drift (vd): −0.01 × |vd| (횡방향 드리프트)
-Centerline distance: −0.05 × |d| (중앙선에서 벗어남)
-Angular velocity: −0.05 × |w| (회전 속도)
-Wall proximity: −0.01 × (0.5 − min_dist) (벽에 가까움)
-Collision: −1000.0 (충돌 시 에피소드 종료)

Total Reward = 0.01 + 1.0×vs − penalties
→ stage1 에서의 핵심 목표는 충돌 없이 완전한 주행이었기에 충돌에 −1000 이라는 매우 큰 페널티를 줌.

성과

- 천만 스텝을 학습시켰고 약 13시간 소요 → gpu: RTX4090
- 평가는 F1tenth_racetracks 23개의 트랙들에서 진행하였으며, 21개의 트랙에서 성공적으로 주행(90%)



stage1 의 tensorboard 그래프
rollout: 학습 중 / eval: 1000스텝마다 평가
이때 success_rate은 완주율, collision_rate은 충돌율

해결책 도출

1. 처음 시도에서 학습이 제대로 이루어지지 않았음 → 이 당시에는 원인을 알지 못해, 450개의 트랙들 중에서 랜덤을 골라 학습을 한다는 점에서 어려움이 있나 라는 생각을 하여 curriculum learning 컨셉을 도입하여 200만 스텝까지는 100개의 트랙에서, 600만 스텝까지는 250개의 트랙에서 천만 스텝까지는 450개의 트랙들에서 랜덤으로 골라 학습을 하게함. 하지만 핵심적인 문제는 맵의 초기화에 제대로 이루어지지 않아 학습이 유효하지 않았음. 추후 원인을 고치고, 학습을 성공적으로 시킬 수 있었는데, 확실히 우측의 남색 그래프를 보면 학습이 앞 장의 그래프보다 안정적으로 우상향하는 현상을 관찰할 수 있음. 하지만 커리큘럼 러닝이나 없이 하는 학습이나 최종적으로 가지는 성능은 비슷했기에 더 단순한 ppo만을 채택하기로 함.

- 맵의 초기화가 어떻게 문제였나?
  - 매 에피소드마다 랜덤한 트랙에서 학습을 진행하는 것이 기본 메커니즘인데, 이때 당연히 각 랜덤한 맵의 centerline 데이터도 로드가 돼야 하지만, 이 로직이 없었음. 학습 후 수많은 맵들을 직접 렌더링을 해보니, 특정 50^{th} 맵은 완벽하게 주행을 하는데, 나머지 맵들에선 전혀 주행을 못함. 이 부분이 굉장히 수상하여, 맨 처음 골라진 맵을 확인해보니 50^{th} 였음. Random_seed 를 42로 고정하여 학습을 진행하였기에, 이 때 로드된 centerline 이 계속 유지돼있었구나는 문제점을 찾았을 수 있었고, 해결함. → 학습이 온전히 가능하게 됨.

남색: curriculum learning 학습 그래프
하늘색은 고려 x

stage 2

진행 과정

- 두 에이전트에 stage 1에서의 모델들을 로드함. 한 에이전트(ego)를 동결시키고, 다른 에이전트(opp)에 대해서만 추가학습을 진행함.
- 이럼에도 opp이 ego에 대한 대처를 전혀 하지 못하는 non-stationary(이는 stage2 해결책 도출 part에서 보다 자세히 다룸) 문제로 주행 능력을 상실하는 문제 발생
- opp가 ego가 장애물이 아닌 상대 레이서라는 인식을 주기 위해 actor network에 opponent_net,opponent_adjustment_net 을 추가하고, 이 부분들만 학습을 하게 하고, 나머지 부분들에 대해선 동결시켜 학습을 진행.
- 보상 체계도 stage1 과는 다르게 진행. 특히 이미 주행은 잘하기에 적극적인 추월과 주행 능력 상실을 막기 위해 forward progress에 더 많은 보상과 추월 보상, 승리 보상을 추가함

성과

- 천만 스텝을 학습시켰고 약 13시간 소요 → gpu: RTX4090
- 평가는 F1tenth_racetracks 23개의 트랙들에서 진행하였으며, 21개의 트랙에서 성공적으로 주행(90%)

- 보상 설계

positive rewards
– Forward progress (vs) : +10.0 × vs ← Stage 1 대비 10배!

negative rewards
–Stop penalty: –2.0(속도 ≤ 0.25 m/s)
–Reverse penalty:–10.0(vs < 0, 후진 방지)
–Centerline distance: –0.05 × |d|
–Lateral velocity: –0.01 × |vd|
–Angular velocity: –2.0 × |w| ← Stage 1 대비 40배!
–Wall collision: –5.0

