Selected Topics in Reinforcement Learning

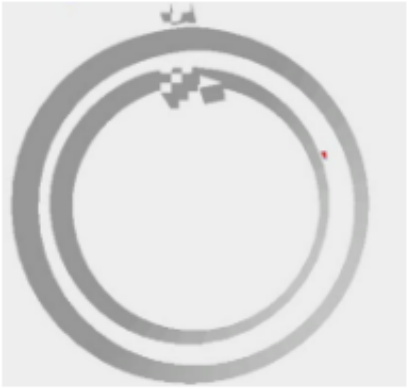
Final Project Report

313554044 黃梓誠

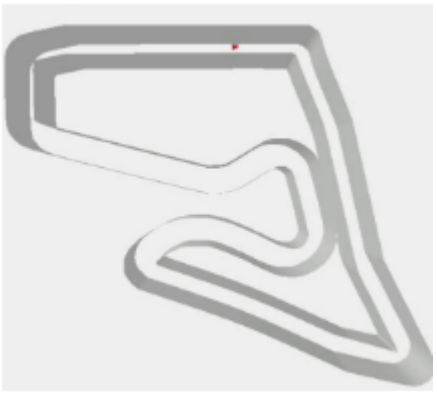
Methodology Introduction

實作策略: 介紹每個地圖中使用的方法

在實作中，由於我想打造出更具有**泛化能力**的 model，也同時拿到 bonus+5 分，因此我採用單一強化學習演算法來實作，在觀察以下兩個賽道後



Map: Circle_cw



Map: Austria

因為圓形賽道行駛模式較為單一即可拿到高分，而 Austria 賽道則需要注意兩個接近 180 度的髮夾彎，並且兩個地圖都有**共同目標**，也就是在有限的時間完成最多的圈數，所以理論上可以用同一個 model 來完成任務。因此我決定在 Austria 賽道上進行訓練，並用最好的 model 對兩個地圖進行測試。

實作方式 PPO continuous

為什麼選擇該方法來應對每個地圖的挑戰 ？

由於 racecar_gym 這個環境連續的 action space，且有鑑於 Lab3、Lab4 使用 PPO 和 TD3 都在遊戲上有不錯的表現，但因為尚未有在 PPO 上實作賽車遊戲的經驗，因此我決定透過 PPO 這個適用於 continuous action space 的 RL Agorithm 來解決本次的任務。

2. Experiment Design and Implementation

(一)神經網路架構設計（層數、輸入/輸出格式等）

我使用 **Stable Baseline 3** 提供的 **CustomCNN** 套件來訓練神經網路，原因是對於 **On-Policy algorithm**（如 **PPO**），**BaseFeaturesExtractor** 可在 agent 之間共享特徵，節省計算資源。

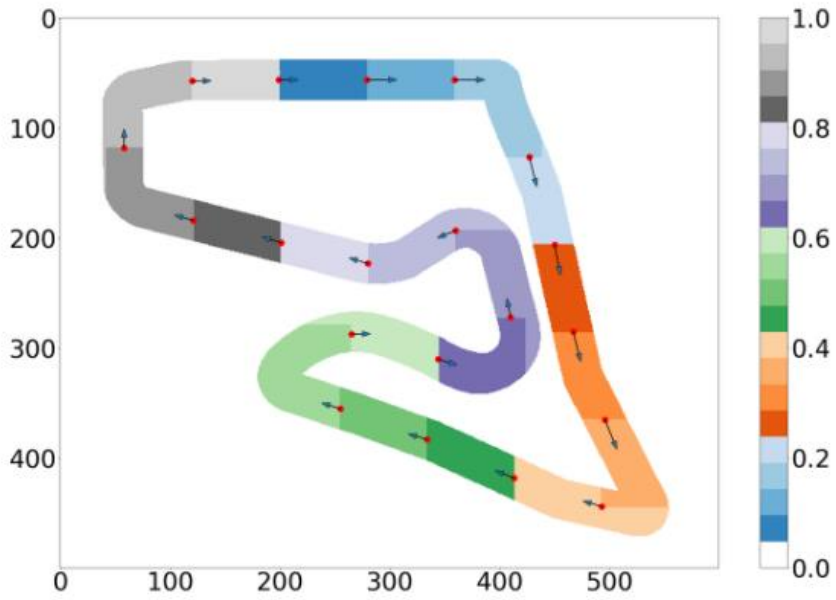
CustomCNN 結構如下：

- **輸入層**: 接收轉為灰階並縮放至 84x84 的影像輸入。
- **卷積層**: 提取影像的空間特徵。
- **全連接層**: 將卷積特徵整合，輸出適用於連續動作空間的馬達和方向盤參數。
- **輸出層**: 提供兩個連續動作值 (馬達動力和方向盤角度)
- **Ref : https://stable-baselines3.readthedocs.io/en/master/guide/custom_policy.html**

```python
class CustomCNN(BaseFeaturesExtractor):
    """
    :param observation_space: (gym.Space)
    :param features_dim: (int) Number of features extracted.
        This corresponds to the number of unit for the last layer.
    """

    def __init__(self, observation_space: spaces.Box, features_dim: int = 256):
        super().__init__(observation_space, features_dim)
        # We assume CxHxW images (channels first)
        # Re-ordering will be done by pre-preprocessing or wrapper
        n_input_channels = observation_space.shape[0]
        self.cnn = nn.Sequential(
            nn.Conv2d(n_input_channels, 32, kernel_size=8, stride=4, padding=0),
            nn.ReLU(),
            nn.Conv2d(32, 64, kernel_size=4, stride=2, padding=0),
            nn.ReLU(),
            nn.Flatten(),
        )
```

