RL_Final_Project_313552041_洪日昇

Methodology Introduction

在這次的 final project 中，demo 的方法是利用 container 作為 server 傳送 observation 給 client，client 得到 observation 之後利用 train 好的 model 當作 agent 做出 action 的選擇並且回傳給 server。

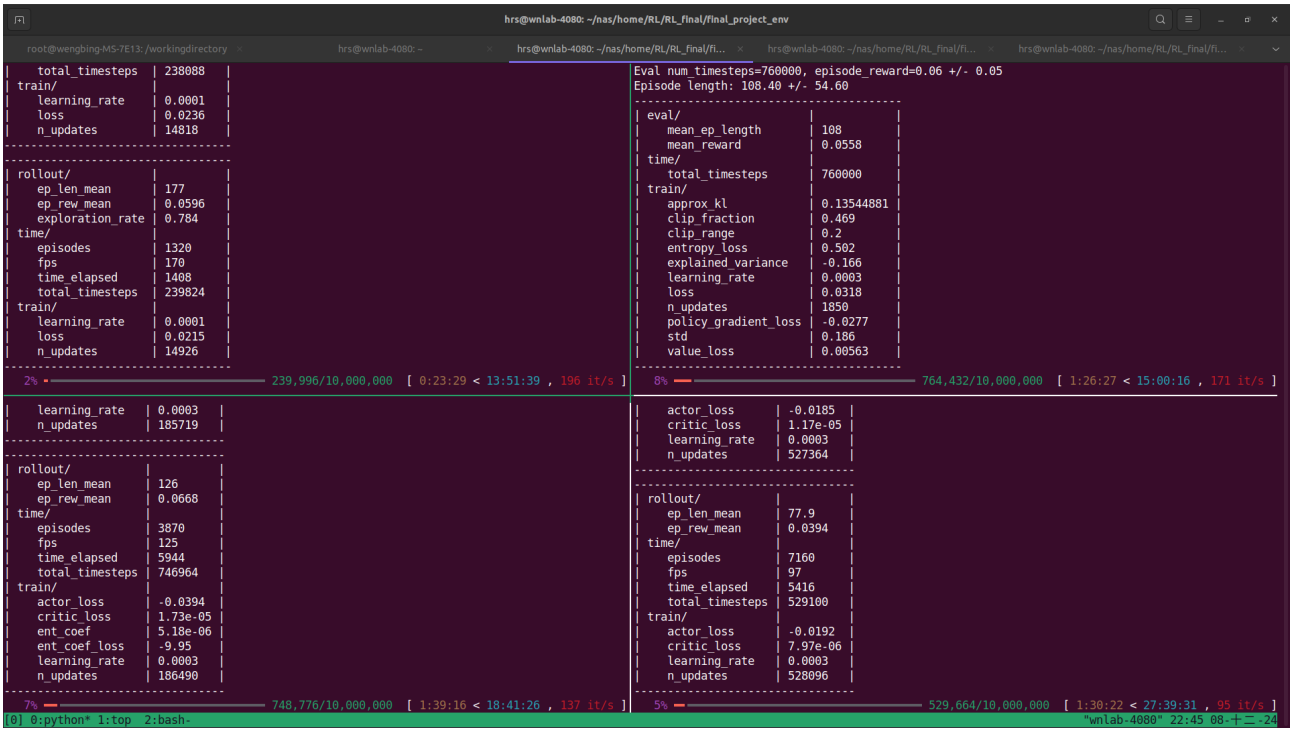
因此比較好的方法是，像前幾次的 lab 一樣，學 server 創建 environment 的方式自己利用 Racecar_env 來訓練 model。

演算法挑選部份為了快速的驗證模型正確性，這邊就不自己手刻演算法，使用 Stable-Baselines3 提供的功能，來訓練模型，會發現這個是一個相當強大的套件，可以快速的驗證想法。

在一開始的訓練過程中我利用 SB3 來訓練，挑選了幾個上課上過的幾個好用的演算法
DQN、PPO、TD3、SAC 來對 circle_cw_competition_collisionStop and austria_competition 兩個地圖做嘗試。

由於 Racecar 為 continuous action space，因此我希望可以透過 PPO、TD3、SAC 來嘗試訓練，此外我也嘗試了將 continuous action space 變成 discrete action space 讓 DQN 做訓練，比較了幾種的 model 最後結果的優劣。