–Overtaking bonus : +500.0 (Agent 0를 추월했을 때, 1회)
–Win bonus: +1000.0 (먼저 1랩 완주했을 때)

추월 감지: prev_relative_position > 0 → relative_position < 0(Agent 0이 앞에 있다가 → Agent 1이 앞으로 갔을 때)

→ 이렇게 설계한 이유는 이미 주행은 가능한 모델이기에 굳이 충돌에 집착할 이유가 없음. 주 목표인 추월을 위한 주행을 학습하는 것이었기에 이렇게 세팅함.

stage 2-actor network

Agent 0(ego)

- **feature_net-frozen**
- **mean_head-frozen**
- **log_std_head-frozen**
- **optimizer=None**

Agent 1(opp)

- **lidar_net-frozen**
- **opponent_net**
- **opponent_adjustment_net**
- **mean_head-frozen**
- **log_std_head-frozen**
- **optimizer=Adam (for just not frozen parts)**

왜 다른 actor network 구조를 가지고 있는가에 대한 답변입니다.

- 우선 Agent 0의 경우 stage 1 과 동일한 구조를 하고 있습니다. 반면 Agent 1의 경우 stage 1 에서의 주행 능력을 유지하면서 (이 부분은 동결) 상대 에이전트에 대한 정보를 처리하며 학습을 하는 부분을 추가해야 했습니다. 그렇기에 기존의 feature_net 을 lidar_net 과 opponent 정보를 처리하는 opponent_net,opponent_adjustment_net 를 나눴어야 했기에 다른 구조를 지니게 됐습니다.

해결책 도출

뛰어난 Stage 1 solo racing 모델을 기반으로 Stage 2 head-to-head racing으로 넘어갔습니다.

First Try: MAPPO

첫 번째 시도로 Multi-Agent PPO(MAPPO)를 채택했습니다. MARL(Multi-Agent Reinforcement Learning)에서는 단일 에이전트 RL과 달리 고유한 챌린지들이 존재합니다:

1. Credit Assignment: 팀 보상에서 각 에이전트의 기여도를 어떻게 분배할 것인가
2. Non-stationarity: 다른 에이전트도 학습하므로 환경이 계속 변화함 → 이때 에이전트는 observation만으로 환경을 인지하기에 이런 문제가 발생함
3. Selection of Equilibrium: 여러 균형점 중 어떤 균형으로 수렴할 것인가 → MARL은 여러 에이전트들을 사용하기에, 게임 이론에 근거함.
4. Scaling: 에이전트 수 증가에 따른 복잡도 폭발

에이전트가 1에서 2로만 늘어나도 이러한 문제들이 발생하며, 이 프로젝트에서는 특히 2)와 3)이 핵심 문제였습니다.

- Critic Loss 발산: Selection of Equilibrium 문제

MAPPO의 구조에 대한 충분한 분석 없이 학습을 진행한 결과, 학습이 진행될수록 두 에이전트의 주행 능력이 상실되고 critic loss가 발산하는 현상을 발견했습니다. 원인은 보상 구조의 부적합이었습니다. MAPPO는 기본적으로 누적 보상 합을 최대화하는 구조로, 한 에이전트의 reward가 다른 에이전트에게도 간접적으로 긍정적 영향을 주는 cooperative 세팅을 가정합니다. 그러나 racing은 본질적으로 competitive 세팅입니다. 승자가 있으면 패자가 있고, 이는 게임이론에서 zero-sum 구조에 해당합니다. Cooperative 알고리즘으로 competitive 문제를 풀려 했으니 균형점 자체가 불안정해질 수밖에 없었습니다.

- 주행 능력 상실: Non-stationarity 문제

각 에이전트는 자신의 observation만으로 환경을 인지합니다. 이때 상대 에이전트는 단순히 "장애물"로 인식되는데, 이 장애물이 움직이므로 에이전트 입장에서는 환경 자체가 계속 변하는 것처럼 보입니다. Stage 1에서 학습한 policy는 정적 환경을 가정했기에, 움직이는 장애물이 등장하자 학습 안정성이 저해되고 결국 기존 주행 능력마저 상실하게 되었습니다.

Second Try: 모델 동결

두 번째 시도로 한 에이전트를 완전히 동결시키고 다른 에이전트만 학습시키는 방식을 적용했습니다. 이렇게 하면 최소한 환경의 변화가 "학습하는 상대"에서 "고정된 상대"로 줄어들어 non-stationarity가 완화될 것이라 기대했습니다. 그러나 여전히 주행 능력 상실 문제가 발생했습니다.

문제를 다시 생각해보니, 근본적인 결함이 보였습니다. 현재 에이전트는 LiDAR로 주변을 스캔할 뿐, 그것이 벽인지 상대 차량인지 구분하지 못합니다. 실제 레이싱에서 드라이버는 상대가 누구인지, 어디에 있는지, 얼마나 빠른지를 파악하고 전략을 세웁니다. 하지만 현재 에이전트는 레이싱을 하면서 상대의 존재조차 명시적으로 인지하지 못하는 상태였습니다.