(二)訓練過程細節

1. **環境設計**
    - 僅使用 racecar_gym 中的 austria 場景。



Austria's 20 sections of track

- 將影像轉為灰階並縮放 obs 大小，以減少計算資源需求。

  原本: (3,128,128)

```
self.observation_space = gym.spaces.Box(low=0, high=255, shape=(3, 128, 128), dtype=np.uint8)
```

  轉灰階並縮放為: (1,84,84)

```
    env = GrayScaleObservation(env, keep_dim=True)
    env = ResizeObservation(env, 84)
```

Ref :
https://gymnasium.farama.org/main/_modules/gymnasium/wrappers/gray_scale_observation/

2. 訓練資料的蒐集&分布式訓練:

在演算法訓練過程中，資料的蒐集對結果有相當大的影響。同時蒐集不同環境的資料能夠讓 model 探索更多種可能，也有加速訓練的效果。因此我使用 Stable Baselines3 提供的 SubprocVecEnv library。它允許同時執行多個 Gym 環境，並利用 subprocesses 並行收集 trajectories。最後，因為我的電腦有 32 個 logic cpu 因此我同時運行 32 個環境。

3. 疊加影像

使用 Stable Baselines3 提供的 VecFrameStack 將疊加影像，捕捉連續影像的動態資訊，讓模型了解連續影像間的關係。

使用前 obs : (1 , 84 , 84)　　--------->　　使用後 obs : (8 , 84 , 84)

4. 最終呈現:

```
> class GrayScaleObservation(gym.ObservationWrapper, gym.utils.RecordConstructorArgs):···

> class ResizeObservation(gym.ObservationWrapper, gym.utils.RecordConstructorArgs):···


if __name__ == "__main__":
    def make_env():
        env = RaceEnv(scenario='austria_competition_collisionStop',
            render_mode='rgb_array_birds_eye',
            reset_when_collision=False)
        env = ChangeObs(env)
        env = GrayScaleObservation(env, keep_dim=True)
        env = ResizeObservation(env, 84)
        return env
    CPU = 32
    env = SubprocVecEnv([lambda: make_env() for i in range(CPU)])
    env = VecFrameStack(env, 8, channels_order='last')
```

(三)　Reward function 設計:

在設計 reward function 階段 我先分析 State 會輸出什麼資料，以下是該 state 中每個參數解釋 :

車輛位置資訊

1. pose: 表示車輛當前的位置和方向，包含 6 個數值：
    - [x, y, z, roll, pitch, yaw]
    - x, y, z: 車輛在空間中的位置坐標（通常是 3D 坐標系中的位置）。
    - roll, pitch, yaw: 車輛的旋轉角度，描述車輛的方向或朝向。

碰撞相關資訊

2. wall_collision:：True 表示車輛碰到牆壁，False 表示未碰到牆壁。
3. opponent_collisions:記錄與其他車輛發生碰撞的對手車輛信息。

4. collision_penalties:碰撞懲罰記錄；如果數組為空，則表示目前沒有懲罰。
5. n_collision:累計碰撞次數，表示車輛在該次運行中發生的總碰撞次數。

速度相關資訊

6. acceleration:表示車輛的當前加速度，包含 6 個分量：
    - [ax, ay, az, roll_rate, pitch_rate, yaw_rate]
    - ax, ay, az: 車輛在 3 個軸上的線性加速度。
    - roll_rate, pitch_rate, yaw_rate: 車輛的角加速度。

7. velocity:車輛的當前速度，包含 6 個分量：
    - [vx, vy, vz, roll_rate, pitch_rate, yaw_rate]
    - vx, vy, vz: 車輛在 3 個軸上的線速度。
    - roll_rate, pitch_rate, yaw_rate: 車輛的角速度。

賽道進度資訊

8. progress:表示車輛在當前賽道上的進度百分比
    - 例如，0.９９表示已完成賽道的 ９９%。
9. dist_goal:表示車輛距離下一個目標（ex:下一檢查點或終點）的距離。
10. checkpoint:
    - 當前達到的檢查點編號，追踪車輛的進度。
11. lap:
    - 車輛當前處於第幾圈。

時間與方向資訊

12. time:表示當前經過的時間（通常以秒為單位）。
13. wrong_way:車輛是否朝錯誤方向行駛。
    - True 表示方向錯誤，False 表示方向正確。

根據上面的 State 輸出的資訊，我抓其中幾個比較重要的特徵來設計 reward:

