벤처스타트업 성과공유 페스티벌

자율주행기반 머신러닝

AWS DeepRacer

대회 리뷰

한국외국어대학교

일해라절해라

김수연, 이은진, 홍석준

Contents

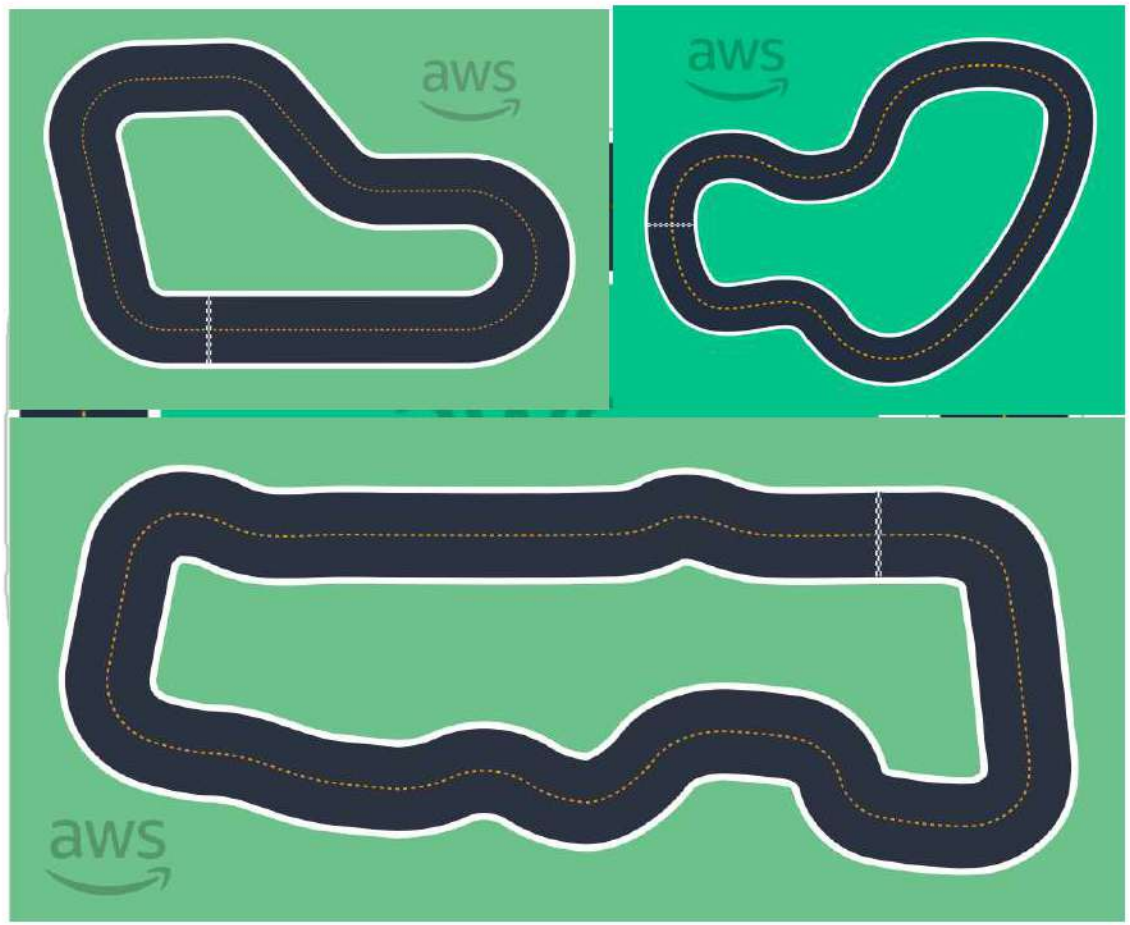
Deep Racer란 무엇인가요?