Solution: Opponent-Aware Observation

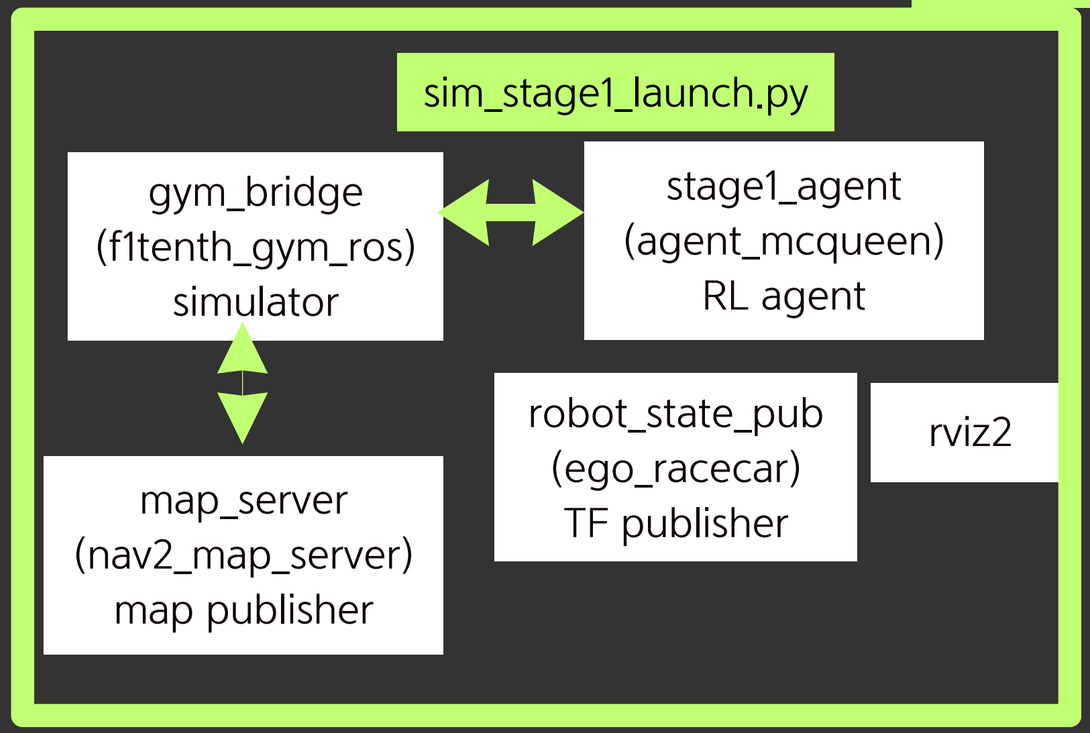
해결책으로 학습 대상 에이전트의 observation에 상대방 정보를 명시적으로 추가했습니다:

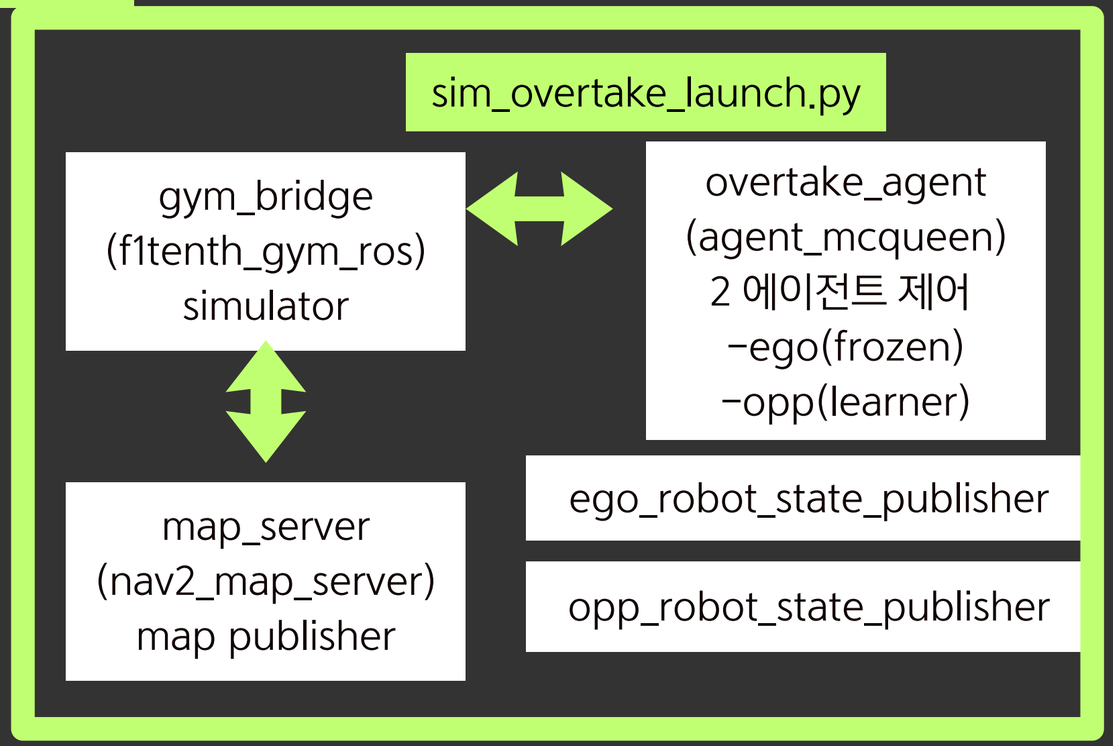
- delta_s: 상대와의 track 상 거리 (progress 차이), delta_v: 상대와의 상대 속도, ahead: 상대가 앞에 있는지 뒤에 있는지를 나타내는 binary flag