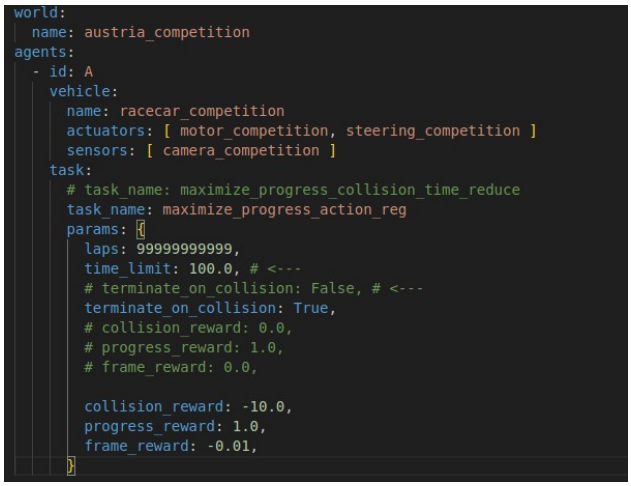
而我最後是使用 PPO 來作為我兩個地圖的訓練演算法，並且利用一些跑得不錯的 model 作為 pretrained model 繼續對它做 fine tunning 加速訓練過程。



圖一、Stable-Baselines3 訓練過程

Experiment Design and Implementation

一開始我更改 scenarios/xxx.yml 更改裡面的內容希望可以透過改動裡面的參數，還有利用 task 來改變 reward shaping，可以參考 racecar_gym/tasks/progress_based.py 以及 racecar_gym/tasks/__init__.py，本來期待可以使用 maximize_progress_action_reg 來幫助滑順的 action 操作，但是結果是不如預期的，最後還是使用了預設的設定來處理 reward shaping。



```yaml
world:
  name: austria_competition
agents:
  - id: A
    vehicle:
      name: racecar_competition
      actuators: [ motor_competition, steering_competition ]
      sensors: [ camera_competition ]
    task:
      # task_name: maximize_progress_collision_time_reduce
      task_name: maximize_progress_action_reg
      params: {
        laps: 99999999999,
        time_limit: 100.0, # <---
        # terminate_on_collision: False, # <---
        terminate_on_collision: True,
        # collision_reward: 0.0,
        # progress_reward: 1.0,
        # frame_reward: 0.0,

        collision_reward: -10.0,
        progress_reward: 1.0,
        frame_reward: -0.01,
      }
```

```yaml
world:
  name: austria_competition
agents:
  - id: A
    vehicle:
      name: racecar_competition
      actuators: [ motor_competition, steering_competition ]
      sensors: [ camera_competition ]
    task:
      # task_name: maximize_progress_collision_time_reduce
      task_name: maximize_progress_action_reg
      params: {
        laps: 99999999999,
        time_limit: 100.0, # <---
        # terminate_on_collision: False, # <---
        terminate_on_collision: True,
        # collision_reward: 0.0,
        # progress_reward: 1.0,
        # frame_reward: 0.0,

        collision_reward: -100.0,
        progress_reward: 100.0,
        frame_reward: -0.01,
      }
```

因此我改成自訂 reward function

 # 在進行行為計算前，得分值被初始化為零：

reward = 0

 # 鼓勵加速懲罰減速

reward += 1 * motor_action

 # 懲罰過度變化 motor 與 steering

reward -= 0.1 * (abs(motor_action - self.prev_info['motor']) + abs(steering_action - self.prev_info['steering']))

 # 撞牆嚴重扣分並且停止遊戲

```
if state['wall_collision'] == True:
    reward = -100
    terminated = True
```

此外因為也是效果不彰我按照助教的 tips 做了幾個 observation 的改動
GrayScale:
(3, 128, 128) -> (1, 128, 128)
Resize:
(1, 128, 128) -> (1, 84, 84)
Frame Stack:
(1, 84, 84) -> (8 , 84, 84)

在 env 中用 opencv 將 obs 變成灰階，再用 opencv 改大小，以上兩個動作可以幫助學習更快抓到重點，因為賽車遊戲的 obs 跟顏色沒有很大的關係，並且可以幫助減少使用的資源。
利用 SB3 的 VecFrameStack 來快速達成 framestack，幫助模型可以抓取時間前後關係來做決定。

