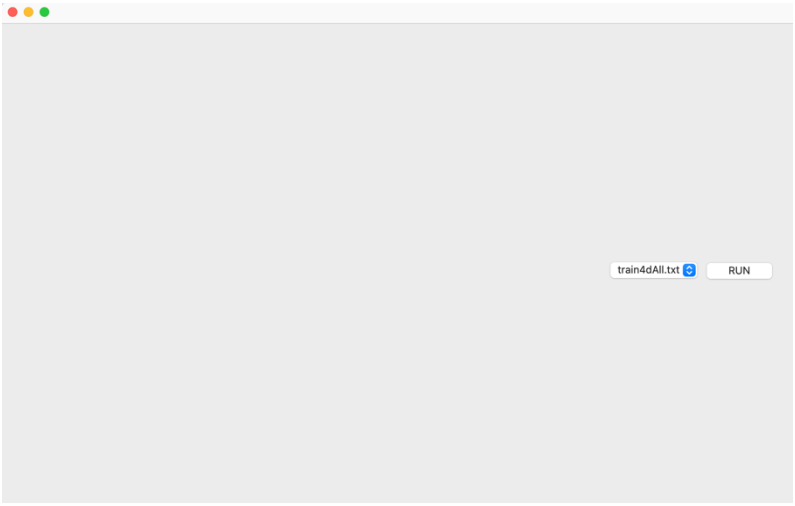
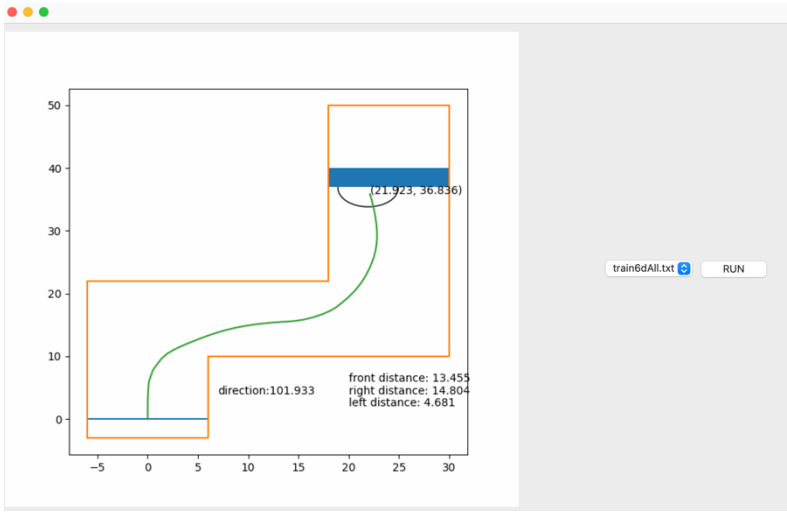
作業二－自駕車 書面報告

1. 程式介面說明

程式一開始執行時會有以下的介面。ComboBox 是讓使用者選擇要訓練哪一個資料集(“Train4dAll.txt”或”Train6dAll.txt”)。選擇完後按下右邊的 RUN 按鈕就可以開始訓練資料。訓練完的資料會以 GIF 檔呈現。GIF 檔則包含了軌道、自走車。自走車的位置會顯示在車子的中心點，方向及偵測到的左、前、右距離則顯示在檔案的右下角。方向是自走車和水平軸的角度。





2. 程式碼說明

這份作業總共分了 6 個 Python 檔案來完成。

a. kmeans4d.py / kmeans6d.py

kmeans4d.py 和 kmeans6d.py 只差在載入 ground truth 值時，提取資料集的行索引值的不同。若訓練資料集是 4 維的，ground truth 值則載入第 3 行的資料，若訓練資料集是 6 維的，ground truth 值則載入第 5 行（從第 0 行開始算）的資料。此兩份程式都是在為 RBFN 初始化鍵結值、中心點及標準差做準備。

