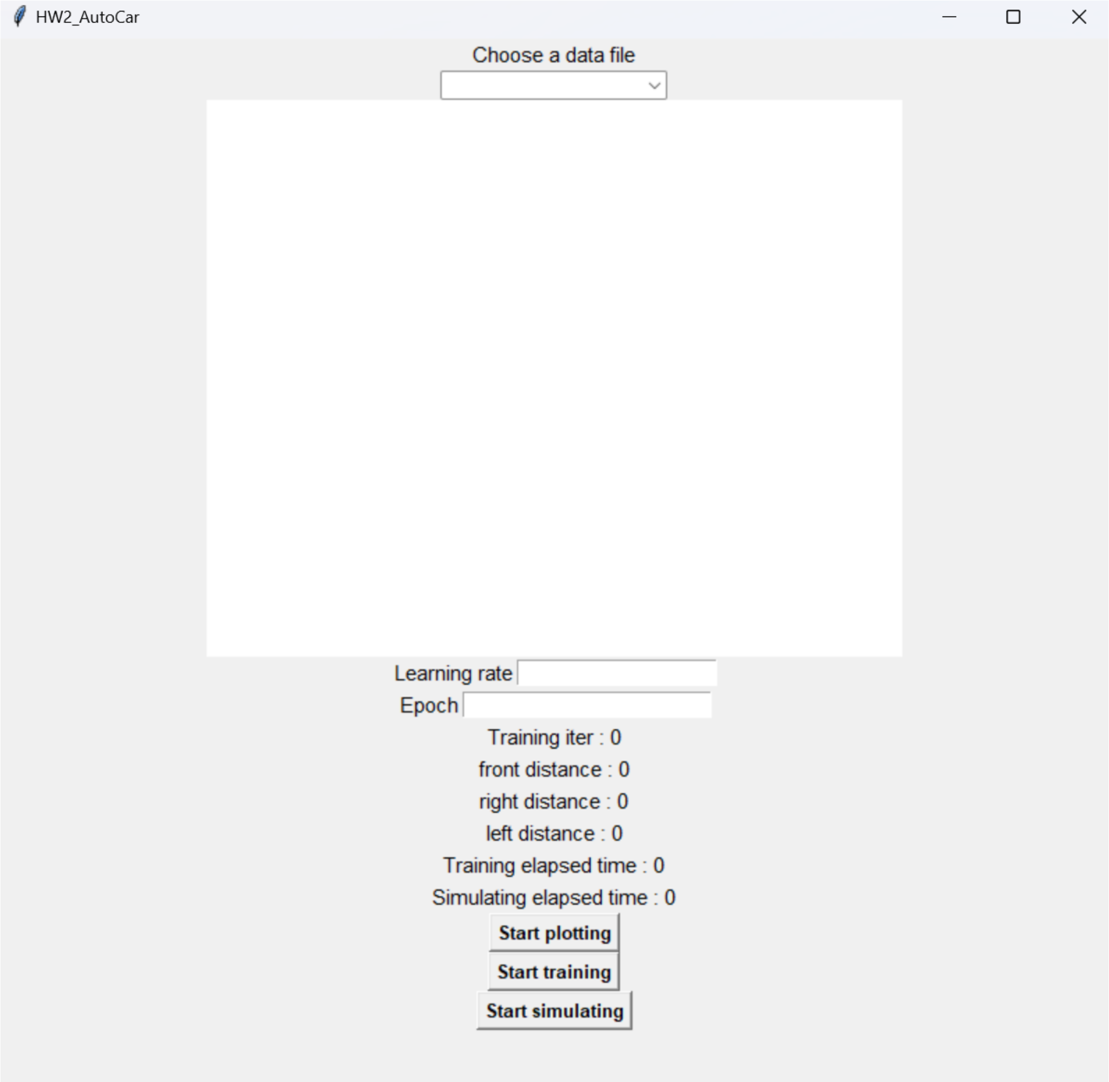
一、程式介面說明

這是尚未選取檔案前的 GUI 畫面



Training iter : 0
front distance : 0
right distance : 0
left distance : 0
Training elapsed time : 0
Simulating elapsed time : 0

