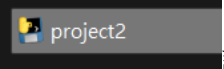


A. 操作流程:

1. 首先執行  請確保目錄底下包含等等讀檔會用到的 train4dAll.txt/train6dAll.txt/軌道座標點.txt



2. 選擇想要的檔案維度

```
C:\Users\User\Desktop\project2>python project2.py
please enter 4 or 6: 4
```

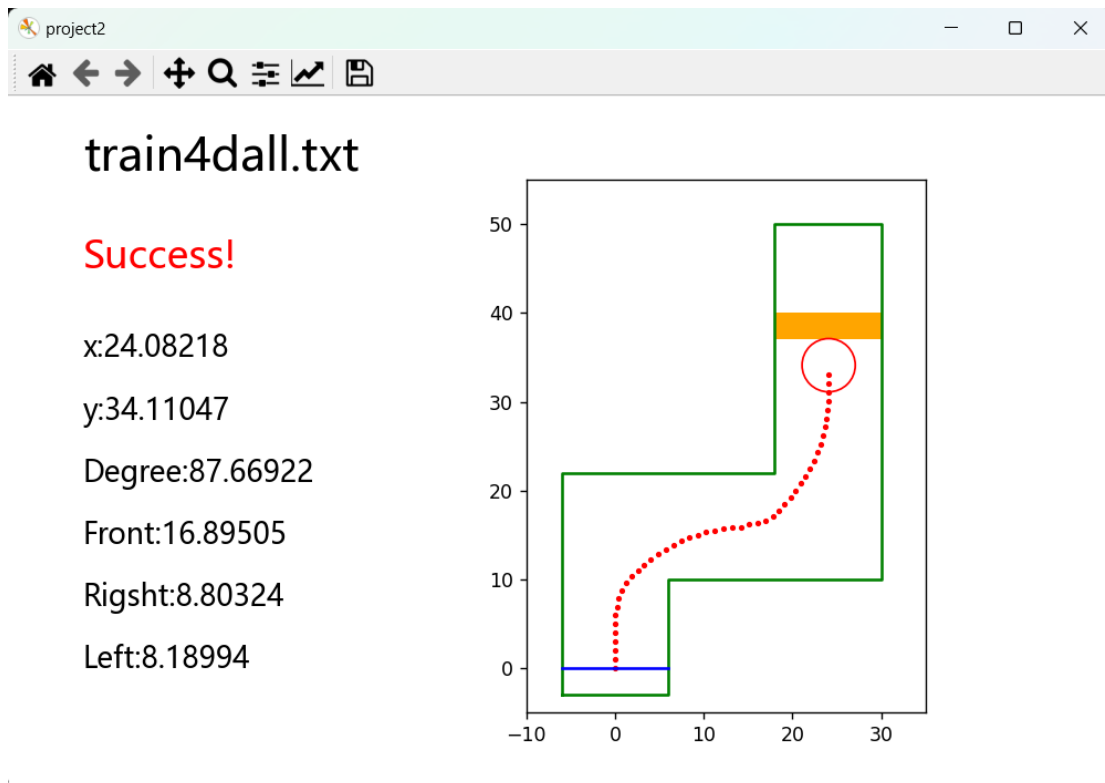
3. 選擇想要的分群數量(K)不一定等於類神經元數目，晚點會解釋，若檔案為 4D 建議 K=15，若為 6D 建議 K=10，晚點於分析中會比較

```
*****第一階段非監督式學習(k-means)*****
請輸入分群數量(K): 15
```