先利用類別 kmeans4d(kmeans6d)載入訓練資料集，並將載入的資料先轉換成浮點數的型態。接著設定好要分類的群數後，就將資料放入另一類別 KmeansClustering 實作 Kmeans 來分群及找到每個群體的中心點。另外在 KmeansClustering 類別就先設好分群的最大次數，當分群次數達到最大次數就不再進行分群，或是當前一次分群的中心點和新一次分群的中心點一樣，也將停止分群。

b. RBFN4d.py / RBFN6d.py

若是 RBFN4d.py 則匯入 kmeans4d 的模組且創建一個類別名為 RBFN4d，反之若是 RBFN6d.py 則匯入 kmeans6d 的模組且創建一個類別名為 RBFN6d。這兩個類別都先初始化 rbfn 的 bias=-1，學習率=0.01，epoch=1000。標準差及鍵結值的維度為 kmeans 的分群數且一開始都先設為 0。接著進入到函示 neuron_init。在 neuron_init 函式中，迭代每個群體，分別將類神經元的鍵結值設定為群體中的平均 ground truth 值，標準差為群體中每個點到中心點的平均

距離。接著就進入到 training 函式。在 training 函式中，會依照 epoch 值來設定調整參數的次數。每一次的 epoch 循環皆會跑過所有的訓練資料，每跑一個訓練資料就調整一次參數。參數調整的方式如下：

w: 鍵結值 / m_j: 第 j 個神經元的中心點 / σ_j: 第 j 個神經元的標準差 / θ: bias

$$\underline{w}(n+1) = \underline{w}(n) + \eta(y_n - F(\underline{x}_n))\underline{\varphi}(\underline{x}_n)$$

$$\underline{m}_j(n+1) = \underline{m}_j(n) + \eta(y_n - F(\underline{x}_n))w_j(n)\varphi_j(\underline{x}_n)\frac{1}{\sigma_j^2}(\underline{x}_n - \underline{m}_j(n))$$

$$\sigma_j(n+1) = \sigma_j(n) + \eta(y_n - F(\underline{x}_n))w_j(n)\varphi_j(n)\frac{1}{\sigma_j^3}\left\|\underline{x}_n - \underline{m}_j(n)\right\|^2$$

$$\theta(n+1) = \theta(n) + \eta(y_n - F(x_n))$$

在調整參數的過程中會需要算到資料的活化函數值($\varphi(x_n)$)及目前的 rbfn 輸出值，此時就會將資料傳入 activate 函式和 output 函式做計算:

activate 函式：$\varphi_j(x) = e^{-\frac{\|x-m_j\|^2}{2\sigma_j^2}}$ / output 函式：$\sum_{j=1}^{J} w_j\varphi_j(x) + \theta$

c. env.py

在這個檔案中創建兩個類別：Car 和 Wall。Car 代表自走車，Wall 代表軌道。在 Car 中的值有 pos, phi, radius，分別代表自走車的 xy 座標位置、和水平軸的夾角角度、半徑。Car 中的 move 函式在更新自走車的 xy 座標位置、和水平軸的夾角角度。sensor_dist 函式在測量自走車前方的三個 sensor 測到的前、左右距離。在此函式中，先將 sensor 的角度取出。正前方的 sensor 角度即為自走車和水平軸的夾角角度，左方的 sensor 角度為正前方 sensor 角度加 45 度，右方的 sensor 角度為正前方 sensor 角度減 45 度。每個 sensor 在量測距離時都要遍歷過每個軌道取得和軌道的交點。若有交點，則計算交點與自走車的距離。最後取有最小距離的交點當作自走車的 sensor 所感測到的點，並記錄此最小距離於陣列中。check_wall 函式則是在檢查自走車有沒有碰到軌道，如果有則回傳 True，反之則回傳 False。

在 Wall 中有 start 值來表示軌道的起始 xy 座標位置，end 值表示軌道的終點 xy 座標位置，vector 就是軌道的向量。在 radar_intersect 中，pos 為自走車的 xy 座標位置，vector_radar 為 sensor 角度的向量。sensor 和軌道的交點可以利用兩個向量方程式相等的結果求出：p1+L1*t1=p2+L2*t2(p1,p2 為兩向量的起始點，L1,L2 為自走車的 sensor 和軌道的向量)。因為交點要在軌道的範圍裡，所以 t2 需要大於 0 且小於 1，而交點要在車子前方，所以 t1 要大於 0。滿足了這兩個條件就可以算出 sensor 和軌道的交點。