# GUI 介面標籤和按鈕寫在 if __name__ == "__main__":

1. 透過下拉選單，選取 train4dAll.txt 或 train6dAll.txt

2. 輸入學習率和 epoch

3. label : 顯示偵測的前方距離、右方距離、左方距離; 顯示訓練、模擬的耗時

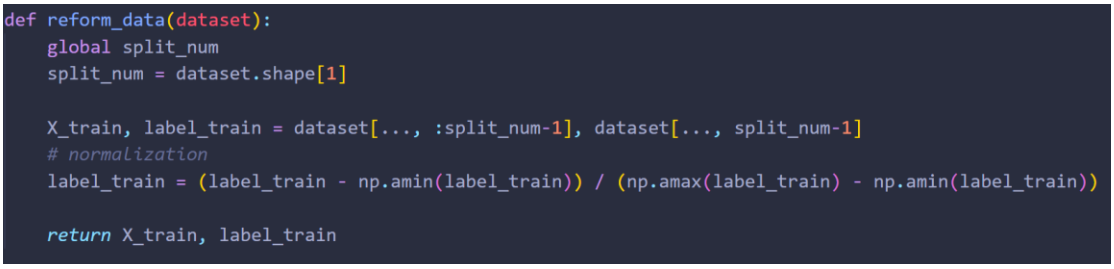
4. button : plot 是畫出軌道圖和起始、終點線; train 是將 Kmeans 獲得的資料集丟進 Training_Model 訓練; simulate 是去預測方向盤角度並繪製出軌跡圖

二、程式碼說明

用到的函式庫為 numpy, matplotlib, sympy, tkinter, os, time

※ 主要 Function 說明:

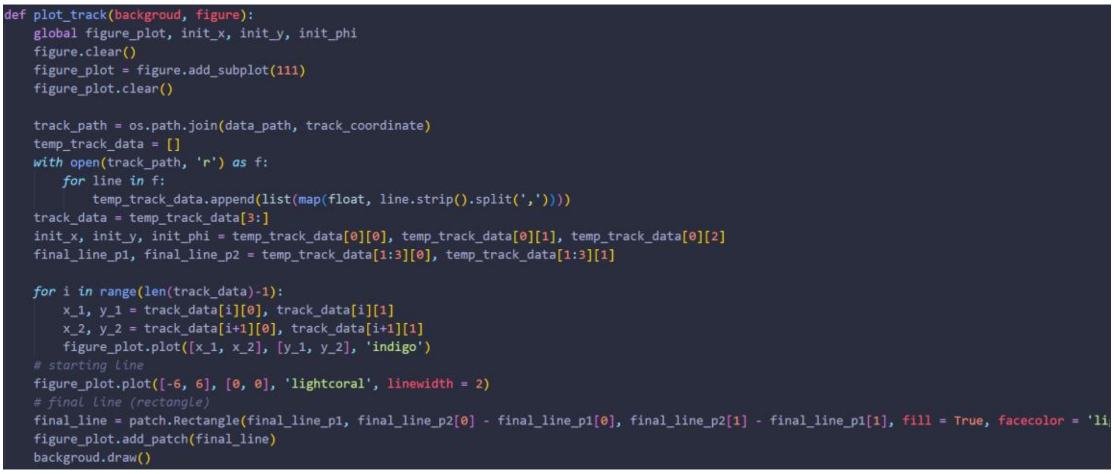
1. reform_data



```
def reform_data(dataset):
    global split_num
    split_num = dataset.shape[1]

    X_train, label_train = dataset[..., :split_num-1], dataset[..., split_num-1]
    # normalization
    label_train = (label_train - np.amin(label_train)) / (np.amax(label_train) - np.amin(label_train))

    return X_train, label_train
```

把 dataset 拆成兩部分，將 txt 檔裡的角度獨立出來為 label_train 並透過函數逼近正規化以便訓練

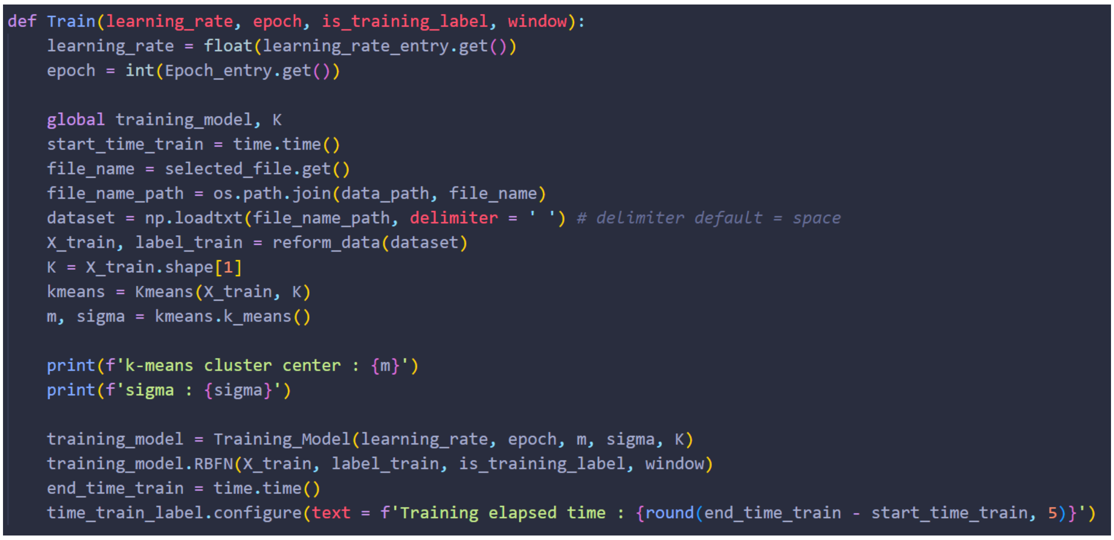
2. plot_track



```
def plot_track(backgroud, figure):
    global figure_plot, init_x, init_y, init_phi
    figure.clear()
    figure_plot = figure.add_subplot(111)
    figure_plot.clear()

    track_path = os.path.join(data_path, track_coordinate)
    temp_track_data = []
    with open(track_path, 'r') as f:
        for line in f:
            temp_track_data.append(list(map(float, line.strip().split(','))))
    track_data = temp_track_data[3:]
    init_x, init_y, init_phi = temp_track_data[0][0], temp_track_data[0][1], temp_track_data[0][2]
    final_line_p1, final_line_p2 = temp_track_data[1:3][0], temp_track_data[1:3][1]

    for i in range(len(track_data)-1):
        x_1, y_1 = track_data[i][0], track_data[i][1]
        x_2, y_2 = track_data[i+1][0], track_data[i+1][1]
        figure_plot.plot([x_1, x_2], [y_1, y_2], 'indigo')
    # starting line
    figure_plot.plot([-6, 6], [0, 0], 'lightcoral', linewidth = 2)
    # final line (rectangle)
    final_line = patch.Rectangle(final_line_p1, final_line_p2[0] - final_line_p1[0], final_line_p2[1] - final_line_p1[1], fill = True, facecolor = 'li
    figure_plot.add_patch(final_line)
    backgroud.draw()
```

