Methodology Introduction

Circle

I used TD3 for circle (best score for circle). The reason for using TD3 for the circle map is that it is my intuition to use TD3 for continuous action space.

Austria

I used PPO with discretized actions for the Austria map (best score for Austria & pass both baselines). The main reason for using PPO is that the agent using TD3 failed to learn (I tried several things but all failed, details in comparison part).

Experiment Design and Implementation

Description of the training processes

Circle – final setting (TD3)

1. Reward function

```
if info["wrong_way"]:
    reward -= 0.001
if info["wall_collision"]:
    reward -= 0.01
```

2. Due to the simplicity of the map, I set both motor and steer to [0, 1].
3. I stack 4 frames together to train the model.
4. When evaluating, I fix the motor to 1.

Austria – final setting (PPO)

1. Reward function: punish wrong_way / collision / slow speed, give additional reward when passing a hard turn.

```
velocity = info['velocity'][0] **2 + info['velocity'][1]**2
if info["wrong_way"]:
    reward = -0.005
elif info["wall_collision"]:
    reward = -0.01
    print("collide")
    print(info["progress"])
elif velocity < 0.2:
    reward = -0.0006
elif (info["progress"] > 0.36 and info["progress"] < 0.37):
    reward *= 5
```

2. Action space [motor, steer]

```
self.actions = [[0.1, 0], [0.1, 1], [0.1, -1], [-0.1, 0], [-0.1, -1], [-0.1, 1]]
```

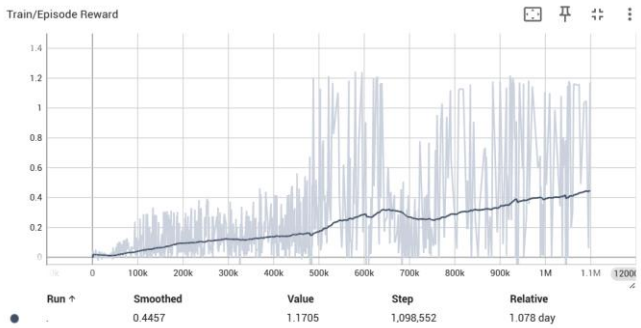
3. Frame stack + frame skip

   I have a deque(maxlen = 5) to store 5 * deque(maxlen = 4), and perform skip half of the frame. When I receive a frame, I will pop one inner deque from the outer deque, put one frame into it, and push the

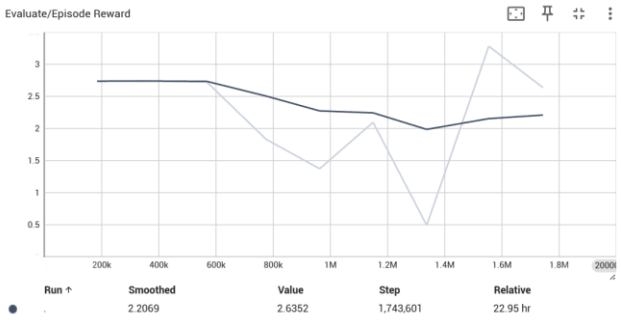
inner deque back to the outer deque. After I decide on the action, I will do the action twice, receiving two sets of objects to put in the replay buffer. I put (reward1+reward2) * 100 and (terninate1 or terminate2) into the buffer.

4. Environment
   I randomize starting point and use “austria_competition_collisionStop” according to the advice of the TAs.

I trained PPO with actions having a maximum of 0.01 first and got the following training curve (rewards are deducted for punishment of collision).



It seems that 0.01 works, then I use model trained on motor 0.01 as pretrain and modify 0.01 to 0.1 directly. Here is the evaluation curve.



Austria TD3 (failed)

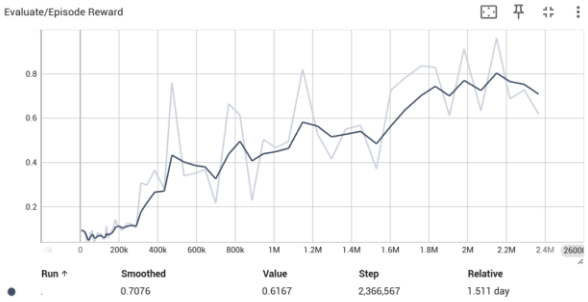
(number sequence = time)

1. Same code as circle with modified action space (-1, 1) for motor & action -> motor too fast.
2. Set motor to (0.1, -0.1) -> failed due to punishing too much on collision.
3. Modified collision penalty to 0.01 (reference the reward/punishment ratio in Lab4) -> still failed.
4. Train accelerate and break separately (three actions) -> still failed,

modify back to 2 actions.

5. Add more positive action noise on sharp turns -> still failed.
6. Add frame stack like the one in PPO & modify the environment like the one in PPO -> still failed to converge.