d. car_run.py

在這份檔案中創建一個名為 Running_Car 的類別，這項類別為記錄自走車實際行走的位置、方向等等的資訊。這個類別中有個 run 函式，傳入參數”dataset”，若 dataset=0 則表示這次的訓練資料集為”train4dAll.txt”，因此所要訓練的 rbfn 模型為”RBFN4d”。反之若 dataset=1 則表示這次的訓練資料集為”train6dAll.txt”，因此所要訓練的 rbfn 模型為”RBFN6d”。接著讀取「軌道座標點」檔案，設定自走車的開始位置、和水平軸的夾角角度和記錄軌道位置。利用 while 迴圈模擬自走車。當自走車進入到終點區域或碰到軌道就跳出迴圈，若此兩個跳出迴圈的條件都沒有達到則繼續走。若 dataset=0，自走車會利用 sensor_dist 函式取得 3 個 sensor 量測到的距離當作 rbfn 的輸入取得方向盤角度。若 dataset=1，自走車除了利用 3 個 sensor 量測到的距離還會利用車子的 xy 座標位置當作 rbfn 的輸入來取得方向盤角度。自走車在最後會利用此方向盤角度和 move 函式來更新 xy 值。

而自走車每跑一次就會紀錄目前的資料：

record 陣列是用來記錄自走車每一次測量到的前、右、左及方向盤角度（若 dataset=1 還會再紀錄自走車的 xy 座標位置），再將 record 陣列加進 track 陣列中。plots 陣列會記錄自走車的 xy 座標位置、方向、三個 sensor 測量到的左、前、右距離。plots 陣列是為了之後畫圖做準備的。在最後會看 dataset 的值是 0 或 1 來決定要將 track 所紀錄的資料寫在”track4D.txt”還是”track6D.txt”。

e. animation.py

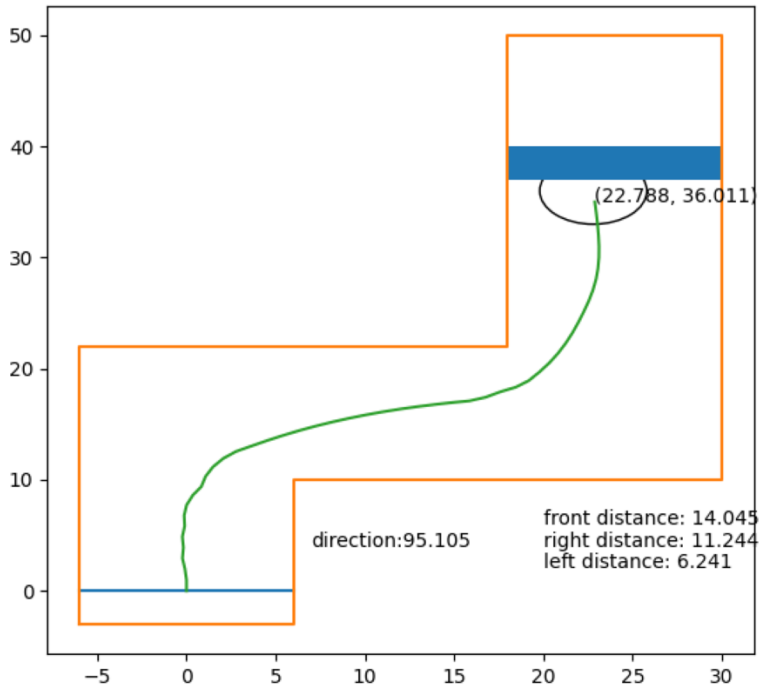
裡面只有一個 drawing 函式。此函式會將參數 dataset 傳進 car_run 的 Running_Car 類別中，再呼叫 car_run 的 run 函數開始模擬自走車。接著利用「軌道座標點」檔案開始畫軌道、終點區域、開始線及自走車。再利用 Running_Car 的 plots 陣列取得自走車的軌跡、phis 陣列取得自走車每走一次的方向角度、distances 陣列取得自走車的 3 個 sensor 測量到的距離。在子函式 Animate 中會更新自走車目前的行走路徑、自走車的中心點、自走車的方向、自走車的 3 個 sensor 測量到的距離。最後再依照 dataset 的值決定存入動畫的檔案。若 dataset=0，將動畫存為”train4d.gif”，若 dataset=1，則將動畫存為”train6d.gif”。