read file 將「軌道座標點.txt」的起點、終點、軌道座標點讀出來，再用 for loop 把每一條線畫出來

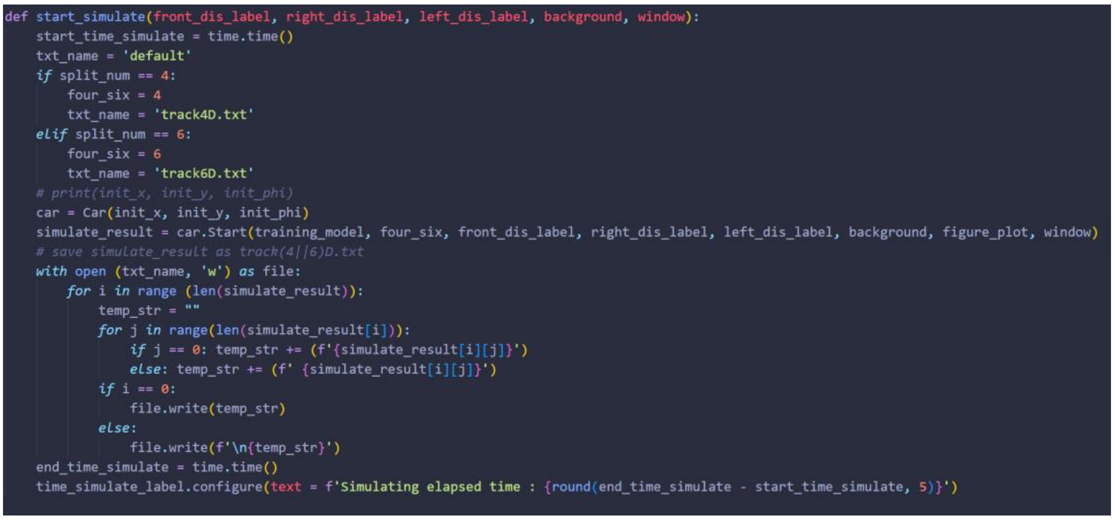
3. Train



```
def Train(learning_rate, epoch, is_training_label, window):
    learning_rate = float(learning_rate_entry.get())
    epoch = int(Epoch_entry.get())

    global training_model, K
    start_time_train = time.time()
    file_name = selected_file.get()
    file_name_path = os.path.join(data_path, file_name)
    dataset = np.loadtxt(file_name_path, delimiter = ' ') # delimiter default = space
    X_train, label_train = reform_data(dataset)
    K = X_train.shape[1]
    kmeans = Kmeans(X_train, K)
    m, sigma = kmeans.k_means()

    print(f'k-means cluster center : {m}')
    print(f'sigma : {sigma}')

    training_model = Training_Model(learning_rate, epoch, m, sigma, K)
    training_model.RBFN(X_train, label_train, is_training_label, window)
    end_time_train = time.time()
    time_train_label.configure(text = f'Training elapsed time : {round(end_time_train - start_time_train, 5)}')
```

將選到的檔案資料送入 Kmeans 做群聚分析，再把獲得的資料集丟入訓練模型做訓練(調整鍵結值, bias……)