4. 終端機會輸出每個訓練回合的參數，以及類神經元的訓練結果，以供比對

Epoch	η	loss	bias	*****類神經元的訓練結果*****	
1	0.1	81.6844	-4.2615	Neuron	Weight
2	0.098	78.5983	-5.3767	1	-18.18955357605208
3	0.096	78.8444	-5.4584	2	2.2946600681251543
4	0.094	78.9546	-5.327	3	16.69278784276762
5	0.092	79.1084	-4.7126	4	-19.99663809691484
6	0.09	79.1032	-4.0563	5	15.82461408184765
7	0.088	79.0727	-3.5108	6	-19.298599069849878
8	0.086	78.9885	-3.0174	7	18.042206841014085
9	0.084	78.8614	-2.5437	8	2.187336273984277
10	0.082	78.7048	-2.0773	9	9.911822394138165
11	0.08	78.5265	-1.6138	10	1.5852336925187933
12	0.078	78.3319	-1.1515	11	0.19482031789537524
13	0.076	78.1246	-0.6896	12	6.055778286725495
14	0.074	77.9075	-0.2285	13	-4.18533029719894
15	0.072	77.6823	0.2299	14	14.414877395956166
16	0.07	77.4502	0.6827	15	12.402281048699328
17	0.068	77.212	1.1261	*****啟動自駕車(RNFN)*****	
18	0.066	76.9686	1.5562		
19	0.064	76.7206	1.9693		
20	0.062	76.4687	2.3625		
21	0.06	76.2138	2.7336		

5. UI 介面顯示: 車體移動動畫、訓練檔案、訓練結果以及六項結果



6. 關掉 UI 介面後，輸入 Y/y 即可重新訓練

```
Press Y/y to retry: Y  
please enter 4 or 6: |
```

7. 程式會將結果寫入自動創建的 track4D 或 track6D

project2	2023/11/15 下午 06:55	PY 檔案	12 KB
track4D	2023/11/15 下午 06:56	文字文件	4 KB
track6D	2023/11/15 下午 06:56	文字文件	6 KB

B. 程式碼說明

1. main function 決定 file 維度 並呼叫 RBFN

```
if __name__ == '__main__':  
    RBFN(int(input('please enter 4 or 6: ')))  
    while (True):  
        r = input('Press Y/y to retry: ')  
        if r == 'Y' or r == 'y':  
            RBFN(int(input('please enter 4 or 6: ')))  
        else:  
            break
```

(以下都在 RBFN 類別中)

2. 呼叫 RBFN 後讀取檔案到 self.data 裡，順便依照數據點數目初始化待會 kmeans 會用到的 kmeans_dist 和 kmeans_who

```
class RBFN:
    def __init__(self, file_dim: int):
        self.d = file_dim
        path = f'./train{self.d}dAll.txt'
        with open(path) as file:
            line = file.readline().split()
            self.kmeans_dist = np.array([math.inf])
            self.kmeans_who = np.array([-1])
            self.data = np.array([[float(i) for i in line] + [math.inf, -1]])
            for line in file:
                line = line.split()
                self.kmeans_dist = np.append(self.kmeans_dist, [math.inf], axis=0)
                self.kmeans_who = np.append(self.kmeans_who, [-1], axis=0)
                self.data = np.append(self.data, [[float(i) for i in line] + [math.inf, -1]], axis=0)
```

3. 讀取軌道座標點.txt

```
with open('軌道座標點.txt') as file:
    line = file.readline().split()
    line_arr = line[0].split(',')
    car_x = int(line_arr[0])
    car_y = int(line_arr[1])
    deg = math.radians(int(line_arr[2]))

    line = file.readline().split()
    line_arr = line[0].split(',')
    self.final_line_left_x = int(line_arr[0])
    self.final_line_left_y = int(line_arr[1])
    line = file.readline().split()
    line_arr = line[0].split(',')
    self.final_line_right_x = int(line_arr[0])
    self.final_line_right_y = int(line_arr[1])
    self.final_line_width = self.final_line_right_x - self.final_line_left_x
    self.final_line_height = self.final_line_right_y - self.final_line_left_y
    self.road_x = []
    self.road_y = []
    for line in file:
        line_arr = line.split(',')
        car_x = int(line_arr[0])
        car_y = int(line_arr[1])
        deg = math.radians(int(line_arr[2]))
```