f. chooseUI.py

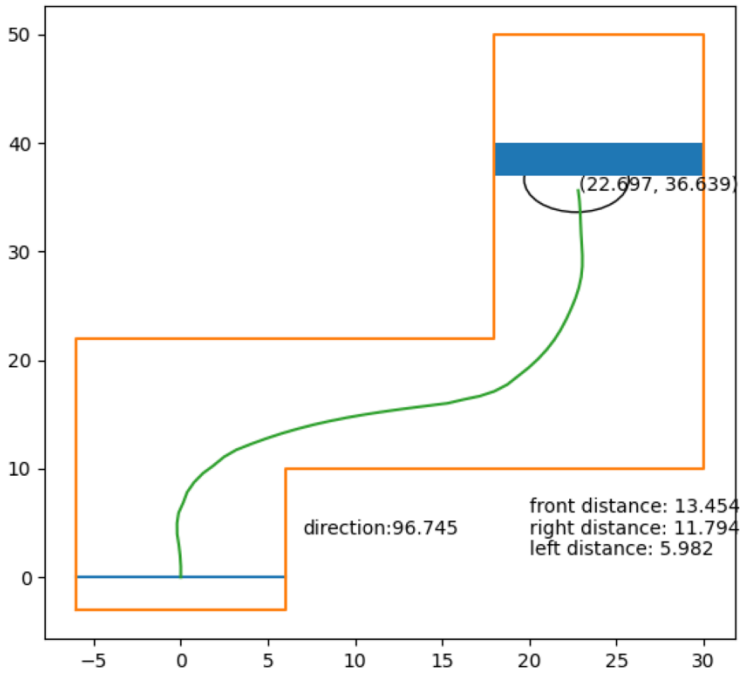
此為控制 GUI 介面的檔案。使用者按下按鈕後，程式會記錄使用者選擇的訓練資料集，再將此資料及傳入函式 drawing 中，開始模擬自走車。再依照選擇的資料集決定要將動畫元件放入哪個檔案(“train4d.gif” 或”train6d.gif”)。”self.graph.Play()”會讓 gif 檔案在 GUI 畫面中播放。

