Artificial Intelligence

Homework 5

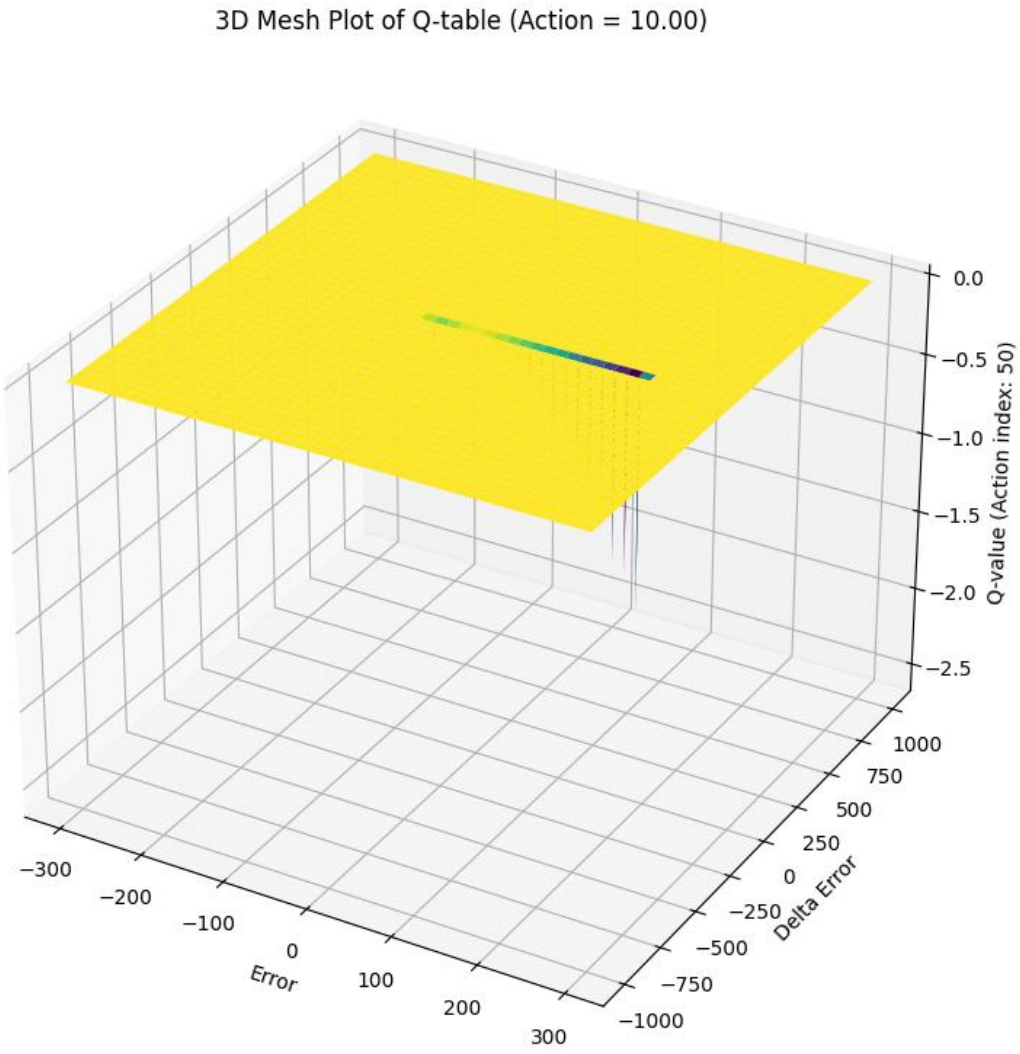
Reinforcement learning

姓名: 劉育辰

學號:110303585

系級:機械 4C

A. the 3D mesh plot of the finished Q-table



上圖為 Q-TABLE 的 3D 圖，可以發現真的被用到的狀態空間只有一小塊，因為在對 e 和 de 的 linespace 切割的不夠精細，又 e 和 de 的數值取到小數點後好幾位，真的有變化的是小數點後 6-7 位數的部分，導致離散化後只集中在特定幾個狀態空間內，所以只有這幾個狀態空間對應的 Q-TABLE 真實被用到。

B. the control and simulation code

```
error_bins = np.linspace(-300, 300, 301)
delta_error_bins = np.linspace(-1000, 1000, 301)
actions = np.linspace(0, 20, 51)
Q_table = np.zeros((len(error_bins), len(delta_error_bins), len(actions)))

omega_train_record = []
target_episodes = [10, 20, 30, 40, 50]


def discretize(value, bins):
    return np.digitize([value], bins)[0] - 1
```