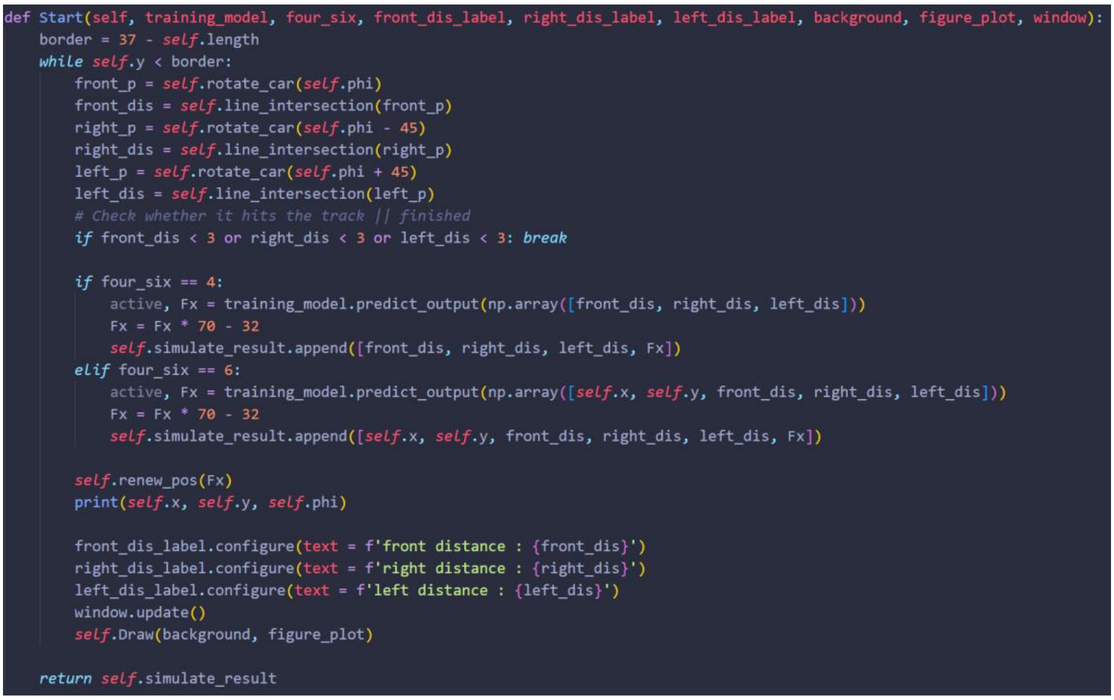
4. start_simulate



```
def start_simulate(front_dis_label, right_dis_label, left_dis_label, background, window):
    start_time_simulate = time.time()
    txt_name = 'default'
    if split_num == 4:
        four_six = 4
        txt_name = 'track4D.txt'
    elif split_num == 6:
        four_six = 6
        txt_name = 'track6D.txt'
    # print(init_x, init_y, init_phi)
    car = Car(init_x, init_y, init_phi)
    simulate_result = car.Start(training_model, four_six, front_dis_label, right_dis_label, left_dis_label, background, figure_plot, window)
    # save simulate_result as track(4||6)D.txt
    with open (txt_name, 'w') as file:
        for i in range (len(simulate_result)):
            temp_str = ""
            for j in range(len(simulate_result[i])):
                if j == 0: temp_str += (f'{simulate_result[i][j]}')
                else: temp_str += (f' {simulate_result[i][j]}')
            if i == 0:
                file.write(temp_str)
            else:
                file.write(f'\n{temp_str}')
    end_time_simulate = time.time()
    time_simulate_label.configure(text = f'Simulating elapsed time : {round(end_time_simulate - start_time_simulate, 5)}')
```