4. Kmeans 分群並開始訓練

```
self.d -= 1
self.neuron_group = []
print('*****第一階段非監督式學習(k-means)*****')
K = int(input('請輸入分群數量(K): '))
self.kmeans(K)

epoch, bias = 50, -1
print('*****開始訓練*****')
print("Epoch\tη\tloss\tbias")
for e in range(epoch):
    loss = 0
    η = 0.1 * np.exp(-0.1 * e) #指數
    η = (epoch - e) / epoch * 0.1 #線性
    for data in self.data:
        predict = bias
        for neuron in self.neuron_group:
            neuron.φ(data[:self.d])
            predict += neuron.w * neuron.y
        for neuron in self.neuron_group:
            neuron.update(η, data[self.d] - predict, data[:self.d])
        bias += η * (data[self.d] - predict)
        loss += (data[self.d] - predict) ** 2 / 2
    loss /= len(self.data)
    print(f'{e + 1}\t{round(η, 3)}\t{round(loss, 4)}\t{round(bias, 4)}')
print()
print(f'*****類神經元的訓練結果*****')
print('Neuron \t Weight \t\t Mean \t\t\t\t\t standard')
```

epoch 預設 50、學習率依照線性遞減、其餘參數參照以下公式

$$F(\bar{x}_n) = \sum_{j=0}^p w_j(n) \varphi_j(\bar{x}_n), E(n) = \frac{1}{2} (y(n) - F(\bar{x}_n))^2, \varphi_j(\bar{x}_n) = e^{\frac{-\|\bar{x}_n - \bar{m}_j(n)\|^2}{2\sigma_j^2}}$$

- i. $w_j(n+1) = w_j(n) - \eta \frac{\partial E(n)}{\partial w_j(n)} = w_j(n) - \eta \frac{\partial E(n)}{\partial F(\bar{x}_n)} \frac{\partial F(\bar{x}_n)}{\partial w_j(n)} = w_j(n) + \eta (y(n) - F(\bar{x}_n)) \varphi_j(\bar{x}_n)$
- ii. $m_j(n+1) = m_j(n) - \eta \frac{\partial E(n)}{\partial m_j(n)} = m_j(n) - \eta \frac{\partial E(n)}{\partial F(\bar{x}_n)} \frac{\partial F(\bar{x}_n)}{\partial \varphi_j(\bar{x}_n)} \frac{\partial \varphi_j(\bar{x}_n)}{\partial m_j(n)} = m_j(n) + \eta (y(n) - F(\bar{x}_n)) w_j(n) \varphi_j(\bar{x}_n) \frac{\bar{x}_n - \bar{m}_j(n)}{\sigma_j^2}$
- iii. $\sigma_j(n+1) = \sigma_j(n) - \eta \frac{\partial E(n)}{\partial \sigma_j(n)} = \sigma_j(n) - \eta \frac{\partial E(n)}{\partial F(\bar{x}_n)} \frac{\partial F(\bar{x}_n)}{\partial \varphi_j(\bar{x}_n)} \frac{\partial \varphi_j(\bar{x}_n)}{\partial \sigma_j(n)} = \sigma_j(n) + \eta (y(n) - F(\bar{x}_n)) w_j(n) \varphi_j(\bar{x}_n) \frac{\|\bar{x}_n - \bar{m}_j(n)\|^2}{\sigma_j^3}$

5. 依照公式設定神經元的初始化、基底函數、更新

```
class Neuron:
    def __init__(self, m: np.ndarray, sigma: float):
        self.w = random.random()
        self.m = m
        self.sigma = sigma
        self.y = 0

    def phi(self, x: np.ndarray) -> None:
        self.y = math.exp(-((x - self.m) ** 2).sum() / (2 * self.sigma ** 2))

    def update(self, eta: float, error: float, x: np.ndarray) -> None:
        curr_w = self.w
        curr_m = self.m.copy()
        curr_sigma = self.sigma
        same = eta * error * self.y
        self.w += same
        self.m += same * curr_w * (x - curr_m) / (curr_sigma ** 2)
        self.sigma += same * curr_w * ((x - curr_m) ** 2).sum() / (curr_sigma ** 3)
```