這段先對 state = (error, delta error) 狀態空間作定義，和 Q-TABLE 作初始化。並定義動作空間 action(u(k))。Discretize 這個 function 是用來離散化輸入的 value，將 value 插入對應的 linspace 內。

```
def motor_model(x, u):
    i, omega = x

    # 限制輸入電壓
    u = np.clip(u, 0, 10)

    # 計算 di/dt 與 dω/dt
    di_dt = (1/L) * (-R * i - Ke * omega + u)
    di_dt = np.clip(di_dt, -1e4, 1e4)  # 防止數值爆炸

    domega_dt = (1/J) * (-B * omega + Kt * i)
    domega_dt = np.clip(domega_dt, -1e3, 1e3)

    i_next = i + Ts * di_dt
    omega_next = omega + Ts * domega_dt

    omega_next = np.clip(omega_next, -200, 200)

    return np.array([i_next, omega_next])
```

該函示對輸入的參數 X = [I, omega], u 做計算到下一步 Q(S, a')的數值，在實驗中有發現有數值浮動很大的情況，所以用 np.clip()來對數值作約束，防止數值爆炸。

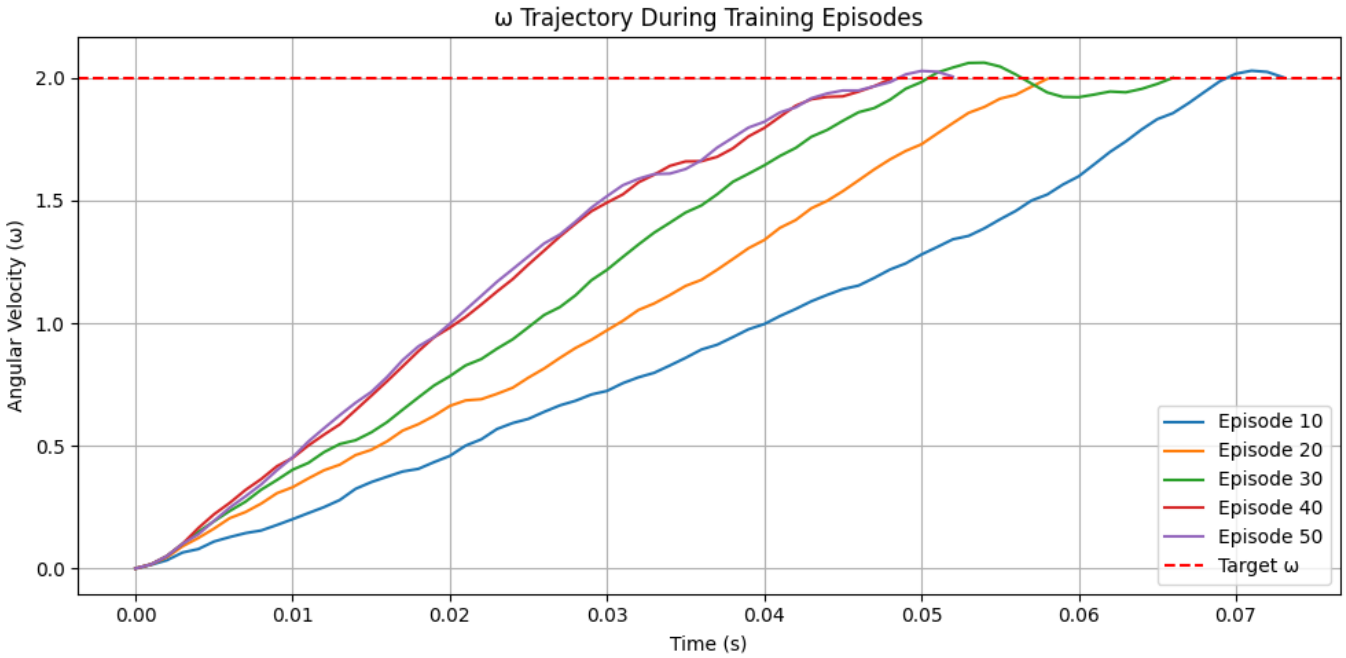
```python
for episode in range(episodes):
    x = np.array([0.0, 0.0])
    e_prev = 0

    omega_history = []

    for t in range(1000):
        omega = x[1]
        e = omega_targ - omega
        de = e - e_prev
        e_prev = e

        SCALE_E = 100
        SCALE_DE = 100

        e_scaled = np.clip(e * SCALE_E, -3 * SCALE_E, 3 * SCALE_E)
        de_scaled = np.clip(de * SCALE_DE, -10 * SCALE_DE, 10 * SCALE_DE)

        s1 = discretize(e_scaled, error_bins)
        s2 = discretize(de_scaled, delta_error_bins)

        # ε-greedy 策略
        if np.random.rand() < epsilon:
            a_idx = np.random.randint(len(actions))
        else:
            a_idx = np.argmax(Q_table[s1, s2])
```