開始模擬 car 軌跡，並將模擬的資料存在 simulate_result file

5. Start



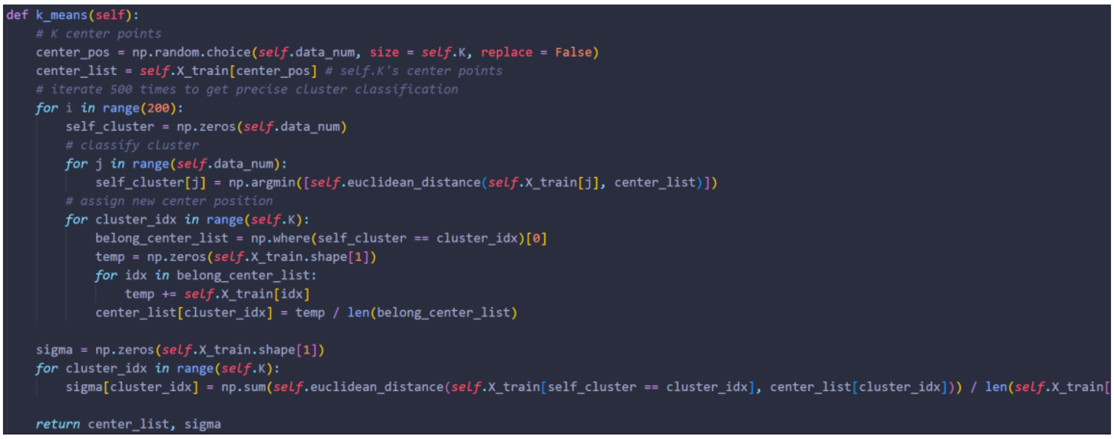
```python
def Start(self, training_model, four_six, front_dis_label, right_dis_label, left_dis_label, background, figure_plot, window):
    border = 37 - self.length
    while self.y < border:
        front_p = self.rotate_car(self.phi)
        front_dis = self.line_intersection(front_p)
        right_p = self.rotate_car(self.phi - 45)
        right_dis = self.line_intersection(right_p)
        left_p = self.rotate_car(self.phi + 45)
        left_dis = self.line_intersection(left_p)
        # Check whether it hits the track || finished
        if front_dis < 3 or right_dis < 3 or left_dis < 3: break

        if four_six == 4:
            active, Fx = training_model.predict_output(np.array([front_dis, right_dis, left_dis]))
            Fx = Fx * 70 - 32
            self.simulate_result.append([front_dis, right_dis, left_dis, Fx])
        elif four_six == 6:
            active, Fx = training_model.predict_output(np.array([self.x, self.y, front_dis, right_dis, left_dis]))
            Fx = Fx * 70 - 32
            self.simulate_result.append([self.x, self.y, front_dis, right_dis, left_dis, Fx])

        self.renew_pos(Fx)
        print(self.x, self.y, self.phi)

        front_dis_label.configure(text = f'front distance : {front_dis}')
        right_dis_label.configure(text = f'right distance : {right_dis}')
        left_dis_label.configure(text = f'left distance : {left_dis}')
        window.update()
        self.Draw(background, figure_plot)

    return self.simulate_result
```

為 class Car 的開始模擬 function，將偵測的前右左距離、預測方向盤角度存

下來，接著透過自走車的運動方程式更新下一個座標位置，繼續模擬

最後會回傳蒐集到的軌跡資料給 start_simulate

6. k_means (由 Train 做 function call)

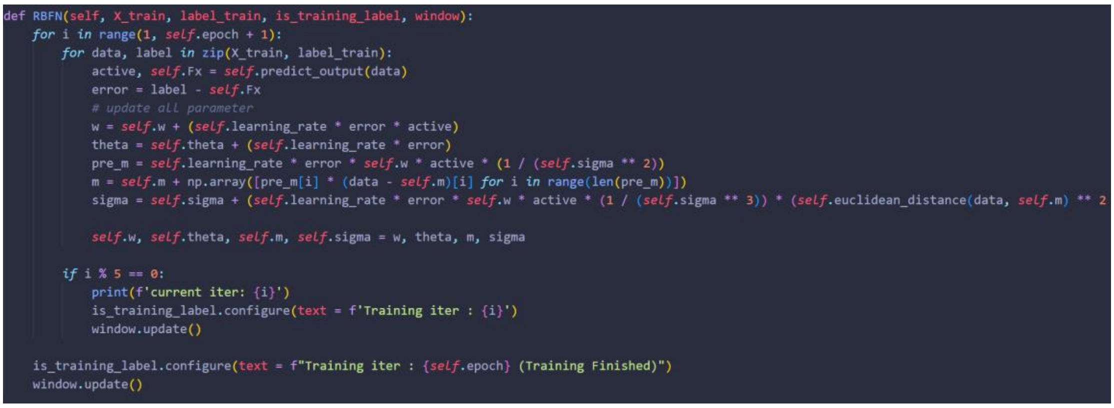


```python
def k_means(self):
    # K center points
    center_pos = np.random.choice(self.data_num, size = self.K, replace = False)
    center_list = self.X_train[center_pos] # self.K's center points
    # iterate 500 times to get precise cluster classification
    for i in range(200):
        self_cluster = np.zeros(self.data_num)
        # classify cluster
        for j in range(self.data_num):
            self_cluster[j] = np.argmin([self.euclidean_distance(self.X_train[j], center_list)])
        # assign new center position
        for cluster_idx in range(self.K):
            belong_center_list = np.where(self_cluster == cluster_idx)[0]
            temp = np.zeros(self.X_train.shape[1])
            for idx in belong_center_list:
                temp += self.X_train[idx]
            center_list[cluster_idx] = temp / len(belong_center_list)

    sigma = np.zeros(self.X_train.shape[1])
    for cluster_idx in range(self.K):
        sigma[cluster_idx] = np.sum(self.euclidean_distance(self.X_train[self_cluster == cluster_idx], center_list[cluster_idx])) / len(self.X_train[

    return center_list, sigma
```