I tried several parameters and tricks with TD3, and this is the best result. According to my observation, the model failed to pass the sharp turns before progress 0.40 and 0.55.



Neural network architectures

Circle (TD3) – same as Lab4

```python
class ActorNetSimple(nn.Module):
    def __init__(self, state_dim: int, action_dim: int, N_frame: int) -> None:
        super().__init__()
        self.conv = nn.Sequential(
            nn.Conv2d(N_frame, 16, kernel_size=3, padding=1),
            nn.ELU(),
            nn.Conv2d(16, 16, kernel_size=3, padding=1),
            nn.ELU(),
            nn.MaxPool2d(kernel_size=2),
            nn.Conv2d(16, 16, kernel_size=3, padding=1),
            nn.ELU(),
            nn.Conv2d(16, 16, kernel_size=3, padding=1),
            nn.ELU(),
            nn.MaxPool2d(kernel_size=2),
            nn.Conv2d(16, 32, kernel_size=3, padding=1),
            nn.ELU(),
            nn.Conv2d(32, 16, kernel_size=3, padding=1),
            nn.ELU(),
            nn.MaxPool2d(kernel_size=2),
        )

        self.linear = nn.Sequential(
            nn.Linear(16*(state_dim//8)**2, 256),
            nn.LayerNorm(256),
            nn.ELU(),
            nn.Linear(256, action_dim),
            nn.LayerNorm(action_dim),
            nn.Tanh()
        )
```

```python
class CriticNetSimple(nn.Module):
    def __init__(self, state_dim: int, action_dim: int, N_frame: int) -> None:
        super().__init__()
        self.conv = nn.Sequential(
            nn.Conv2d(N_frame, 16, kernel_size=3, padding=1),
            nn.ELU(),
            nn.Conv2d(16, 16, kernel_size=3, padding=1),
            nn.ELU(),
            nn.MaxPool2d(kernel_size=2),
            nn.Conv2d(16, 16, kernel_size=3, padding=1),
            nn.ELU(),
            nn.Conv2d(16, 16, kernel_size=3, padding=1),
            nn.ELU(),
            nn.MaxPool2d(kernel_size=2),
            nn.Conv2d(16, 32, kernel_size=3, padding=1),
            nn.ELU(),
            nn.Conv2d(32, 16, kernel_size=3, padding=1),
            nn.ELU(),
            nn.MaxPool2d(kernel_size=2),
        )

        self.action_linear = nn.Sequential(
            nn.Linear(action_dim, 256),
            nn.LayerNorm(256),
            nn.ELU()
        )

        self.state_linear = nn.Sequential(
            nn.Linear(16*(state_dim//8)**2, 256),
            nn.LayerNorm(256),
            nn.ELU(),
        )

        self.concat_linear = nn.Sequential(
            nn.Linear(512, 64),
            nn.LayerNorm(64),
            nn.ELU(),
            nn.Linear(64, 64),
            nn.LayerNorm(64),
            nn.ELU(),
            nn.Linear(64, 1)
        )
```

Austria (PPO) – same as Lab3

```python
class AtariNet(nn.Module):
    def __init__(self, num_classes=4, init_weights=True):
        super(AtariNet, self).__init__()

        self.cnn = nn.Sequential(nn.Conv2d(4, 32, kernel_size=8, stride=4),
                                 nn.ReLU(True),
                                 nn.Conv2d(32, 64, kernel_size=4, stride=2),
                                 nn.ReLU(True),
                                 nn.Conv2d(64, 64, kernel_size=3, stride=1),
                                 nn.ReLU(True)
                                 )
        self.action_logits = nn.Sequential(nn.Linear(1024, 512),
                                 nn.ReLU(True),
                                 nn.Linear(512, num_classes)
                                 )
        self.value = nn.Sequential(nn.Linear(1024, 512),
                                 nn.ReLU(True),
                                 nn.Linear(512, 1)
                                 )
```

Details of the hyper-parameters

Circle (TD3)

```
"gamma": 0.99,
"tau": 0.005,
"batch_size": 64,
"lra": 4.5e-5,
"lrc": 4.5e-5,
"update_freq": 2,
```

Austria (PPO)

```
"update_sample_count": 10000,
"discount_factor_gamma": 0.99,
"discount_factor_lambda": 0.95,
"clip_epsilon": 0.2,
"max_gradient_norm": 0.5,
"update_ppo_epoch": 3,
"learning_rate": 2.5e-5,
"value_coefficient": 0.5,
"entropy_coefficient": 0.01,
"horizon": 128,
"batch_size": 128,
```