這段是根據 PPT 上 Q-learning 得步驟來做 RL 的模擬，先根據每一次的的 episodes 做 1000 步的計算，一開始先根據現在的 state(e, delta_e) 得到 Q(s,a)，並從現在的 Q-TABLE 中選一個最好的 ACTION。這裡在選擇策略上是根據 ε-greedy ，會有 10%機會隨機給 ACTION，90%的機會選擇 Q(S,a)值最大的 ACTION 來做。

```
        u = actions[a_idx]
        x_next = motor_model(x, u)

        omega_next = x_next[1]
        e_next = omega_targ - omega_next
        e_next = np.clip(e_next, -3, 3)

        reward = -e_next**2

        s1_next = discretize(e_next, error_bins)
        s2_next = discretize(e_next - e, delta_error_bins)

        # Q-learning 更新
        Q_table[s1, s2, a_idx] += alpha * (
            reward + gamma * np.max(Q_table[s1_next, s2_next]) - Q_table[s1, s2, a_idx]
        )

        x = x_next
        omega_history.append(x[1])

        # debug 印出
        print(f"u = {u:.1f}V, i = {x[0]:.2f}A, ω = {x[1]:.5f}rad/s, e = {e}, de = {de}")

        if abs(e_next) < 0.01:
            print(f"收斂：step {t}, ω = {x[1]:.5f}")
            break
    if (episode + 1) in target_episodes:
        omega_train_record.append(omega_history)
```

這一段落，會根據剛剛 ε-greedy 選到的 ACTION 來給 motor_modle(x,u)，這函式會根據現在狀態的電流、omega 和電壓(u)，來得到下一步的狀態 Q(S,a')，最後根據公式更新 Q-TABLE 上的值和現在的狀態 S(e,delta_e)。同時我有紀錄每次步數 omega 的變化，以利於之後畫圖分析 omega 得變化。

C. simulation of controlling results

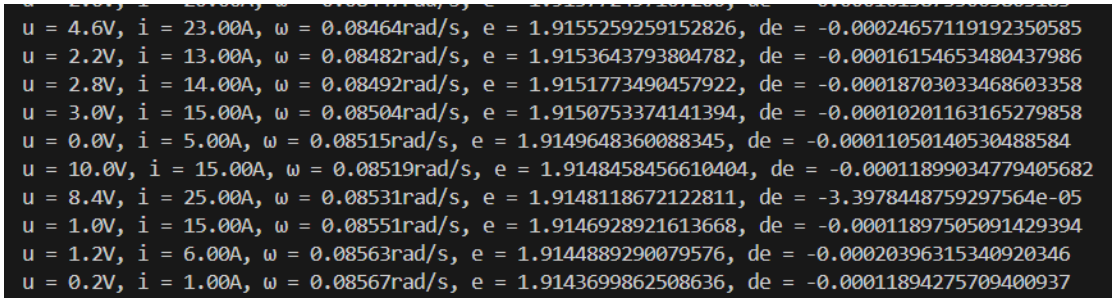


從結果圖可以知道馬搭大約花到 0.05 秒後才會穩定達到 2.0rad/s 的轉速，且成長數度很穩定，趨近線性成長，我在模擬的結束機制是「如果 e 的值小於 0.01 時代表收斂」。所以當輸出差異都是在 2.0 正負 0.01 值左右代表馬達的轉速已趨於穩定。

D. Discussions

用上圖的結果來討論，我總共跑 50 次 Episode，每 10 次紀錄 omega 的變化，可以看出第一個 10 Episode(藍色線)，花了約 0.07 秒才達到穩定狀態，但隨著模擬次數增加，Q-TALBE 不斷跟新，策略選擇的 ACTION 越來越精準，到第 50 次只需要 0.05 秒，比第 10 次快 0.02 秒。但還有一個值得注意的是第 30 次，這次其實就能在 0.05 秒達到目標轉速，但是卻花了 0.01 秒左右才趨於穩定，可以知道機器找到最快速達到目標的 ACTION 策略，但在後續的穩定上還不夠好，要再繼續更新 Q-TABLE 內的值。

再來是系統變數上我有自己做改動，J = 0.01，如果用原本的數值 J = 2，會導致慣量很大，要很長時間才能轉起來，每次的 omega 都是如下圖，可能要花更多的 episodes 和計算時間，我感覺我的電腦可能會起飛，為了減少計算時間所以修改成 0.01。



E. Conclusions

就這次實驗作業比上次的還更要花時間來對模型環境去做設定，在對狀態空間去做大小的定義就花了很多時間，為了要讓狀態空間能被有效利用，但是從 Q-TABLE 的圖可以看出來還是不夠好，爾且也不是說花更多 episodes，就能被有效利用，因為最終機器會找到最好的 ACTION 策略來做，花的時間只會更少，收斂的時間更快。