為 class Kmeans 的主要 function，進行 k-means 資料分群

迭代完成後，會回傳獲得的最終中心點資料集及 sigma 資料集

7. RBFN (由 Train 做 function call)



```
def RBFN(self, X_train, label_train, is_training_label, window):
    for i in range(1, self.epoch + 1):
        for data, label in zip(X_train, label_train):
            active, self.Fx = self.predict_output(data)
            error = label - self.Fx
            # update all parameter
            w = self.w + (self.learning_rate * error * active)
            theta = self.theta + (self.learning_rate * error)
            pre_m = self.learning_rate * error * self.w * active * (1 / (self.sigma ** 2))
            m = self.m + np.array([pre_m[i] * (data - self.m)[i] for i in range(len(pre_m))])
            sigma = self.sigma + (self.learning_rate * error * self.w * active * (1 / (self.sigma ** 3)) * (self.euclidean_distance(data, self.m) ** 2

            self.w, self.theta, self.m, self.sigma = w, theta, m, sigma

        if i % 5 == 0:
            print(f'current iter: {i}')
            is_training_label.configure(text = f'Training iter : {i}')
            window.update()

    is_training_label.configure(text = f"Training iter : {self.epoch} (Training Finished)")
    window.update()
```

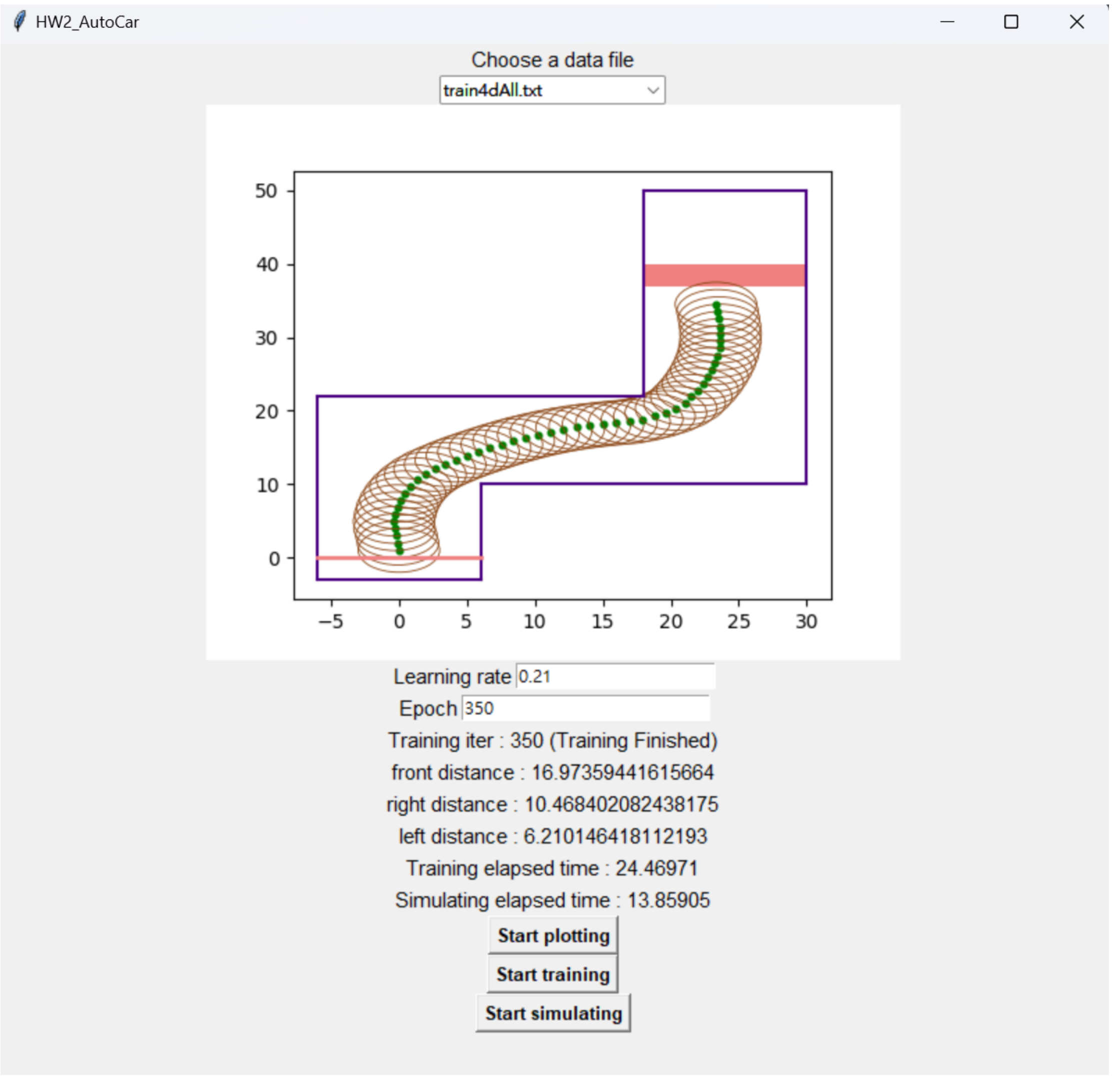
為 class Training_Model 的主要 function，根據講義第三章 RBFN 的推導公式去不斷調整所有參數 (w, theta, m, sigma) 直到 current iter 達到 epoch 上限