1. 速度類型 Reward :

我同時參照車輛當前的速度以及 motor 的大小來增加 reward，以鼓勵車輛以更高速度前進，並且因為 motor 在正負一之間，因此他也有防止車輛倒退的功用。

```
reward += 0.5 * abs(state['velocity'][0])  # 獎勵更高速度
reward += 0.5 * motor_action
```

2. 進度類型 Reward：

當車輛每通過新的 checkpoint 就加 20 分，鼓勵 agent 通過較多 checkpoint。

```
    # checkpoint獎勵
    if state['checkpoint'] > self.previous_info['state']['checkpoint']:
        reward += 20
```

檢查車輛在當前賽道上的進度百分比，因為每個時間推進的比例不多，因此前面乘了一個較大的常數來讓進度 reward 不會太小。

並且如果車輛停止的話就減少 reward 以驅使 agent 前進

```
    # progress 表示車輛在當前賽道上的進度百分比
    if state['progress'] > self.previous_info['state']['progress']:
        reward += 500 * (state['progress'] - self.previous_info['state']['progress'])
    elif state['progress'] == self.previous_info['state']['progress']:   # not moving
        reward -= 0.1
```

3.碰撞類型 Reward

當車輛產生碰撞時就立刻中止且給予較大的懲罰，確保模型能避免無效操作。

```
# 碰撞懲罰
if state['wall_collision'] or state['n_collision'] > 0:
    reward -= 200
    done = True
```

(四) Action Space 設計:

在 racecar_gym 賽車環境中，動作空間由： **motor action** 和 **steering action** 兩個主要參數組成。

---

1. **動作空間設定**

```
# 添加noise
motor_scale = 0.005 # dont too big
steering_scale = 0.01
motor_action = np.clip(motor_action + np.random.normal(scale=motor_scale), -1., 1.)
steering_action = np.clip(steering_action + np.random.normal(scale=steering_scale), -1., 1.)
```

為了模擬現實中的控制誤差

我對動作加入高斯 noise，該 noise 的設計是為了**模擬現實中的不確定性**，模擬現實駕駛過程的控制誤差，使得學習到的策略更加穩健。

其中 motor_scale 和 steering_scale 分別設計如下：

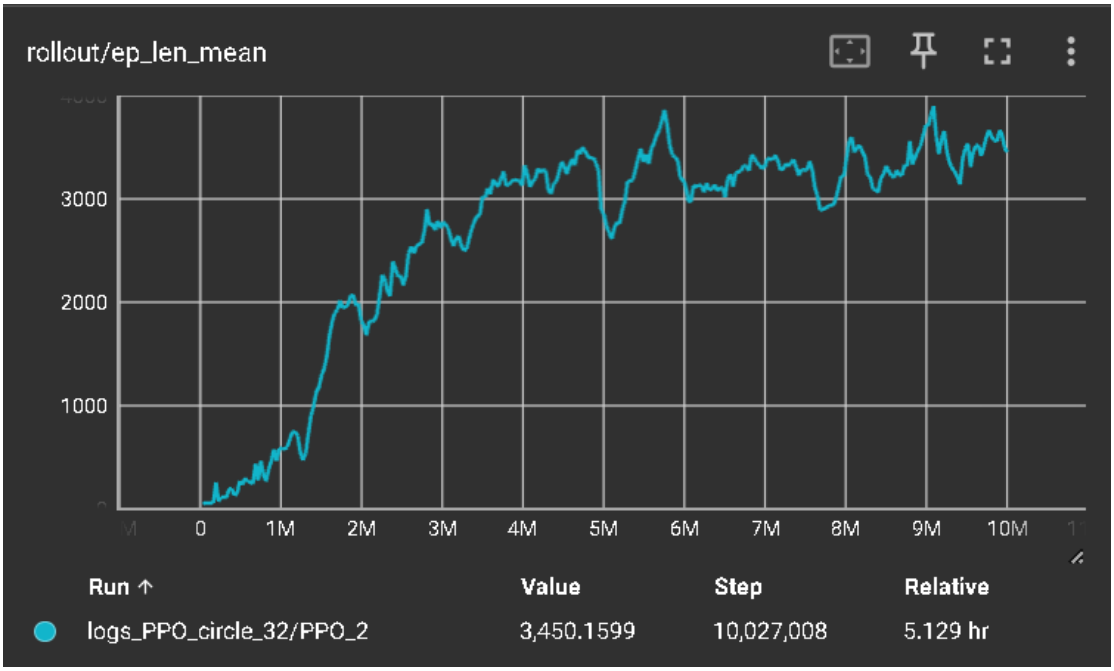
- noise 的標準差，分別對應馬達和方向盤動作的隨機變化幅度。
- 因為我不希望 motor 的變化太大導致意外沒辦法通過髮夾彎，因此 **motor_scale** 的 noise 設計的較小
- 使用 np.clip 將值裁剪，保證動作仍然在 [-1, 1] 範圍內。