Neural network architectures

利用 SB3 中的 CnnPolicy 來做訓練，CnnPolicy 使用卷積神經網路來提取圖像特徵，然後將特徵輸入到全連接層。以下是其主要結構:

```
ActorCriticCnnPolicy(
  (features_extractor): NatureCNN(
    (cnn): Sequential(
      (0): Conv2d(8, 32, kernel_size=(8, 8), stride=(4, 4))
      (1): ReLU()
      (2): Conv2d(32, 64, kernel_size=(4, 4), stride=(2, 2))
      (3): ReLU()
      (4): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1))
      (5): ReLU()
      (6): Flatten(start_dim=1, end_dim=-1)
    )
    (linear): Sequential(
      (0): Linear(in_features=3136, out_features=512, bias=True)
      (1): ReLU()
    )
  )
  (pi_features_extractor): NatureCNN(
    (cnn): Sequential(
      (0): Conv2d(8, 32, kernel_size=(8, 8), stride=(4, 4))
      (1): ReLU()
      (2): Conv2d(32, 64, kernel_size=(4, 4), stride=(2, 2))
      (3): ReLU()
      (4): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1))
      (5): ReLU()
      (6): Flatten(start_dim=1, end_dim=-1)
    )
    (linear): Sequential(
      (0): Linear(in_features=3136, out_features=512, bias=True)
      (1): ReLU()
    )
  )
  (vf_features_extractor): NatureCNN(
    (cnn): Sequential(
      (0): Conv2d(8, 32, kernel_size=(8, 8), stride=(4, 4))
      (1): ReLU()
      (2): Conv2d(32, 64, kernel_size=(4, 4), stride=(2, 2))
      (3): ReLU()
      (4): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1))
      (5): ReLU()
      (6): Flatten(start_dim=1, end_dim=-1)
    )
    (linear): Sequential(
      (0): Linear(in_features=3136, out_features=512, bias=True)
      (1): ReLU()
    )
  )
  (mlp_extractor): MlpExtractor(
    (policy_net): Sequential()
    (value_net): Sequential()
  )
  (action_net): Linear(in_features=512, out_features=2, bias=True)
  (value_net): Linear(in_features=512, out_features=1, bias=True)
)
```

Details of the hyper-parameters

一般參數

- 策略類型 (Policy Type): CnnPolicy
    - 適用於處理高維度輸入（如影像數據）。

    - 預設使用 Nature CNN 架構。
- 環境 (Environment): SubprocVecEnv 搭配 VecFrameStack 和 VecMonitor
    - 並行執行 16 個環境以加速數據收集。
    - 每個環境的輸入堆疊 8 幀，用於處理具有時序依賴性的問題。

學習率

- 類型: 線性衰減 (Linear schedule)
    - 初始學習率：3e-4
    - 最終學習率：5e-5
    - 隨著訓練進度的推進，學習率逐漸衰減：
      learning_rate=final_lr+progress_remaining×(initial_lr−final_lr)

訓練參數

- 總時間步數 (Total Timesteps): 3e6 (3,000,000 步)
- 每次更新的步數 (n_steps): 1024
    - 每個環境在收集 1024 步數據後進行一次策略更新。
- 批量大小 (batch_size): 64
    - 用於每次梯度更新的數據樣本數量。
- 迭代次數 (n_epochs): 10
    - 每次更新時執行 10 次梯度下降。

List of packages, tools, or resources used

套件 (Packages)

1. numpy
    - 用於數學計算和數據操作。
2. stable_baselines3
    - 提供強化學習演算法 (PPO) 的實現，並支援訓練與評估過程中的多種功能。
3. gymnasium
    - 用於創建和操作強化學習環境。
4. racecar_gym
    - 自定義賽車環境庫，支援不同賽道與渲染模式的賽車模擬。

工具 (Tools)

1. SubprocVecEnv
    - 用於並行執行多個環境，提升數據收集效率。
2. VecFrameStack
    - 幀堆疊工具，將多幀影像堆疊為單一輸入以捕捉時序特徵。
3. VecMonitor
    - 用於監控環境執行過程中的回報與評估數據。
4. EvalCallback