三、實驗結果

***成功走到終點**

train4dAll.txt

Learning rate: 0.21,　Epoch: 350



Learning rate 0.21

Epoch 350

Training iter : 350 (Training Finished)

front distance : 16.97359441615664

right distance : 10.468402082438175

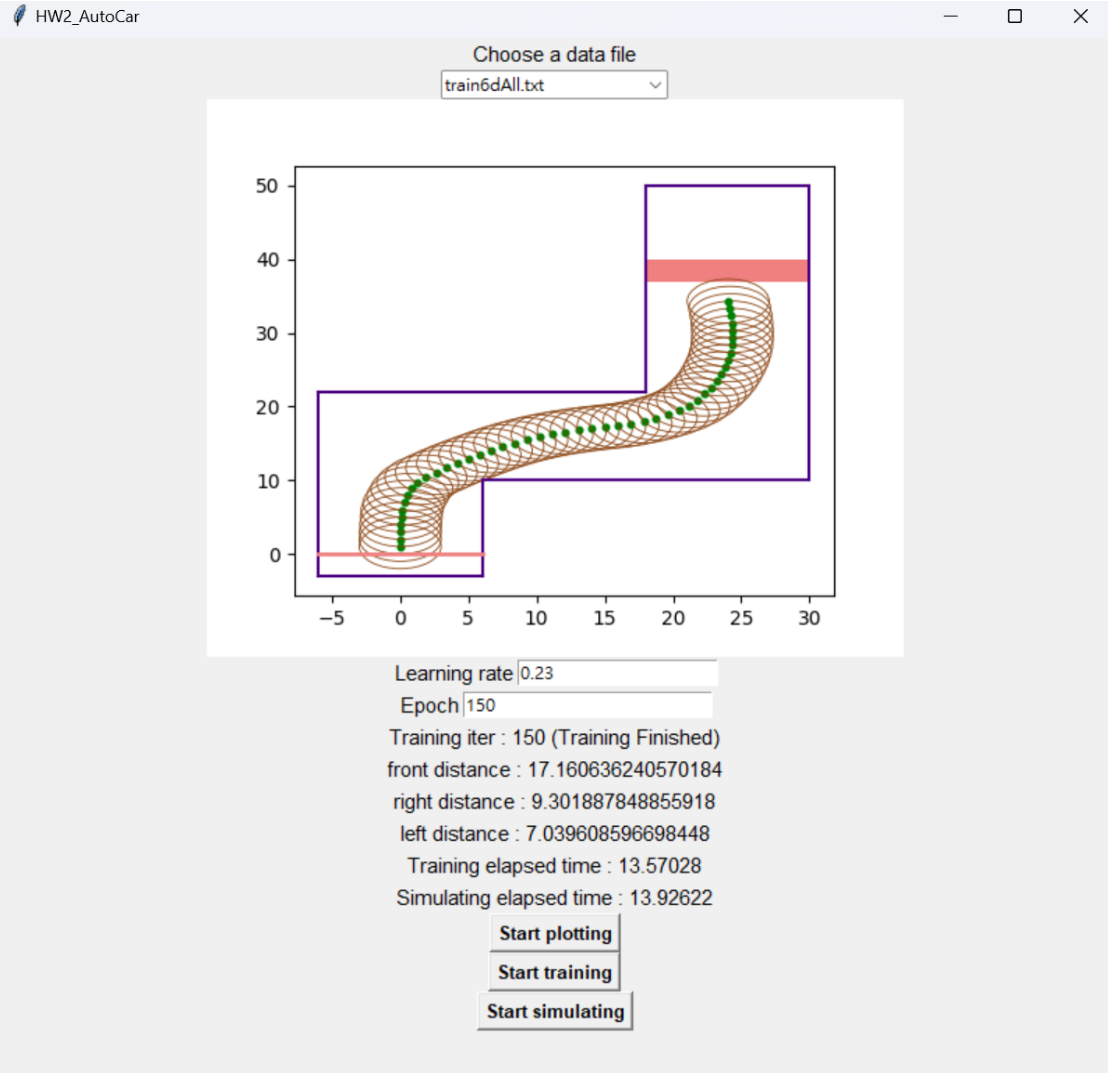
left distance : 6.210146418112193

Training elapsed time : 24.46971

Simulating elapsed time : 13.85905

train6dAll.txt

Learning rate: 0.23,   Epoch: 150



train6dAll.txt

Learning rate 0.23

Epoch 150

Training iter : 150 (Training Finished)