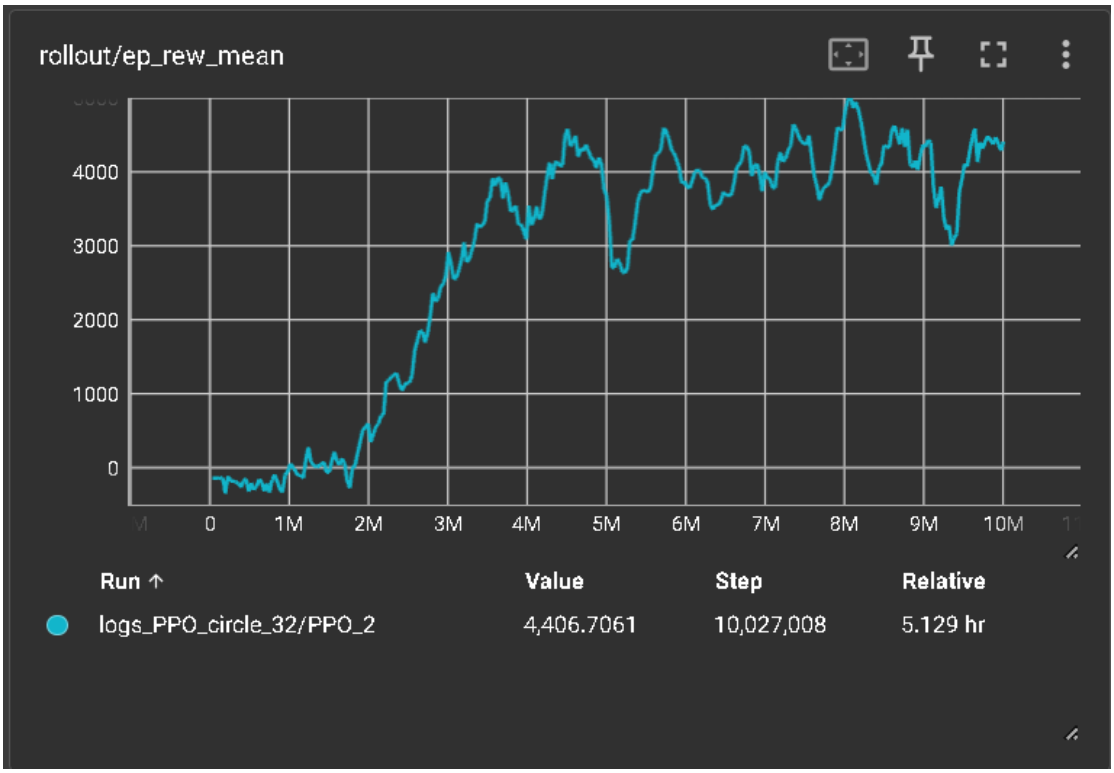
(五)模型檢查點與監控:

- 設定 callback 函數，每訓練 2048 step 儲存模型權重。

```
class LoggingCallback(BaseCallback):
    def __init__(self, check_freq, save_path, verbose=1):···

    def _init_callback(self):
        if self.save_path is not None:
            os.makedirs(self.save_path, exist_ok=True)

    def _on_step(self):
        if (self.calls) % self.check_freq == 0:
            model_path = os.path.join(self.save_path, '{}'.format(self.calls))
            self.model.save(model_path)
        return True
```

- 使用 TensorBoard 進行訓練監控和可視化。


rollout/ep_len_mean
3000
2000
1000
0
1M
2M
3M
4M
5M
6M
7M
8M
9M
10M
Run
Value
Step
Relative
logs_PPO_circle_32/PPO_2
3,450.1599
10,027,008
5.129 hr


rollout/ep_rew_mean
4000
3000
2000
1000
0
1M
2M
3M
4M
5M
6M
7M
8M
9M
10M
Run
Value
Step
Relative
logs_PPO_circle_32/PPO_2
4,406.7061
10,027,008
5.129 hr

(六)實驗主要超參數設定：

我嘗試了不同的 hyperparameter，參數落在以下區間，後續會介紹不同參數設定的結果

- **Learning Rate**: 2e-4 ~ 1e-5
- **Discount factor (Gamma)**: 0.99 ~ 1
- **Batch Size**: 64 ~ 128
- **更新頻率 (n_steps)**: 1024
- **熵正則化係數 (Entropy Coefficient)**: 0.005
- **Clip Range**: 0.1 ~ 0.3
- **Total Timesteps**: 5e6 ~ 2e7

(七)使用工具、框架及資源

- **Stable-Baselines3**: 強化學習演算法套件。
- **PyTorch**: 提供 NN 建立與訓練框架。
- **gymnasium**: 環境管理工具。
- **racecar_gym**: 賽車環境模擬工具。
- **OpenCV**: 影像處理。
- **SubprocVecEnv**: 用於處理多環境訓練。