3. 實驗結果

a. 訓練資料集：train4dAll.txt , learning rate=0.01, epoch=1000

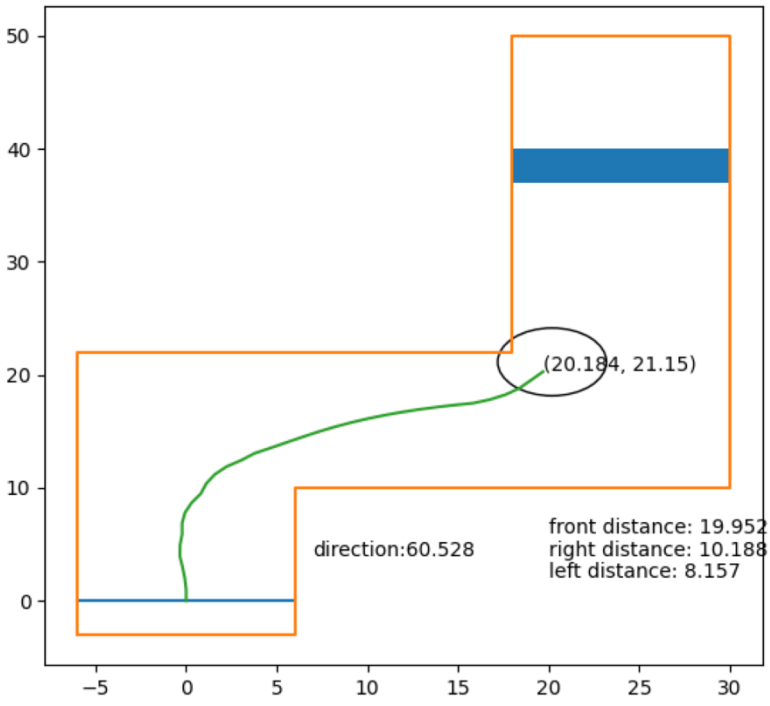


b. 訓練資料集：train6dAll.txt, learning rate=0.01, epoch=1000

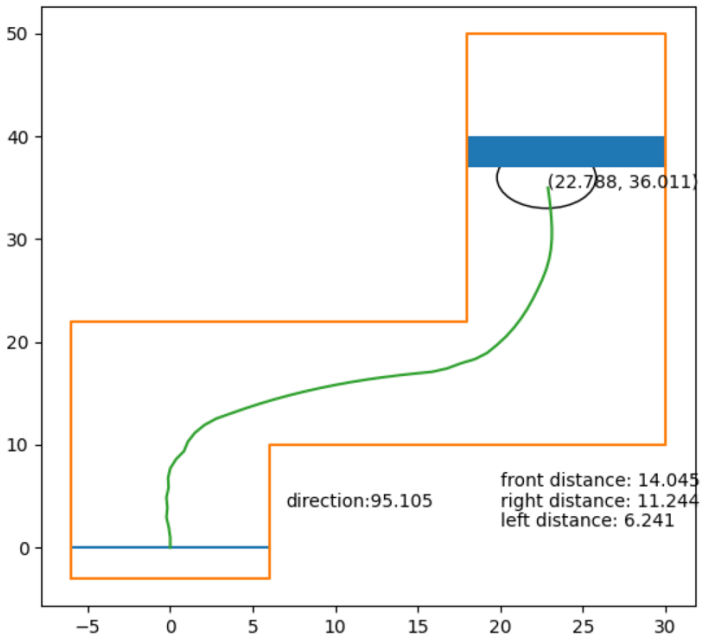


4. 分析

a. 當訓練資料集為 train4dAll.txt 時，若 epoch 只設成 1，自走車不會順利抵達終點。（碰到軌道就停止）若 epoch 設為 1000，自走車就可以順利抵達終點。

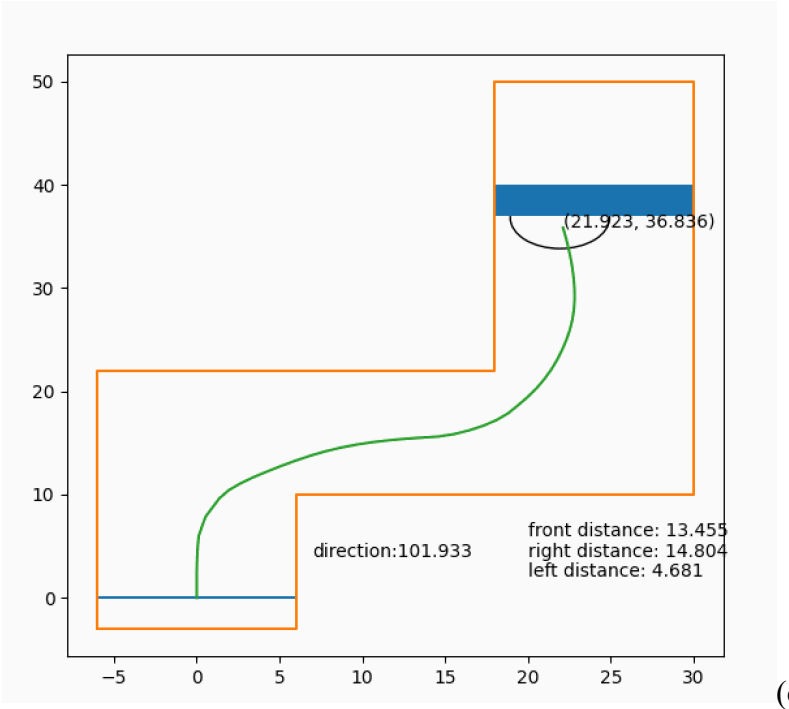


（epoch=1）



(epoch=1000)