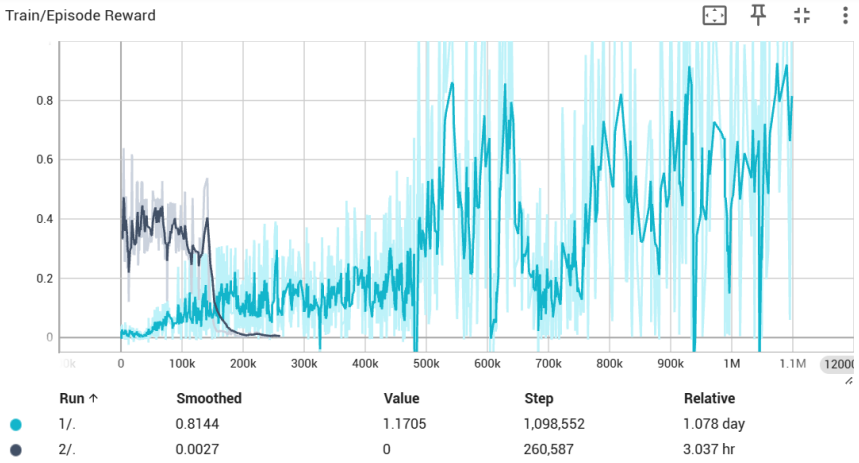
List of packages, tools, or resources used

Source code from Lab3, Lab4. Packages from normal lab + racarEnv.

Method Comparison and Evaluation

Trying different methods and showing their results

As mentioned above, I tried both PPO and TD3 on austria map, so I'm gonna just draw the curve here. The time I train these two models does not match, but you can still the trend of the two curves. Dark blue – TD3, blue – PPO.



Run	Smoothed	Value	Step	Relative
1/.	0.8144	1.1705	1,098,552	1.078 day
2/.	0.0027	0	260,587	3.037 hr

Comparing the effectiveness of different approaches

When training TD3, I never passed the two sharp turns, and the agent tended to stop at the beginning point even if I added a penalty for not moving. PPO does not have the problem of not moving because it uses a stochastic policy. As you can see in the graph, TD3 drops instantly and starts to get no reward (this happens all the time, maybe because of the reward function), whereas PPO grows continuously.

Analyzing the successful and unsuccessful cases

The action space of this environment is quite big, so if using TD3, you might have to explore a lot. I set the motor space to (-1, 1), so the TD3 agent might take a lot of time to explore. My PPO agent has only 6 options for the action and samples an action with probability output by the policy. I think the size of the action space and environment complexity are the main reasons why TD3 succeeded on the circle map but failed on the Austria map.

Discussing the key observations and insights

1. Actions might need smoothed
   I observed that the agent kept going left and right even if it was on a straight road. I watched the videos on the top of the leaderboard. Their

actions are quite smooth and they do not make unnecessary moves like my agent did. I might need to have more actions or have an average between actions to make my agent drive smoother.

2. 1st person view is hard to train
   The map in Lab 4 is as complicated as the final project. TD3 did well on Lab4 but failed on austria. The reason might be that the observation received in Lab 4 is a bird eye view, so the agent will know what the angle of the next turn is. However, this information cannot be observed from the 1st person's viewpoint, so the agent cannot have enough time to prepare for the next turn.
3. Continuous action to discrete action gap
   When I started to shift my agent from TD3 to PPO, I had a huge question – what actions should I take, what should be the scale for the motor? I'm not sure how much motor I can give to the agent, so I tried starting with 0.01 and increasing to 0.1 at the end. I did train a version of motor 1 before I demo, but it turned out that it is unstable, so I did not use the agent.

Challenges and Learning Points

Encountered challenges during the implementation process

1. Python environment
   There is a requirement for python version, and I have some Cuda driver / torch / python version conflicts, but I solved this by upgrading torch by one version.
2. The 2 sharp turns
   It took me 2 weeks in total to train these two sharp turns using TD3 and never have I succeeded. But it took me two days to pass the two corners using PPO.

The learnings from these challenges

There are actually lots of details I did not take into consideration when implementing. For example, I used reward -= 0.1 when collision at first so the agent receives a reward when collision / I stack the 4 coherent frames together, which is approximately the same as stacking nothing / I did not use collisionStop so the agent has chaotic frames between collision, and the critic value therefore calculated incorrectly. I found these problems one by one, and the training of one model is absolutely long… so it took me lots of time staring at the screen with anxiety. Maybe I could be more cautious of the environment in the future.

Future Work

The proposal of ideas for potential improvements

1. Have more actions for my PPO agent
   I tried motor value 1 but failed, maybe I can give more options like 1 / 0.5 / 0.1…
2. Action smoothing
   I tried adding previous actions and /3 per action to have higher maximum acceleration and smoother actions, but I didn't have enough time to finish the training.

The suggestion of additional research directions for the future

Continuous actions are more natural for driving, maybe I can demo and record the trajectory and put it in the replay buffer of TD3 per n episode or use other methods to leverage the demonstration (DQfD)