AWS DeepRacer

자율주행 차량을 활용해
강화학습(Reinforcement Learning)
알고리즘을 실습할 수 있는 플랫폼

- **Observation**: 카메라로 트랙 이미지를 받아 상태로 변환
- **Action**: 강화학습을 통해 학습한 행동(조향각 및 속도) 선택
- **Reward**: 보상 함수를 기반으로 주행 성능을 평가하고 학습

프로젝트 개요



개발 배경

Oval Track 에서 진행

가장 단순하면서도 에이전트가
다양한 전략을 학습할 수 있는 공간 제공

Time trial 방식의 레이스

구간을 가능한 한 **빠르게 주파**하는 것을
목표로 하는 레이스


AWS DEEPRACER
주최
중소벤처기업부
KOVA
(사)벤처기업협회
한국외국어대학교
제주대학교
협력
aws
Amazon Web Services
수행
iCORE
iCORE E&C

작품 구성 및 개발내용

Action space type
Discrete

Action space

No.	Steering angle (°)	Speed (m/s)
0	-30.0	1.60
1	-24.0	1.80
2	-16.0	2.00
3	-8.0	2.00
4	0.0	2.00
5	8.0	2.00
6	16.0	2.00
7	24.0	1.80
8	30.0	1.60

Framework
Tensorflow

Reinforcement learning algorithm
PPO

Hyperparameter	Value
Gradient descent batch size	64
Entropy	0.01
Discount factor	0.9
Loss type	Huber
Learning rate	0.0003
Number of experience episodes between each policy-updating iteration	20
Number of epochs	10

01. Action space 설정

Discrete action space

에이전트가 선택할 수 있는 동작이

미리 정의된 고정된 조합으로 이루어짐

조향각, 속도 설정

조향각: -30° ~ 30°

속도: **1.6 m/s ~ 2.0 m/s**

총 9개의 조합으로 설계

작품 구성 및 개발내용

Action space type
Discrete

Action space

No.	Steering angle (°)	Speed (m/s)
0	-30.0	1.60
1	-24.0	1.80
2	-16.0	2.00
3	-8.0	2.00
4	0.0	2.00
5	8.0	2.00
6	16.0	2.00
7	24.0	1.80
8	30.0	1.60

Framework
Tensorflow

Reinforcement learning algorithm
PPO

Hyperparameter	Value
Gradient descent batch size	64
Entropy	0.01
Discount factor	0.9
Loss type	Huber
Learning rate	0.0003
Number of experience episodes between each policy-updating iteration	20
Number of epochs	10

02. Hyperparameter

디폴트 값을 따르면서 학습 진척에 따라 적절한 값으로 조정

Discount factor를 초기값 **0.99 -> 0.9**로 조정

Why?

한정된 시간 동안 학습하는 상황에서 에이전트가 너무 먼 미래의 보상에 지나치게 의존하지 않도록 하기 위해서

작품 구성 및 개발내용

03. Reward Function (보상함수)

```python
import math


SIGHT = 0.9

MAX_REWARD = 3.0
MIN_REWARD = 0.0001


def dist(point1, point2):
    return ((point1[0] - point2[0]) ** 2 + (point1[1] - point2[1]) ** 2) ** 0.5


def rect(r, theta):
    """
    theta in degrees
    returns tuple; (float, float); (x,y)
    """

    x = r * math.cos(math.radians(theta))
    y = r * math.sin(math.radians(theta))
    return x, y

def polar(x, y):
    """
    returns r, theta(degrees)
    """

    r = (x ** 2 + y ** 2) ** 0.5
    theta = math.degrees(math.atan2(y, x))
    return r, theta

def angle_mod_360(angle):
    """
    Maps an angle to the interval -180, +180.
    Examples:
    angle_mod_360(362) == 2
    angle_mod_360(270) == -90
    :param angle: angle in degree
    :return: angle in degree. Between -180 and +180
    """
    n = math.floor(angle / 360.0)

    angle_between_0_and_360 = angle - n * 360.0

    if angle_between_0_and_360 <= 180.0:
        return angle_between_0_and_360
    else:
        return angle_between_0_and_360 - 360


def get_waypoints_ordered_in_driving_direction(params):
    # return get_center_waypoints()

    # waypoints are always provided in counter clock wise order
    if params["is_reversed"]:  # driving clock wise.
        return list(reversed(params["waypoints"]))
    else:  # driving counter clock wise.
        return params["waypoints"]


def up_sample(waypoints, factor=10):
    """
    Adds extra waypoints in between provided waypoints
    :param waypoints:
    :param factor: integer. E.g. 3 means that the resulting list has 3 times as many points.
    :return:
    """
    p = waypoints
    n = len(p)

    return [
        [
            i / factor * p[int((j + 1) % n)][0] + (1 - i / factor) * p[j][0],
            i / factor * p[int((j + 1) % n)][1] + (1 - i / factor) * p[j][1],
        ]
        for j in range(n)
        for i in range(factor)
    ]


def get_target_point(params):
    waypoints = up_sample(get_waypoints_ordered_in_driving_direction(params), 20)

    car = [params["x"], params["y"]]

    distances = [dist(p, car) for p in waypoints]
    min_dist = min(distances)
    i_closest = distances.index(min_dist)

    n = len(waypoints)

    waypoints_starting_with_closest = [waypoints[(i + i_closest) % n] for i in range(n)]

    r = params["track_width"] * SIGHT

    is_inside = [dist(p, car) < r for p in waypoints_starting_with_closest]
    i_first_outside = is_inside.index(False)

    if i_first_outside < 0:
        # this can only happen if we choose r as big as the entire track
        return waypoints[i_closest]

    return waypoints_starting_with_closest[i_first_outside]


def get_target_steering_degree(params):
    tx, ty = get_target_point(params)
    car_x = params["x"]
    car_y = params["y"]
    dx = tx - car_x
    dy = ty - car_y
    heading = params["heading"]

    _, target_angle = polar(dx, dy)

    steering_angle = target_angle - heading
```