6. 讓車子依照訓練好的神經元去預判角度，車子如果未碰壁並且 car_y 超過終點線，即停止預測

```

print('*****啟動自駕車(RNFN)*****')
self.steps_group = []
All_result = []
self.pas = True
self.steps_group.append([[car_x,car_y], deg, get_x_y_dis(car_x,car_y, deg)])
while car_y < self.final_line_right_y-3:
    all_x_y_dis = get_x_y_dis(car_x,car_y, deg)
    predict = bias
    for neuron in self.neuron_group:
        if self.d==3:
            neuron.φ(all_x_y_dis[:, 2])
        elif self.d==5:
            neuron.φ(np.append([car_x,car_y], all_x_y_dis[:, 2]))
        predict += neuron.w * neuron.y
    predict *= -1
    if self.d==3:
        result=f'{all_x_y_dis[:, 2][0]} {all_x_y_dis[:, 2][1]} {all_x_y_dis[:, 2][2]} '
    elif self.d==5:
        result =f'{car_x} {car_y} {all_x_y_dis[:, 2][0]} {all_x_y_dis[:, 2][1]} {all_x_y_dis[:, 2][2]} '
    All_result.append(result)
    predict = math.radians(predict)

```

All_result 用於紀錄所有結果，self.steps_group 用於紀錄每一步車子的位置以及三個方向的距離以供後續畫圖使用，Self.pas 紀錄成功與否、all_x_y_dis 取得三個方向的 x,y,distance，再來就是依照 RBFN 原理先套用基底函數再乘上權重並加總取得預測值。

7. 判斷有無撞車並更改車體位置

```

if car_x < -3 or car_x > 27 or car_y < 0:
    self.pas = False
    break
elif 0<car_y<10 and (car_x>3 or car_x<-3 or math.dist((car_x,car_y), (0,0))>10):
    self.pas = False
    break
elif 10<car_y<22 and (car_x>27 or car_x<-3 or math.dist((car_x,car_y), (27,22))>10):
    self.pas = False
    break
elif 22<car_y<50 and (car_x>27 or car_x<15 or math.dist((car_x,car_y), (27,50))>10):
    self.pas = False
    break
elif -6<car_x<6 and (car_y>19 or car_y<0):
    self.pas = False
    break
elif 6<car_x<18 and (car_y>19 or car_y<7):
    self.pas = False
    break
elif 18<car_x<30 and car_y<7:
    self.pas = False
    break
car_x += math.cos(deg + predict) + math.sin(predict) * math.sin(deg) #x方向位移
car_y += math.sin(deg + predict) - math.cos(predict) * math.cos(deg) #y方向位移
deg = (deg - math.asin(2 * math.sin(predict) / 3)) % (math.pi * 2) #決定新角度
self.steps_group.append([[car_x,car_y], deg, all_x_y_dis])

```

8. KMEANS 處使化 K 個中心點依照公式去分群，終止條件設為和上次結果相同

```
def kmeans(self, K) -> None:
    K_center = self.data[np.random.choice(self.data.shape[0], K, replace=False), :self.d]
    Each_sum_in_K = [np.zeros(self.d + 1) for i in range(K)]
    same = False
    while not same:
        for i in range(K):
            if o_sum[i] != 0 and o_num[i] != 0:
                self.neuron_group.append(Neuron(K_center[i], o_sum[i] / o_num[i]))
```