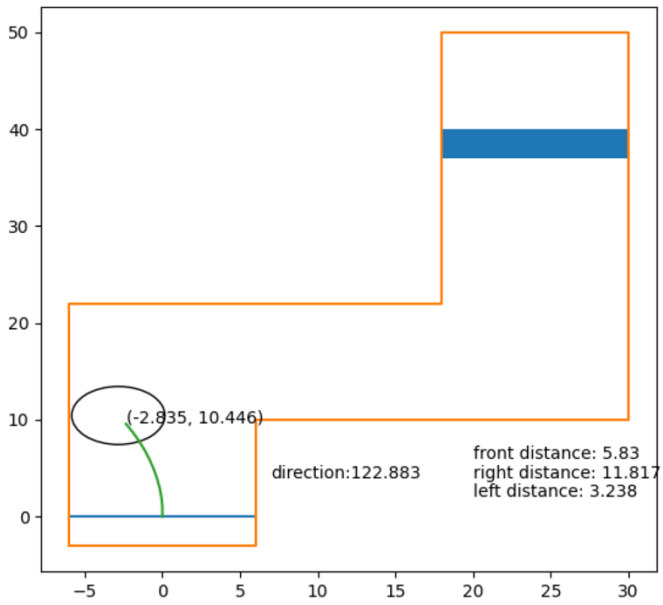
b. 當訓練資料集為 train6dAll.txt 時，若 epoch 只設成 1，自走車還是可以順利抵達終點。



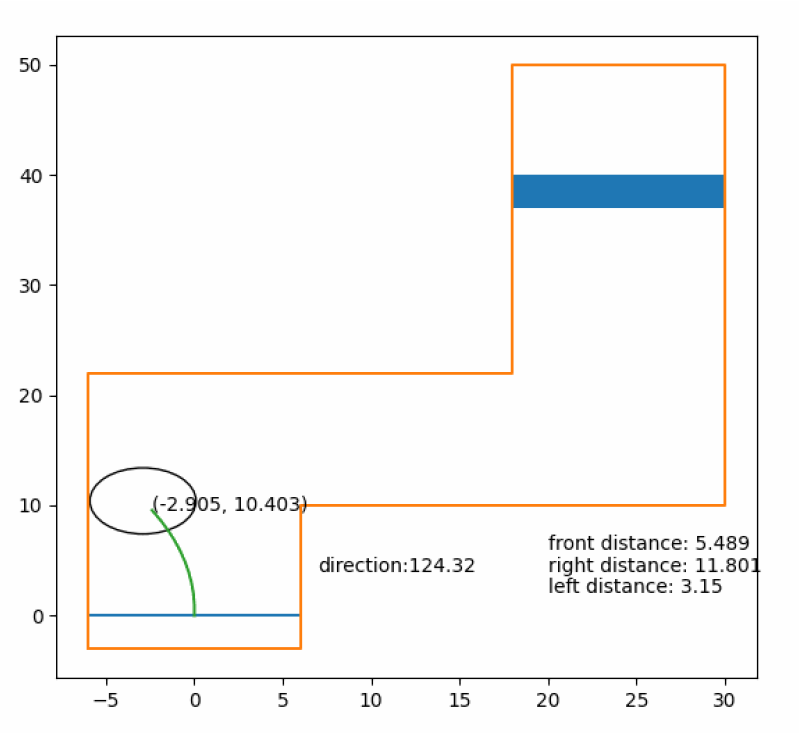
(epoch=1)

c. 由 4-(a)(b)可以推得，在學習率和 epoch 值相同的情況下，維度越高的訓練資料越有可能訓練成功。

d. 當訓練資料集為 train4dAll.txt，若將學習率設為 0.8，即使 epoch 達到 1000，還是沒辦法訓練成功。



e. 由 4-(b)可知，即使 epoch=1，當學習率為 0.01 時，train6dAll.txt 可以訓練出一個成功的模型。但是若將學習率調高為 0.8 時，train6dAll.txt 就無法訓練出一個成功的模型了。



f. 由 4-(a)(e)可知，訓練一個成功的模型，epoch 值的設定及學習率的選擇一樣重要。一開始測試的時候，我只將 epoch 設為 20，學習率設為 0.5，但是怎麼跑，自走車轉彎的角度都很小，所以我一開始一直懷疑我調整參數的方法有錯誤，但後來把 epoch 調到 1000，學習率設為 0.01 就成功了。