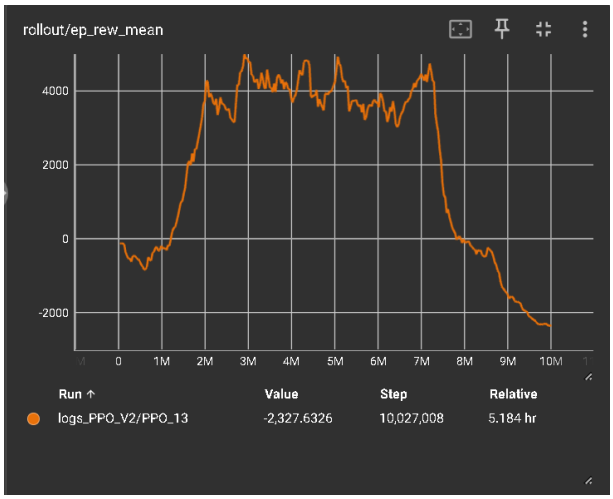
3. 方法比較與評估 (Method Comparison and Evaluation)

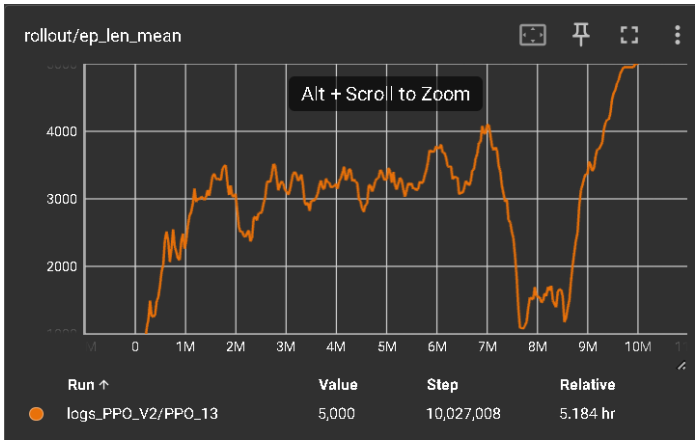
方法一：

一開始我在 reward function 中使用 checkpoint, velocity , motor_action , 以及 dist_goal 當作獎勵，目的是獎勵高速度並且可以利用 dist_goal 來判斷是否要過彎。

```
        # 檢查點獎勵
        if state['checkpoint'] != self.prev_info['state']['checkpoint']:
            reward += 15

        reward += 0.5 * abs(state['velocity'][0])  # 獎勵更高速度
        reward += 0.5 * motor_action
        reward -= 0.1 * (abs(motor_action - self.prev_info['motor']) + abs(st

        # 過彎平滑獎勵
        if state['dist_goal'] > 0.4:  # 假設目標距離大代表在彎道
            reward += 0.1 * (1 - abs(state['velocity'][0]))  # 獎勵減速過彎
            reward += 0.1 * (1.1 - abs(state['pose'][2]) ) # 獎勵平滑轉向


        # 碰撞懲罰
        if state['wall_collision'] or state['n_collision'] > 0:
            reward -= 200
            done = True
```

結果:





結論:

一開始 model 有正常訓練，但在 8M 左右急遽下降，最後我發現原因是我誤判 dist_goal 的定義。dist_goal 指的是:和下一個 checkpoint 的距離而非之後跑道的彎度。

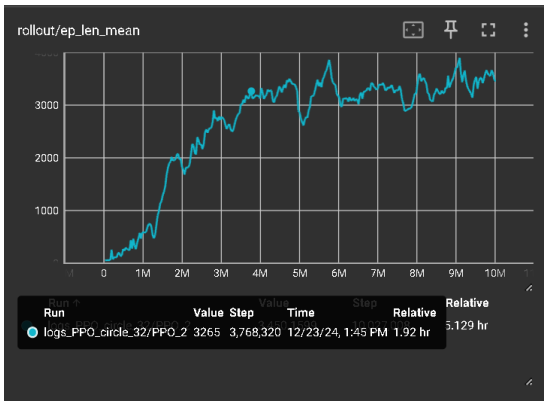
除此之外，我檢查 5~6M 處的 model 是可以跑起來的，代表加速以及 checkpoint reward 確實起到了幫助訓練的效果。

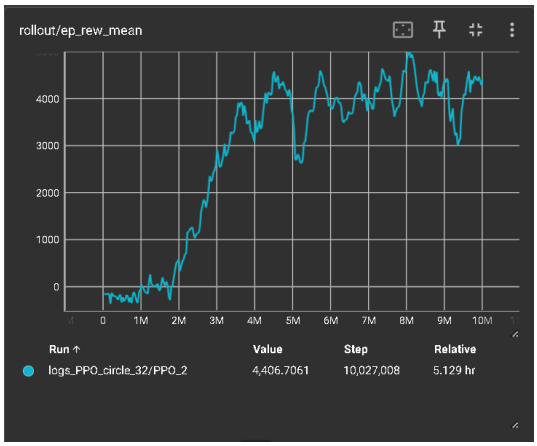
方法二 ：

有鑑於方法一的定義錯誤，第二版的 reward 我將方向判斷移除，但保留速度類型 reward 跟 checkpoint 類型 reward

```python
# 速度類型 reward
reward += 0.5 * abs(state['velocity'][0])  # 獎勵更高速度
reward += 0.5 * motor_action


# 進度類型reward

# checkpoint獎勵
if state['checkpoint'] > self.previous_info['state']['checkpoint']:
    reward += 20

# progress 表示車輛在當前賽道上的進度百分比
if state['progress'] > self.previous_info['state']['progress']:
    reward += 500 * (state['progress'] - self.previous_info['state']['progress'])
# 懲罰不移動
elif state['progress'] == self.previous_info['state']['progress']:
    reward -= 0.1

# 碰撞懲罰
if state['wall_collision'] or state['n_collision'] > 0:
    reward -= 200
    done = True
```