分群完後初始化各個神經元

9. 依照 x 方向和 y 方向的牆體去計算車體再三個方向的距離，並存入最小的， $k \geq 0$ 確保取得的是車體的前方方向，l, h 確保車體沒超過上下線。

```
def get_x_y_dis( car_x: int, car_y: int, deg: float) -> np.ndarray:
    x_wall = [(-6, -3, 22), (6, -3, 10), (18, 22, 50), (30, 10, 50)]
    y_wall = [(-3, -6, 6), (10, 6, 30), (22, -6, 18), (50, 18, 30)]
    min_x_y_dis = []
    for i in range(3):
        min_x_y_dis.append((0,0,math.inf))
    left_front_right=[-1,0,1]
    for i in left_front_right:
        for x, l, h in x_wall:
            k = (x - car_x) / math.cos(deg + i * math.pi / 4)
            y = car_y + k * math.sin(deg + i * math.pi / 4)
            dis = math.dist((car_x, car_y), (x, y))
            if k >= 0 and l <= y <= h and dis < min_x_y_dis[i+1][2]:
                min_x_y_dis[i+1] = (x, y, dis)
        for y, l, h in y_wall:
            k = (y - car_y) / math.sin(deg + i * math.pi / 4)
            x = car_x + k * math.cos(deg + i * math.pi / 4)
            dis = math.dist((car_x, car_y), (x, y))
            if k >= 0 and l <= x <= h and dis < min_x_y_dis[i+1][2]:
                min_x_y_dis[i+1] = (x, y, dis)
    return np.array([min_x_y_dis[1], min_x_y_dis[2], min_x_y_dis[0]])
```

10. 紀錄檔案並畫出車體行徑動畫設定 subplot 要橫跨(1,3)並從(01)開始

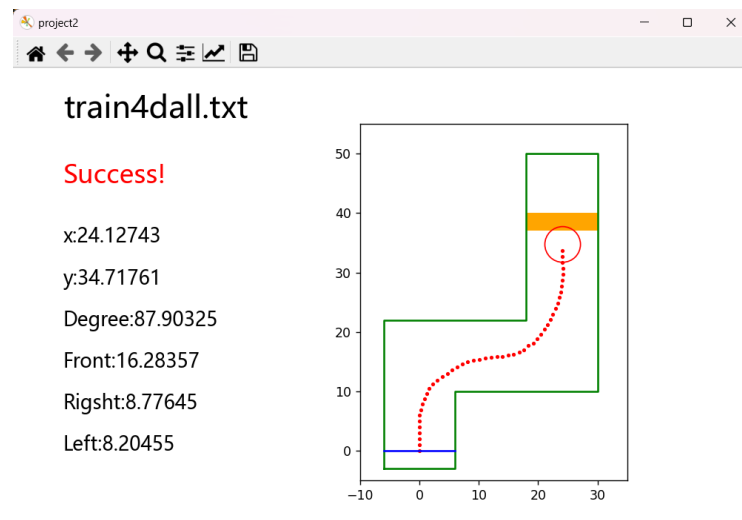
```
self.steps_group.append([car_x, car_y], deg, min_x_y_dis)
with open(f'track{self.d+1}0.txt', 'w') as file:
    file.writelines(All_result)
fig = plt.figure(figsize=(8,5))
plt.get_current_fig_manager().set_window_title('project2')
self.sub_plot = plt.subplot2grid((1, 3), (0, 1), colspan=2)
self.sub_plot.set_aspect('equal', 'box')
anim=animation.FuncAnimation(fig, self.draw, frames=len(self.steps_group),repeat=False)
plt.show()
```