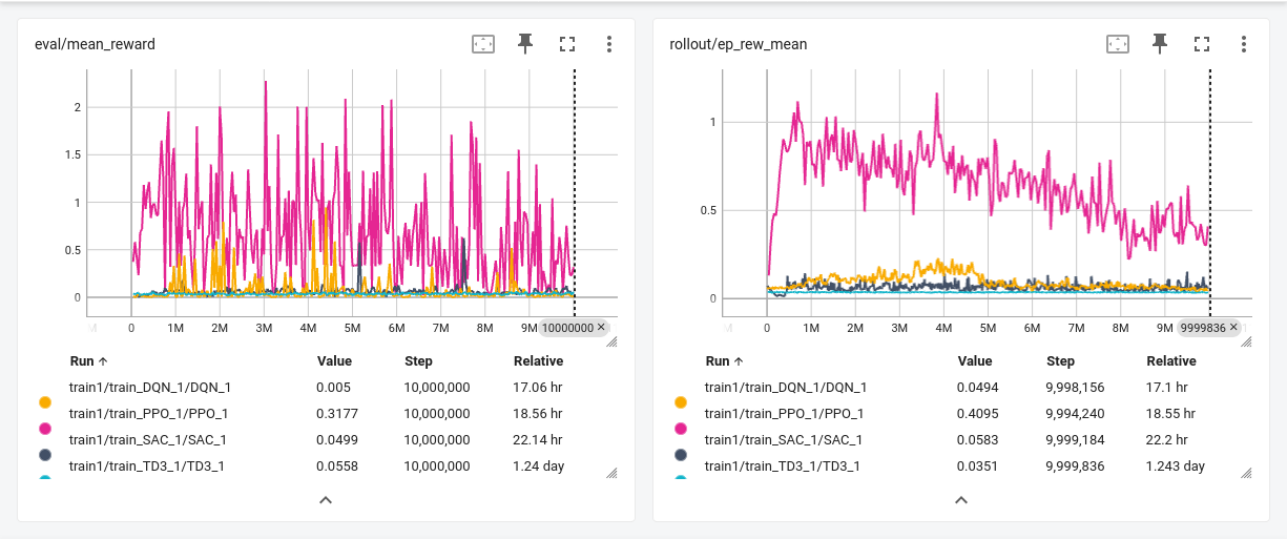
- 評估回調工具，用於定期評估模型性能並自動保存表現最佳的模型。

5. TensorBoard
    - 用於記錄和可視化訓練過程數據，例如學習率、回報值等。

資源 (Resources)

1. 賽車環境 (RaceEnv)
    - 來自 racecar_gym 的自訂賽車模擬環境，用於訓練和測試。
    - 支援地圖名稱如 austria_competition。
2. 預訓練模型
    - 文件路徑：log/train6/train_PPO_6/PPO_6/best_model_austria.zip
    - 包含已訓練模型的策略權重，用於模型的初始化與加速訓練。
3. 訓練日誌
    - 文件路徑：./log/train6/train_PPO_6/
    - 用於儲存訓練過程的 TensorBoard 日誌和模型檔案。

Method Comparison and Evaluation



一開始嘗試使用 DQN、PPO、TD3、SAC 這四種不一樣的 model 來對最基本的 reward shaping 做測試來看看哪一個的效果更好，在這邊可以發現 PPO 表現最佳，DQN 次之，其餘兩個的效果非常有限，並沒有什麼起色，因此後續將著重於 PPO 的訓練。

```
# 1st attempt: 1 , and the car stopped, afraid of hitting wall
reward = 0
reward += 1 * motor_action
reward -= 0.1 * (abs(motor_action - self.prev_info['motor']) + abs(steering_action - self.prev_info['steering']))
if state['progress'] > self.prev_info['state']['progress']: # move forward
    reward += 1000 * (state['progress'] - self.prev_info['state']['progress'])
elif state['progress'] == self.prev_info['state']['progress']:   # not moving
    reward -= 0.1
if state['wall_collision'] == True:
    reward = -100
    terminated = True
self.prev_info['motor'] = motor_action.copy()
self.prev_info['steering'] = steering_action.copy()
self.prev_info['state'] = state.copy()
```

```
# 2nd attempt:
reward = 0
reward += 1 * motor_action
reward -= 0.1 * (abs(motor_action - self.prev_info['motor']) + abs(steering_action - self.prev_info['steering']))
if state['progress'] > self.prev_info['state']['progress']: # move forward
    reward += 1000 * (state['progress'] - self.prev_info['state']['progress'])
elif state['progress'] == self.prev_info['state']['progress']:   # not moving
    reward -= 1
if state['wall_collision'] == True:
    reward = -100
    terminated = True
self.prev_info['motor'] = motor_action.copy()
self.prev_info['steering'] = steering_action.copy()
self.prev_info['state'] = state.copy()
```

```
# 3rd attempt: # speed up by not
reward = 0
reward += 0.5 * motor_action
reward -= 0.1 * (abs(motor_action - self.prev_info['motor']) + abs(steering_action - self.prev_info['steering']))
if state['progress'] > self.prev_info['state']['progress']: # 往前得分
    reward += 1000 * (state['progress'] - self.prev_info['state']['progress'])
elif state['progress'] == self.prev_info['state']['progress']:   # 停滯不前
    reward -= 0.3
if state['wall_collision'] == True:
    reward = -500
    terminated = True
self.prev_info['motor'] = motor_action.copy()
self.prev_info['steering'] = steering_action.copy()
self.prev_info['state'] = state.copy()
```