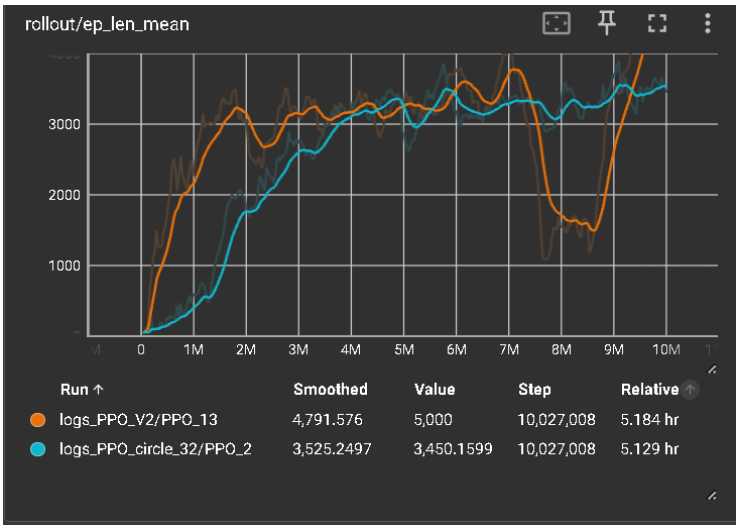
結果:

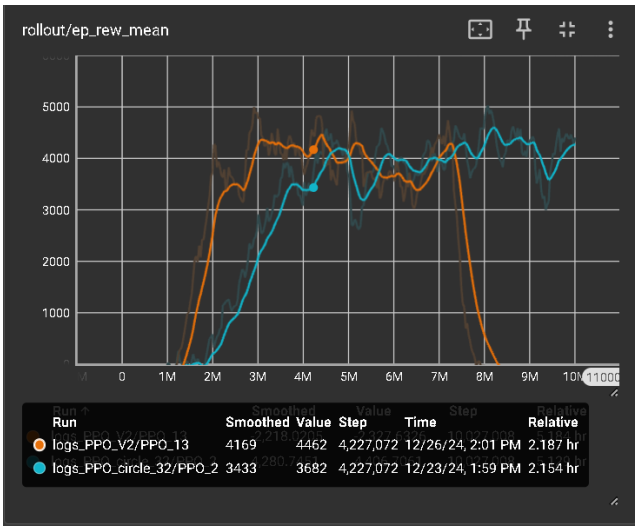




不同 Model reward 的比較

和第一版相比，模型在後期訓練較為穩定，但我也發現後期訓練成果持平，而 reward 跟 len_mean 都開始震盪，代表模型在此處已經無法繼續收斂。





Fine_tune 方法二 ：

接著，因為我發現模型開始震盪，我認為是 hyperparameter 需要調整才能繼續訓練下去，因此我從方法二的參數來 fine_tune

原先方法二(藍線)的超參數設定為:
lr=1e-4, n_steps=1024, batch_size=64, n_epochs=10, clip_range=0.1, ent_coef = 0.05

為了讓賽車更穩定，我選擇加大 batch size 並調整 learning rate 以及降低 ent_coef(降低探索程度)，並固定 vf_coef = 0.5 降低 value function 的影響力，

最後我 fine tune 了四種不同的超參數，以下是他們的設定:

(黃線)：提升 learining rate,降低 ent_coef

lr = :2e-4 vf_coef = 0.5.batch_size = 128 clip_range = 0.2 ent_coef = 0.02

(紫線)：些微提升 learining rat e,降低 ent_coef

lr = 1.5e-4 vf_coef = 0.5.batch_size = 128 clip_range = 0.2 ent_coef = 0.02

(橘線)：使用原始 learining rate, 但大幅降低 ent_coef

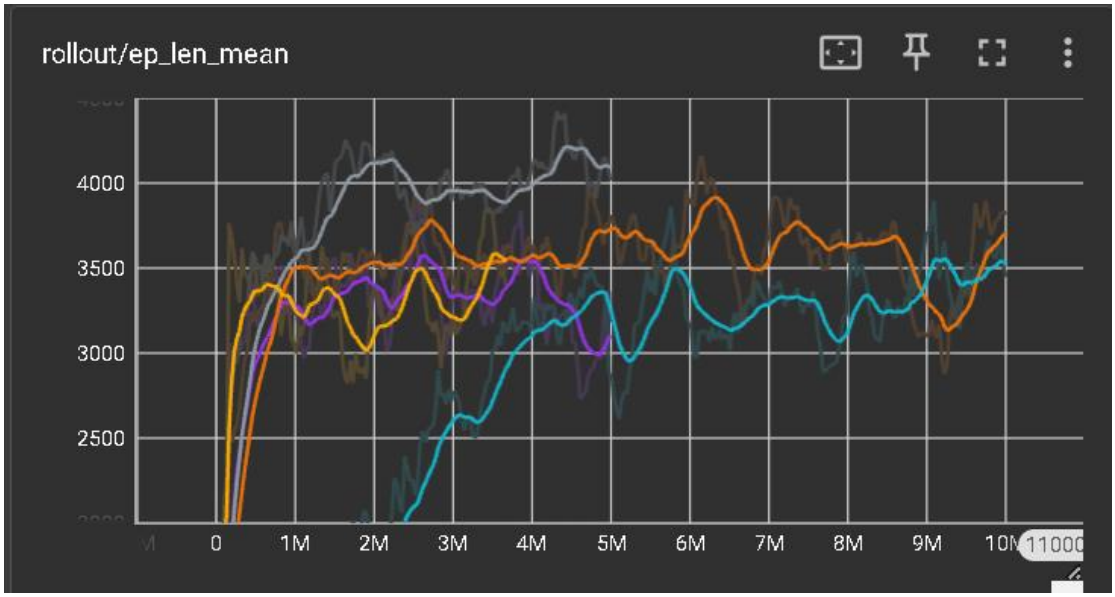
lr = 1e-4 vf_coef = 0.5 batch_size = 128 clip_range = : 0.2　ent_coef = 0.015

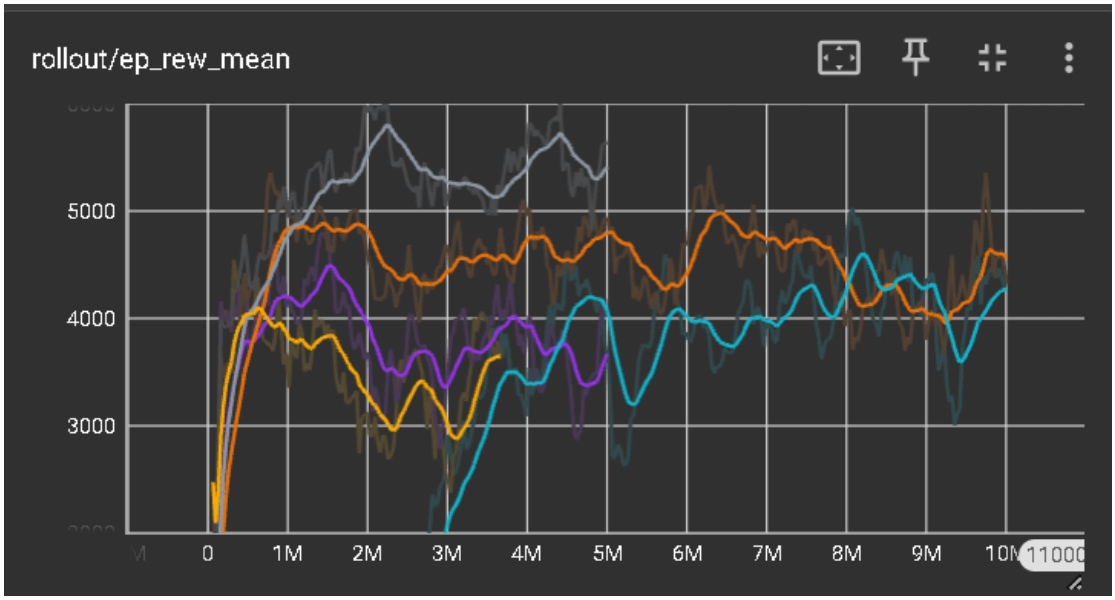
(灰線)：調降 learining rate, 以及大幅降低 ent_coef

lr = 1e-5　　vf_coef = 0.5 batch_size = 128 clip_range = 0.3　ent_coef =　0.005

Fine tune 結果比較：

從結果可以看到灰線以及橘線結果較好，而黃線以及紫線結果甚至不如最初的方法二，因此可以知道**增加 Batch Size 的確能平滑梯度更新**，但最重點應是 lr 的設定以及**適中的探索係數 ent_coef** ，而雖然在分數上灰線相較於橘線來的高，但同時他的 len_mean 也比較高，len_mean 是完成的時間記數，若該值較高，**除了代表遊玩較久之外，也可能是長時間作出無意義的動作**，並且實際測試時的確橘線的表現最好，而灰線則表現不太穩定。因此權衡之下，我選擇橘線的 model 作為我最後 Demo 的模型。





4. 挑戰與學習點 (Challenges and Learning Points)

挑戰(一)賽道環境熟悉:

- 一開始對於 racecar_gym 環境不夠熟悉，尤其是 3D 資料中包含的多種速度、位置與碰撞資訊，導致在實作上花了相當多時間理解各個參數的意義與其對訓練流程的影響。

學習點:

