RL Final Project Report

Student ID: 110550014 Name: 吳權祐

1. Methodology Introduction

I have tried **PPO** and **TD3** in this project. After my implementation and experiment, I find that PPO has better results on evaluation. So I utilize PPO as my base model for both maps, and focus more on other designs like action space or reward and termination.

2. Experiment Design and Implementation

2.1. Training Process

I train all my PPO agents with the original reward and termination function. I have tried to train TD3 with my own reward and termination function like the code below, however this didn't improve performance of my model.

```
if not np.any(info['acceleration']) and not np.any(info['velocity']):
    terminates = True
    reward -= 1
```

I wrap observations with these two functions in environments for PPO and TD3 respectively.

PPO

```
def wrap_obs(self, obs):
    # convert to grayscale obs = 128*128*3
    obs = np.transpose(obs, (1, 2, 0))
    obs = cv2.cvtColor(obs, cv2.COLOR_BGR2GRAY) # 96x96
    obs = cv2.resize(obs, (84, 84), interpolation=cv2.INTER_AREA)
    if len(self.frames) != self.frames.maxlen:
        for _ in range(self.frames.maxlen - len(self.frames)):
            self.frames.append(obs)
    else:
        self.frames.append(obs)
    obs = np.stack(self.frames, axis=0)
    return obs
```

TD3

```
def wrap_obs(self, obs):
    # convert to grayscale obs = 128*128*3
    obs = np.transpose(obs, (1, 2, 0))
    obs = cv2.cvtColor(obs, cv2.COLOR_BGR2GRAY) # 96x96
    # obs = cv2.resize(obs, (84, 84), interpolation=cv2.INTER_AREA)
    if len(self.frames) != self.frames.maxlen:
        for _ in range(self.frames.maxlen - len(self.frames)):
            self.frames.append(obs)
    else:
        self.frames.append(obs)
    obs = np.stack(self.frames, axis=0)
    return obs
```

For all agents, **the range of steer** and **motor** are hyper-parameters in my training processes, while the PPO model has another additional hyper-parameter used to decide **numbers of discretization** of these two ranges. This is an example of hyper-parameter configuration.

```
"motor_range": [0.5, 1],
"steering_range": [0.5, 1],
"discretize_num": [2, 4]
```

In my training process, I have tried lots of different configurations of hyper-parameters I mentioned. I observed that when I train PPO models, smaller discretization numbers get better training efficiency, and the motor range of Austria map should not be too high.

2.2. Neaural Network

Both network architectures of PPO and TD3 are the same as those in hw3 and hw4.

PPO

```python
class RaceCarNet(nn.Module):
    def __init__(self, num_classes=4, init_weights=True):
        super(RaceCarNet, 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(7*7*64, 512),
                                 nn.ReLU(True),
                                 nn.Linear(512, num_classes)
                                 )
        self.value = nn.Sequential(nn.Linear(7*7*64, 512),
                                 nn.ReLU(True),
                                 nn.Linear(512, 1)
                                 )

        if init_weights:
            self._initialize_weights()
```

TD3

```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)
        )
```

2.3. Hyper-parameters

Circle

```python
config = {
    "gpu": True,
    "training_steps": 1e8,
    "update_sample_count": 10000,
    "discount_factor_gamma": 0.99,
    "discount_factor_lambda": 0.95,
    "clip_epsilon": 0.2,
    "max_gradient_norm": 0.5,
    "batch_size": 128,
    "logdir": 'log/circle_origin_reward/',
    "update_ppo_epoch": 3,
    "learning_rate": 2.5e-4,
    "value_coefficient": 0.5,
    "entropy_coefficient": 0.01,
    "horizon": 128,
    "env_id": 'ALE/Enduro-v5',
    "eval_interval": 100,
    "eval_episode": 3,
    "scenario": 'circle_cw_competition_collisionStop'
    "motor_range": [-1, 1],
    "steering_range": [-1, 1],
    "discretize_num": 20
}
```

Austria

```
config = {
    "gpu": True,
    "training_steps": 1e8,
    "update_sample_count": 10000,
    "discount_factor_gamma": 0.99,
    "discount_factor_lambda": 0.95,
    "clip_epsilon": 0.2,
    "max_gradient_norm": 0.5,
    "batch_size": 128,
    "logdir": 'log/austria_original_reward_4/',
    "update_ppo_epoch": 3,
    "learning_rate": 2.5e-4,
    "value_coefficient": 0.5,
    "entropy_coefficient": 0.01,
    "horizon": 128,
    "env_id": 'ALE/Enduro-v5',
    "eval_interval": 100,
    "eval_episode": 3,
    "scenario": 'austria_competition',
    "motor_range": [-0.1, 0.1],
    "steering_range": [-1., 1.],
    "discretize_num": [3, 5]
}
```

2.4. Environment

These are some important packages, tools, and resources I used.