11. 畫出 UI 介面中的 subplot

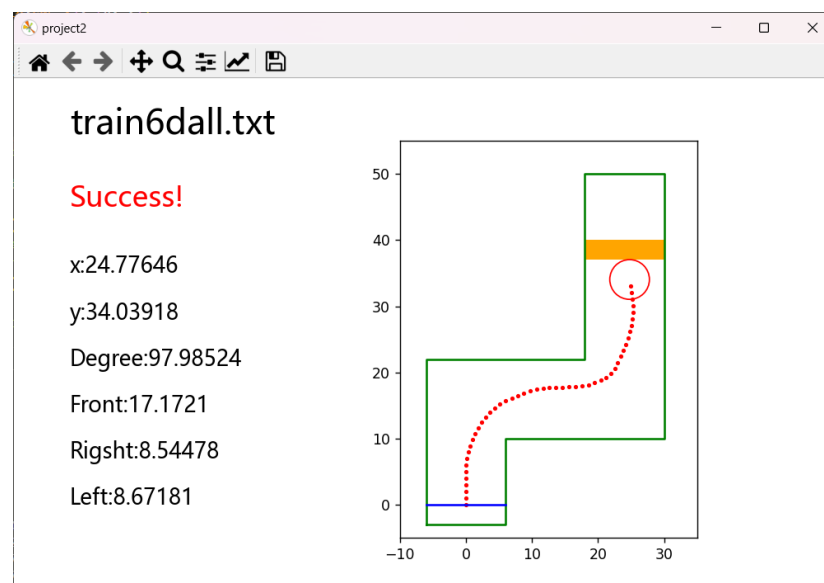
```
def draw(self, step):
    self.sub_plot.clear()
    self.sub_plot.set_xlim(-10, 35)
    self.sub_plot.set_ylim(-5, 55)
    self.sub_plot.plot(self.road_x, self.road_y, color='green')
    self.sub_plot.plot([-6, 6], [0, 0], color='blue')
    self.sub_plot.add_patch(Rectangle((self.final_line_left_x, self.final_line_left_y),
                                      self.final_line_width, self.final_line_height, facecolor='orange', fill=True))
    for s in self.steps_group[:step]:
        self.sub_plot.scatter(s[0][0], s[0][1], color='red', marker='o', s=4)
    self.sub_plot.add_patch(Circle((self.steps_group[step][0][0], self.steps_group[step][0][1]), 3, color='red', fill=False))
    if self.pas==True:
        self.sub_plot.text(-60, 45, f'Success!', family='Microsoft YaHei', size=20, color='red')
    else:
        self.sub_plot.text(-60, 45, f'Please try again!', family='Microsoft YaHei', size=20, color='red')
    self.sub_plot.text(-60, 56, f'train{self.d+1}dall.txt', family='Microsoft YaHei', size=24)
    self.sub_plot.text(-60, 35, f'x:{round(self.steps_group[step][0][0], 5)}', family='Microsoft YaHei', size=15)
    self.sub_plot.text(-60, 28, f'y:{round(self.steps_group[step][0][1], 5)}', family='Microsoft YaHei', size=15)
    self.sub_plot.text(-60, 21, f'Degree:{round(math.degrees(self.steps_group[step][1]), 5)}', family='Microsoft YaHei', size=15)
    self.sub_plot.text(-60, 14, f'Front:{round(self.steps_group[step][2][0][2], 5)}', family='Microsoft YaHei', size=15)
    self.sub_plot.text(-60, 7, f'Rightsht:{round(self.steps_group[step][2][1][2], 5)}', family='Microsoft YaHei', size=15)
    self.sub_plot.text(-60, 0, f'Left:{round(self.steps_group[step][2][2][2], 5)}', family='Microsoft YaHei', size=15)
```

C. 實驗結果

1. Train4dall.txt 成功的結果:



2. Train6dall.txt 成功的結果



3. 4D、K=10 or 15

```
4D.txt K: 10 Dsuceess rate: 18/20
***** 階段非監督式學習 (K means) *****
第 一 階段非監督式學習 (K means)
4D.txt K: 15 Dsuceess rate: 17/20
***** 階段非監督式學習 (K means) *****
```

4. 6D、K=10 or 15

```
6D.txt K: 10 Dsuceess rate: 13/20
***** 階段非監督式學習 (K means) *****
第 一 階段非監督式學習 (K means)
6D.txt K: 15 Dsuceess rate: 7/20
***** 階段非監督式學習 (K means) *****
```