front distance : 17.160636240570184

right distance : 9.301887848855918

left distance : 7.039608596698448

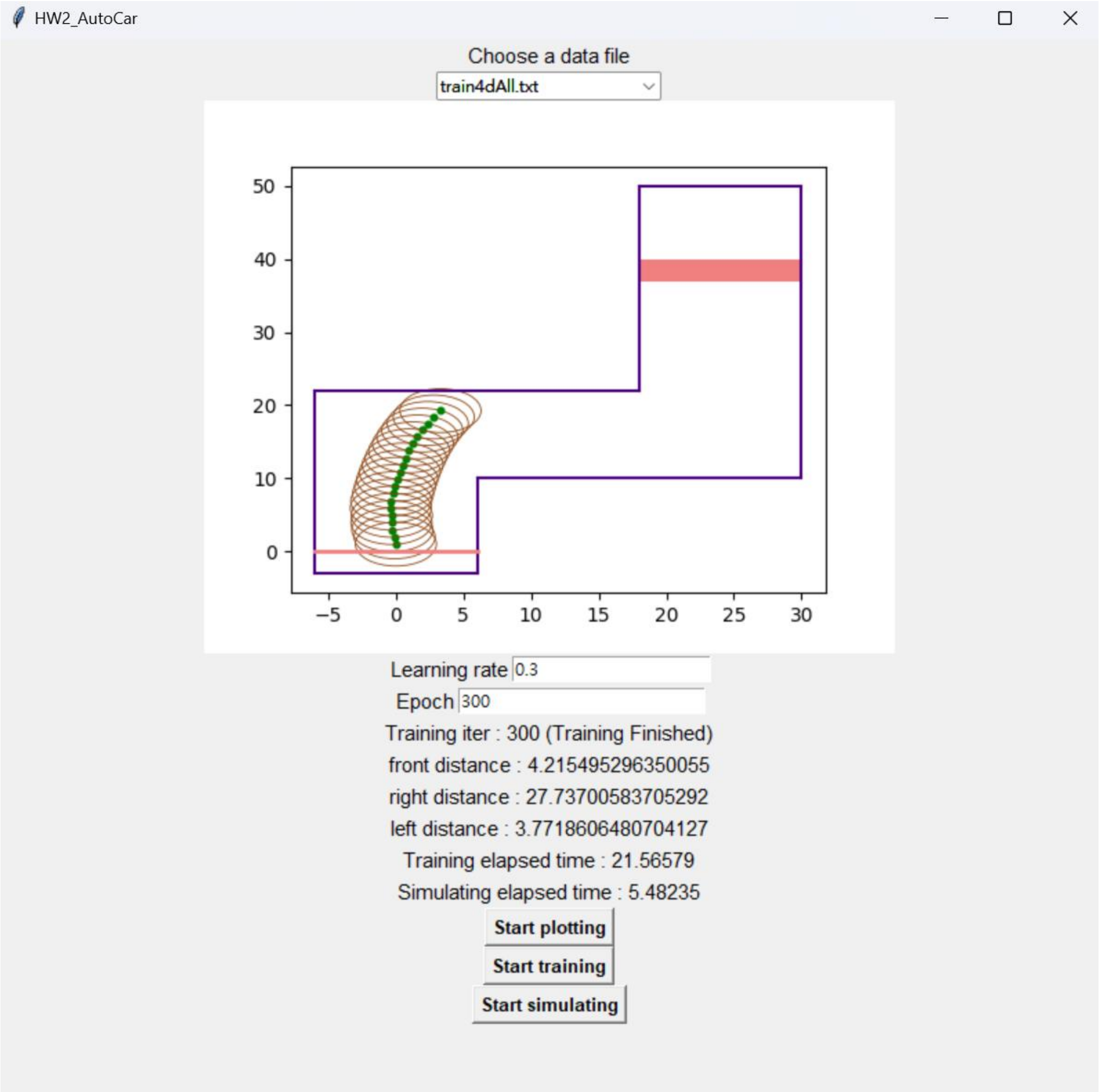
Training elapsed time : 13.57028

Simulating elapsed time : 13.92622

***中途碰壁**

train4dAll.txt

Learning rate: 0.3,　Epoch: 300



Learning rate 0.3

Epoch 300

Training iter : 300 (Training Finished)

front distance : 4.215495296350055

right distance : 27.73700583705292

left distance : 3.7718606480704127

Training elapsed time : 21.56579

Simulating elapsed time : 5.48235

四、分析

4d 的資料在「學習率: 0.1 ~ 0.25、epoch: 200 ~ 300」的範圍內軌跡最漂亮

epoch 超過 0.25 後軌跡會開始偏掉，到 0.3 以上通常就都會撞牆了

而實測訓練時間約落在 epoch: 1000 耗時 75 秒，所以通常 200~300 epoch

都要先等個 25 秒上下去訓練，接著才能去模擬軌跡

6d 的資料大概在 50~100 epoch 軌跡就很漂亮了，epoch 不必設定過高

至於學習率就算設定 0.5，軌跡依舊沒碰壁，猜測是因為資料維度較大

(多了 x, y 座標) 所以不必太多次 iteration，訓練模型就可以很精確了

五、加分

模擬程式 new_simulate.py