Ubuntu 20.04

NVIDIA GeForce RTX 3080 Ti

CUDA 12.4

Conda 24.9.2

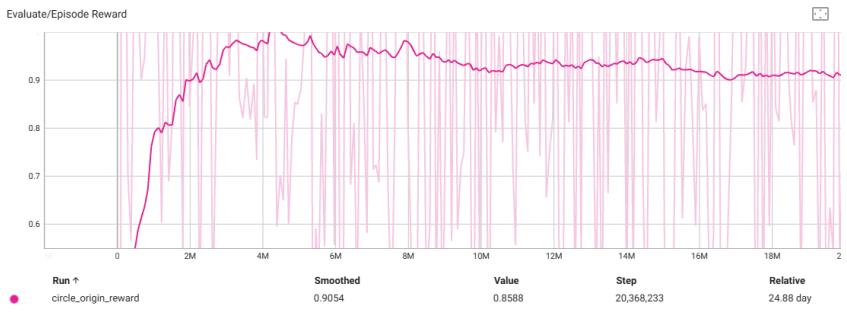
Python 3.10.15

- gymnasium 0.28.1
- numpy 1.25.2
- pytorch 2.5.1
- tensorboard 2.18.0
- opencv 4.8.1

3. Method Comparison and Evaluation

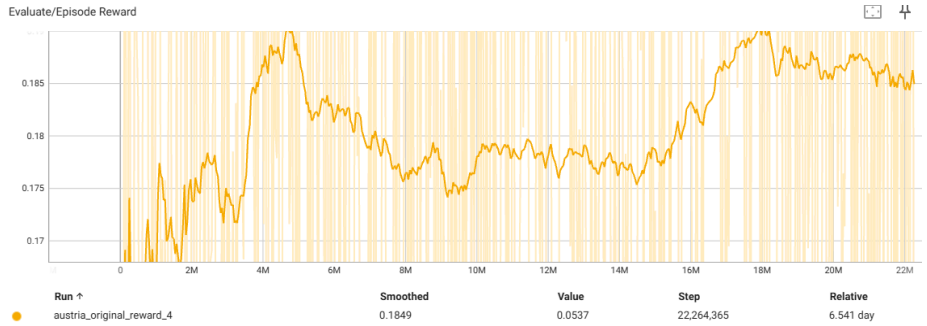
3.1. PPO - Training Curve & Evaluation Result

Circle



```
==========================================
Evaluating...
episode 1 reward: 1.2744630071599046
episode 2 reward: 1.2923627684964203
episode 3 reward: 1.2136038186157523
average score: 1.2601431980906923
```

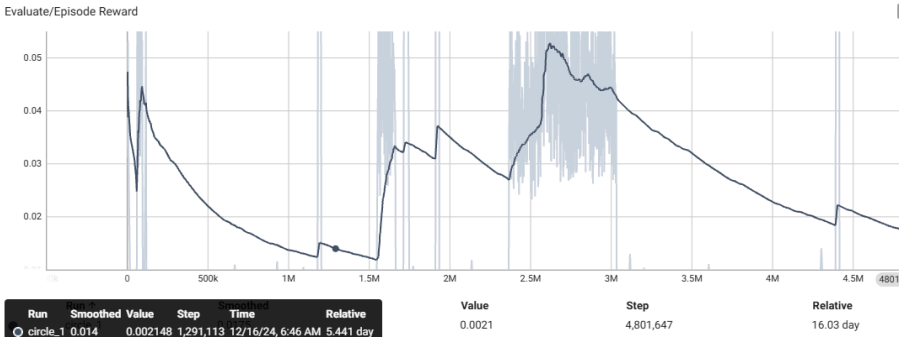
Austria




```
==========================================
Evaluating...
episode 1 reward: 0.35242290748898686
episode 2 reward: 0.23222152297042165
episode 3 reward: 0.303964757709251
average score: 0.2962030627228865
==========================================
```

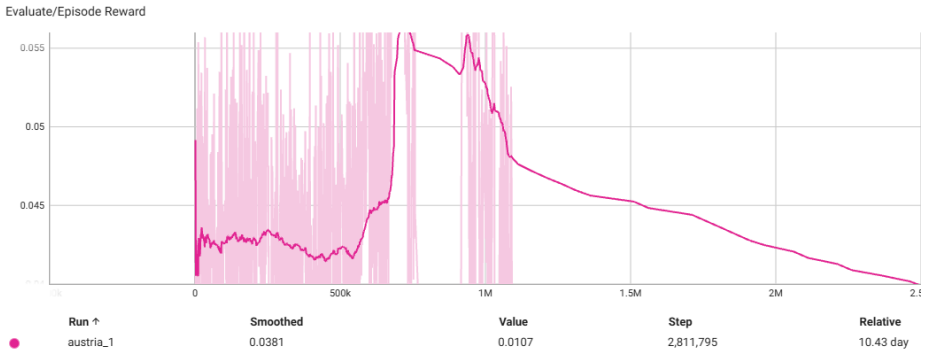
3.2. TD3 - Training Curve & Evaluation Result

Circle




```
==========================================
Evaluating...
Episode: 1      Length: 1249    Total reward: 0.00
Episode: 2      Length: 1249    Total reward: 0.00
Episode: 3      Length: 1249    Total reward: 0.00
Episode: 4      Length: 1249    Total reward: 0.00
Episode: 5      Length: 1249    Total reward: 0.00
Episode: 6      Length: 1249    Total reward: 0.00
Episode: 7      Length: 1249    Total reward: 0.00
Episode: 8      Length: 1249    Total reward: 0.00
Episode: 9      Length: 1249    Total reward: 0.00
Episode: 10     Length: 1249    Total reward: 0.00
average score: 0.0015513126491647489
```

Austria



3.3. Analysis & Comparison & Discussion

Since I could not train my TD3 agents well, I guess the reason is that I didn't design a suitable reward and termination policy for the model. The comparison result shows that PPO agents could get better performance. Within comparison of different configurations of PPO training, I observed that a smaller discretization number could lead to better performance.

4. Challenges and Learning Points

One of the challenges I have met is wrapping observations from the original environment. To address this problem, I inherited another environment from the original environment and reimplemented some functions from the original environment.

5. Future Work

Reward and termination function might be one of the directions I am interested in. Maybe some learning based adaptable function or modules could be applied into this stage to improve the training process of RL agents, reducing the manual design of these functions.