주요 보상 함수

adjust_waypoints_order

차량이 주행하는 방향에 따라 트랙 웨이포인트를 정렬합니다.

*웨이포인트
트랙을 작은 조각으로 나눈 점들의 집합으로,
트랙 상에서 차량이 어디로 가야 할지 방향을 제공

interpolate_waypoints

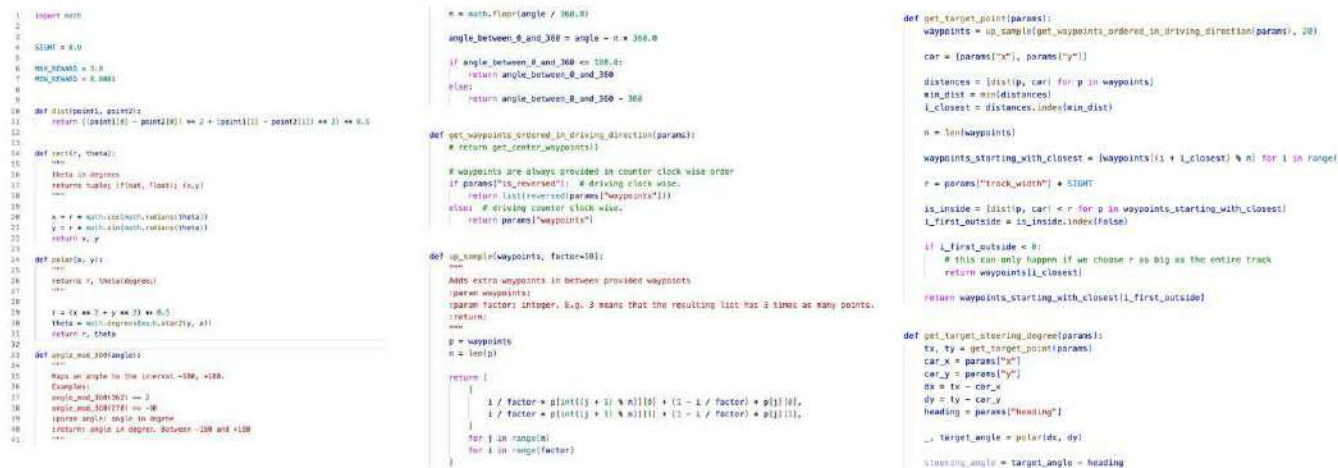
웨이포인트 사이에 더 많은 점을 추가하여 트랙을 매끄럽게 만듭니다.

find_target_point

차량이 주행해야 할 목표 지점을 계산합니다.

calculate_steering_degree

목표 지점으로 향하는 조향각을 계산합니다.