1. 我學會如何觀察環境輸出（如 pose, wall_collision, velocity, progress 等）以及如何透過 API 呼叫或修改，再嘗試設定多環境並行 (如: SubprocVecEnv)以加速資料蒐集。

2. 在程式碼、註解上做好紀錄，避免日後維護或調整時混淆。

挑戰(二) reward Function 設計與調整:

- 早期嘗試各種方式改寫原本的 reward function，如加入速度、checkpoint、motor 的懲罰或加成等，但若 scaling 不當，可能造成 PPO 的 critic 輕易收斂到次佳解，或是 reward 數值過小，讓模型失去學習動機。
- 同時也曾誤用 dist_goal（與下一個 checkpoint 的距離）來判定過彎，卻導致訓練後期 reward 曲線急遽下降，發現是 dist_goal 跟實際髮夾彎處的“彎度”判斷並不相同。

學習點:

- 在 reward 設計中應該**分解任務目標**，合理區分「速度類型獎勵」、「進度類型獎勵」、「碰撞處罰」等；每個部分都應有清楚的設計初衷、權重與上限/下限，避免彼此牴觸。

挑戰(三) 髮夾彎（Hairpin Bend）的處理:

- 在 Austria 賽道上，兩個幾近 180 度的髮夾彎尤其難以通過。嘗試調低整體速度雖能提升通過彎道的成功率，但也犧牲了整體圈數與最終分數。
- 有時模型會出現劇烈轉向或減速過程太長等不穩定策略，造成得分波動。

學習點:

- 透過助教提醒可以隨機設定初始位置，讓模型在不同位置重新開始，能大幅增加探索空間，最重也成功通過了髮夾彎等極端情況。

```
def reset(self, *args, **kwargs: dict):
    if kwargs.get("options"):
        kwargs["options"]["mode"] = 'random'
    else:
        kwargs["options"] = {"mode": 'random'}
    self.cur_step = 0
    obs, *others = self.env.reset(*args, **kwargs)
    # print("Initial Observation:", obs)
```

5. 未來工作 (Future Work) - 佔報告分數的 10%

在 fine tune 的過程中，我比較了頂尖賽車手與 PPO 賽車 model 的區別

技能	頂尖賽車手	PPO 賽車 model
良好的駕駛技巧	Yes	Yes
知道何時該如何行動	完全習得	部分習得
知道地圖該怎麼跑	完全習得	部分習得

簡單的列出表格後，我發現不同的 model 同時具有幾個特性

1. 即使是最好的 model 他過彎時仍像是看到彎道才緊急轉彎

2. 直線加速並不夠快，而彎道又太慢過去

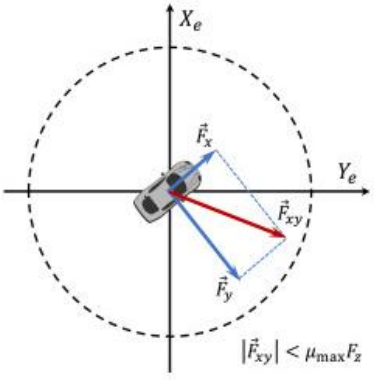
我想，造成有所不同的原因，應該是 PPO model 僅僅是記住”在看到某場景時應該作什麼”，雖然在環境建立時已經透過疊 8 個 frame 讓模型補捉動態畫面，但仍然不夠全面；相較之下，頂尖賽車手跑過該地圖後會擁有**整張地圖的記憶，而非只有幾幀畫面**，我認為這就是模型本身效能有所限制的原因。

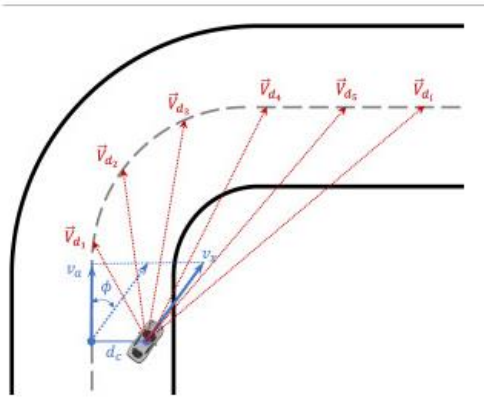
接著，在 Demo 時我發現成績不如我測時時理想，我認為可能是 noise 或是 obs 帶來的影響，綜合上述的分析，我的 Future Work 包含以下幾點

1. 嘗試[1] [3]的方法使用 Transformer or LSTM 捕捉更長期的地圖關係

2. 嘗試[2]作者提出的 action mapping 讓模型能夠在不同條件下也能穩定行駛

[2]的作者提出結合 action mapping 機制來處理由輪胎-路面摩擦引起的狀態依賴關係，並提出一種數值近似方法來實現 action mapping，從而提高在不同摩擦條件下的駕駛策略泛化能力。





Reference:

[1]Exploring Transformer-Augmented LSTM for Temporal and

Spatial Feature Learning in Trajectory Prediction

[2] Learning autonomous race driving with action mapping reinforcement learning

[3] "PredFormer: Transformers Are Effective Spatial-Temporal Predictive Learners"