D. 分析

1. 學習率的調整方法

```
 $\eta = 0.1 * \text{np.exp}(-0.1 * e)$  #指數
 $\eta = (\text{epoch} - e) / \text{epoch} * 0.1$  #線性
```

經過實驗後線性的遞減很明顯表現的較好

可能原因：線性遞減的特性是相對簡單易於實現，也能提供整個過程較穩定的收斂，如果我提供的問題相對簡單，線性遞減就足夠適應，指數遞減的特性是前期快速收斂而有後續提供更細微的調整。

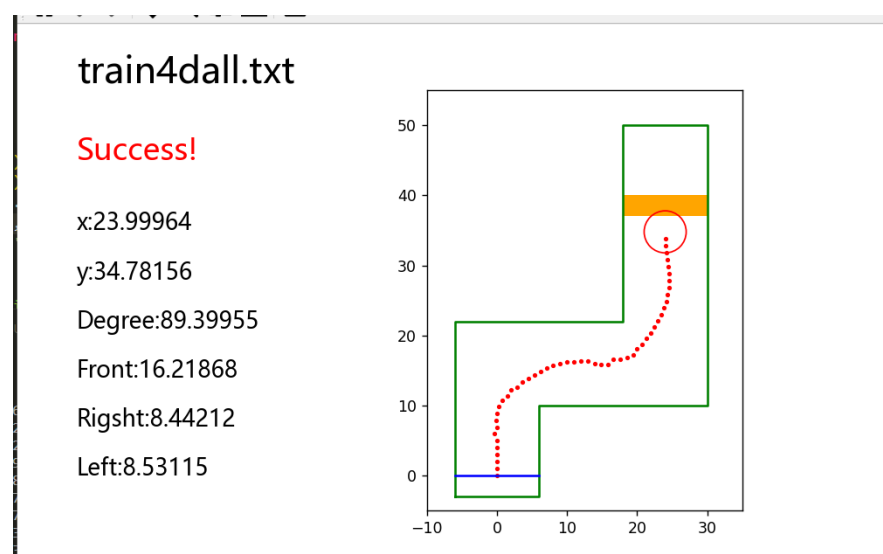
基於我們的訓練目的以及地形車體大小等等一系列因素都可能導致線性遞減成效較佳。

2. 車子位置的更改

```
car_x += math.cos(deg + predict)
car_y += math.sin(deg + predict)
```

不考慮車體的旋轉，可見下圖：

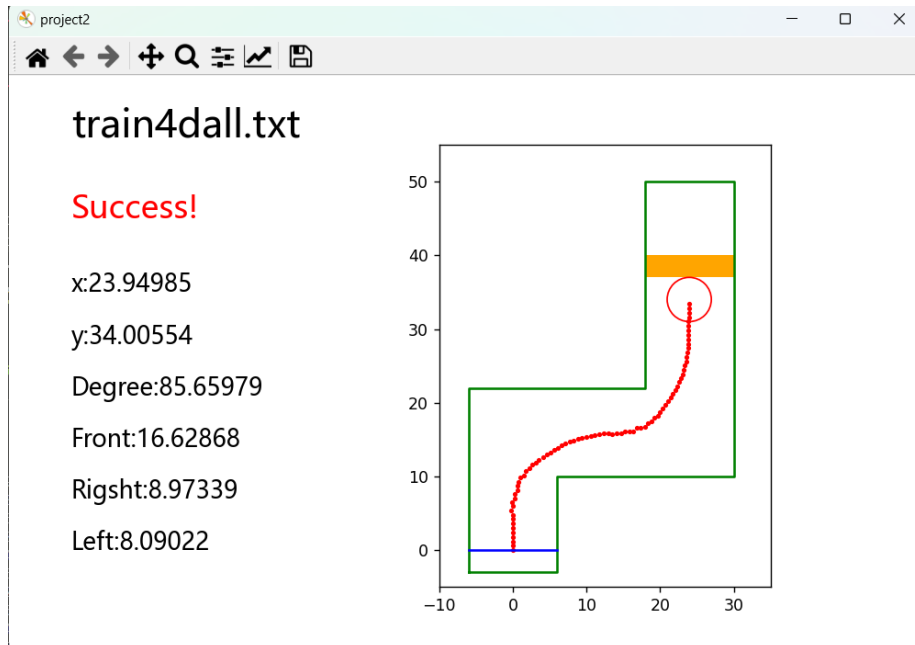
車子的行徑路徑會不夠平滑，從而導致失敗率的上升，且不符合真實情況




```
car_x += 0.6*math.cos(deg + predict)
car_y += 0.6*math.sin(deg + predict)
```