一開始的跑分是相當的不理想，reward funtion 的部份讓賽車太過於懼怕撞牆而變得不敢往前，覺得停下來得分數可能比撞牆還要高，因此慢慢調整 reward function，先讓車子停下後有更嚴重的懲罰，使車子能夠往前，平衡往前以及不要撞牆的 reward function。

此外 circle 與 austria 都可以用這些 reward function 下可以跑得不錯。

Challenges and Learning Points:

在訓練一開始的時候車子都會很懼怕前進，變成很難控制車子，訓練也一直沒有進展，本來以為模型練不好一直重新訓練，也都沒有起色，可是訓練又慢很難快速知道錯誤在哪裡，也因此在網路上查到可以利用 SB3 輕鬆使用多環境來訓練模型，這能夠加速訓練兩三倍，非常好用。

嘗試了多環境的測試後發現，更改 reward 對於訓練模型是非常重要的一步驟，SB3 都能夠按照我們的 reward function 收斂，只是效果如果不好，可能就要對模型進行 fine tunning 改善 reward function 的設計。

後續查到原來使用 pretrained model 可以幫助模型訓練更快進入狀況，這也大幅減少了模型從頭開始訓練浪費時間的過程。

此外訓練出來的模型也不是非常穩定可能因為 noise 造成最高分能達到 3 laps 的模型有時候可能 0.5 有時候 1.5 有時候 2.5，相當不穩定。

Future Work:

在此次賽車遊戲的訓練過程中，Agent 完全依靠自身的探索與環境互動來學習。這種方式讓 Agent 能夠自主發現有效的策略，但仍存在潛在的改進空間，可以進一步加速訓練並提升 Agent 的表現。

首先，引入專家操作數據是一個重要的方向。透過模仿學習技術，例如行為克隆或生成對抗模仿學習，Agent 可以模仿專家的操作，快速獲得基礎的駕駛能力。這種方式能顯著縮短 Agent 探索低效策略的時間，加速學習過程。同時，專家的行為示例還可以幫助 Agent 更快理解環境的動態特徵，為後續的強化學習提供更好的初始策略。

其次，結合路徑規劃演算法也能有效提升 Agent 的駕駛表現。透過應用路徑規劃技術，Agent 可以學習到專業賽車比賽中常用的策略，例如利用最小曲線半徑來縮短行駛距離，並減少速度損失。此外，外-內-外策略能在彎道中最大化曲線半徑，讓 Agent 能夠實現平滑轉彎並保持較高的速度。這些方法不僅可以作為 Agent 的參考路徑，還能結合獎勵函數設計，進一步引導 Agent 學習最佳駕駛模式。

優化 reward function 則是另一個提升訓練效果的關鍵。針對 Agent 的行為進行更加精細的評估，可以鼓勵高效的駕駛行為，例如流暢的過彎、保持高速行駛以及貼近最佳路徑。同時，對於不必要的煞車或不穩定的操作，應進行適當的懲罰，從而促進 Agent 學習更為穩定和高效的駕駛策略。

通過引入專家操作數據、結合路徑規劃演算法以及優化獎勵函數，訓練過程可能可以變得更加高效，希望未來有機會可以以這個方向做學習。

Reference:

https://stable-baselines3.readthedocs.io/en/master/

https://repositum.tuwien.at/bitstream/20.500.12708/17644/1/Brunnbauer%20Axel%20-%202021%20-%20Model-based%20deep%20Reinforcement%20learning%20for%20autonomous...pdf

https://arxiv.org/html/2408.04198v1#bib.bib11

https://dl.acm.org/doi/pdf/10.5555/3546258.3546526