이 값들은 tanh로 스케일링하여 [-1, 1] 범위로 정규화했습니다. 결과적으로 에이전트는 주행 능력을 유지하면서 더 적극적인 추월 행동을 학습할 수 있게 되었습니다. 상대의 존재를 "알게" 해주는 것만으로도 non-stationarity 문제가 크게 완화된 것입니다.

참고문헌: Stefano V. Albrecht Filippos Christianos Lukas Schäfer, MULTI-AGENT REINFORCEMENT LEARNING FOUNDATIONS AND MODERN APPROACHES(2024)

node graph





F1tenth 시뮬레이터(연동 완료)
gym_bridge → stage1_agent/ overtake_agent
/sim/scan
/sim/ego_racecar/odom
/sim/opp_scan
stage1 agent/ overtake_agent → gym_bridge
/sim/drive
/sim/opp_drive

ForzaETH 시뮬레이터(연동 완료)
gym_bridge → stage1_agent/ overtake_agent
/scan
/car_state/odom_GT
/opp_scan
stage1 agent/ overtake_agent → gym_bridge
/drive
/opp_drive

현재 AGENT MCQUEEN의 한계

- 우선 stage2 에 있어서 제대로 된 평가를 진행하지 않았습니다. stage2 에서의 학습을 전혀 하지 않았을 때, 같은 모델 2개를 로드하고, 에이전트 0의 속력을 80%로 맞추고, 렌더링을 해보니, 아무 추월에 관한 학습을 하지 않았음에도 추월을 하는 모습을 관찰했었습니다. 그리고 적극적인 추월을 시도하는 모델로 학습 후에는 보다 더 적극적인 추월을 시도하기 했지만, 정확히 얼마나 이것이 개선됐는지 실험을 하지 못한 부분이 아쉬운 것 같습니다.

- stage1에서 학습된 같은 모델을 로드하여 stage 2 를 진행하다 보니, 두 에이전트의 기본적인 전략이 같습니다. 전략의 다양성이 없기에 색다른 변수 창출을 하지 못해, 경쟁적인 레이싱을 하는데에 제약이 있습니다.

- stage1 이후에 centerline 이 아닌 레이싱의 최적 경로로 파인튜닝을 하면 어땠을까 라는 아쉬움이 남습니다. 또한 f1 레이서의 다큐멘터리를 보니, f1 레이서들은 자신이 경주할 트랙을 눈을 감고도 주행할 수 있도록 트랙에 대해 공부한다고 합니다. 반면 Agent Mcqueen 은 매번 다른 맵들을 주행하다 보니, 레이싱의 주목적에서 약간 벗어나, 탐험에 가까운 레이싱을 하는 거 같습니다.

- 사실 레이싱이라는 게 훨씬 복잡합니다. 물리적인 변수들은 물론이고, 최고 속력까지 얼마나 빠르게 도달하는지, brake 를 얼마나 잡아야 하는지 코너를 돌 때 어떻게 진입할지 등 디테일한 요소들이 많고, 이들의 목적은 최소 랩타임이 주목적일 수 있지만, 대부분 이런 변수들을 이용해 어떻게 상대 레이서를 추월할 수 있는지, 추월을 방어할 수 있는지에 대한 가능성을 높이는 전략적인 변수이기도 합니다. 그러나 이러한 디테일한 부분들은 모두 고려한 상태에서 하기에 아직까지는 제 능력에 한계가 있었습니다. 제어 알고리즘들을 결합하고, 의사결정에 강화학습을 활용하기까지 하였다면 더 경쟁적인 레이싱을 가능했겠다는 생각도 듭니다.

THANK YOU