주요 보상 함수

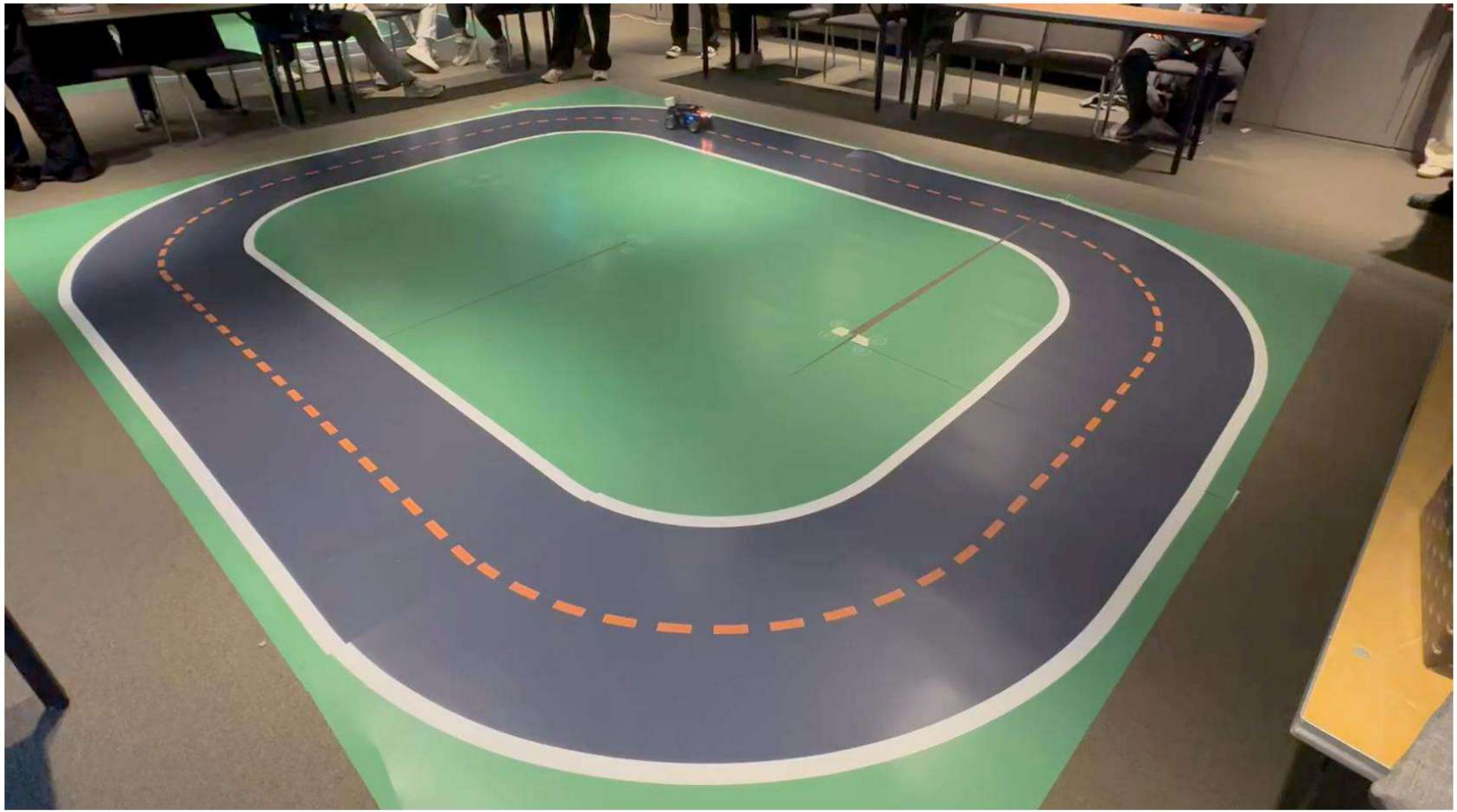
유연성

조향각 최적화

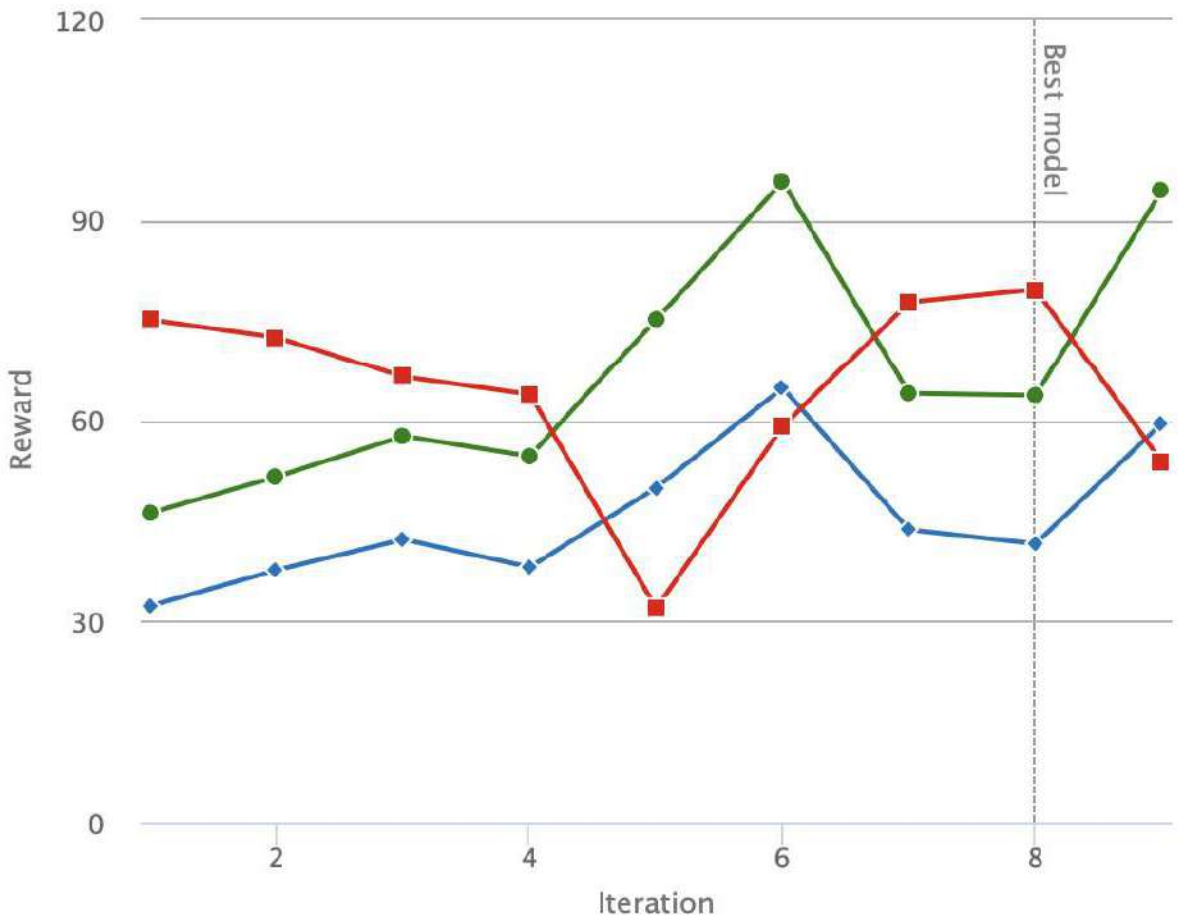
정밀한 경로 계산

목표 지점 기반 학습

구현 결과



구현 결과



보상 함수 개선

주요 변경점

1. 중심선 거리 보상 강화: 안정적인 주행 유도

2. 곡선 적응 로직 추가: 방향 차이에 따른 보상 조정

3. 조향 각도 제한: 과도한 조향 억제

개선 후 구현 결과


newmodel-clone2...
1/3
Race | 00:00.000
Reset | 0
00.00 m/s
Time trial evaluation

구현 결과

1차 주행 시간 (초)	2차 주행 시간 (초)	3차 주행 시간 (초	최고 기록 (초)	패널티	최종 기록 (초)
9.00	7.02	8.52	8.52	2.00	10.18

결과 분석

주행 테스트 때에는 평균 7초 대의 기록을 가졌지만 실제 주행 때는 좀 더 낮은 성능을 보였다.
그 이유를 유추해보자면, 테스트 주행 때 트랙 근처에 사람이 서있으면 차가 이탈하는 상황이 발생했다.

이는 **강화학습 모델**이 시각적으로 입력받는 환경에서 **그림자의 영향**을 제대로 처리하지 못했기 때문일 가능성이 크다.
모델이 흰색의 트랙 경계선 기준으로 학습했다면, 그림자가 생길 경우 그 기준이 달라져 제대로 경로를 유지하지 못할 것이다.

발전 가능성

👨‍💻**기술적 성장**: 강화학습 원리 설계 및 테스트 경험 축적
🤖**응용 가능성**: 자율주행 외 다양한 로봇 시스템에 활용 가능성 확인

향후 계획

- **심화 학습**: 복잡한 환경에서의 강화학습 실험 진행
- **스마트 로봇 기술 응용**: IoT 스마트 홈 기기, 스마트 물류 최적화 등 실질적인 문제 해결
- **대규모 프로젝트 참여**: 자율 주행 차량, 드론 경로 최적화 프로젝트 참여

감사합니다.