透過修改步數來改善，可見下圖：

不平整的狀況稍為解決了，成功率也不錯，但缺點是耗時以及計算量增長許多，且真實情況中，我們會期望更快的預測路徑

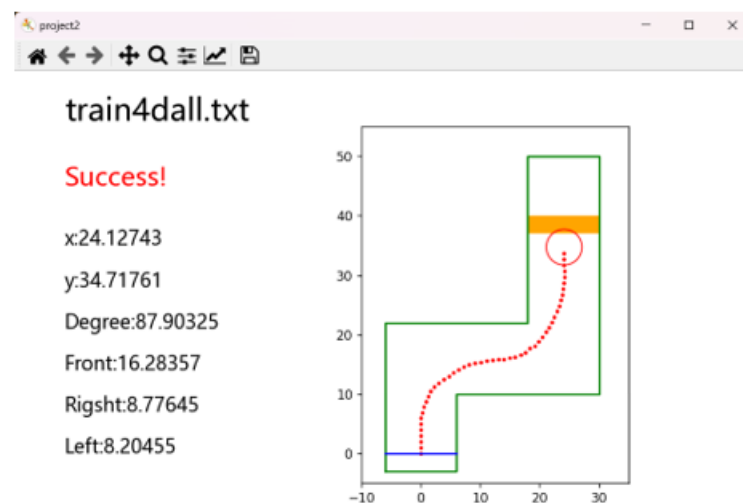


(最佳解)

```
car_x += math.cos(deg + predict) + math.sin(predict) * math.sin(deg) #考慮到車體的旋轉
car_y += math.sin(deg + predict) - math.sin(predict) * math.cos(deg)
```

考慮車的旋轉後，可見下圖

車子行徑路徑更為平滑，成功率上升，且花費步數更少，計算效率更快



3. 偏轉角度的修整

```
deg = (deg - math.asin(2 * math.sin(predict) )) % (math.pi * 2)
```

若不做任何減緩，我們可以從下圖看到車子行徑路近相當不穩定，從而導致失敗率大大上升

train4dall.txt

Please try again!

x:3.46697

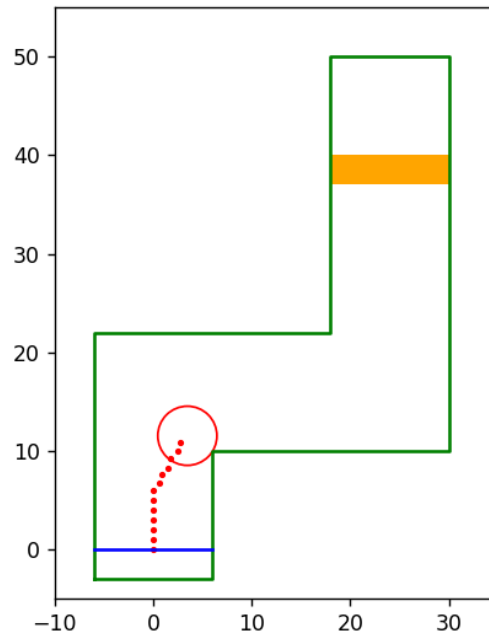
y:11.53663

Degree:28.31276

Front:16.81653

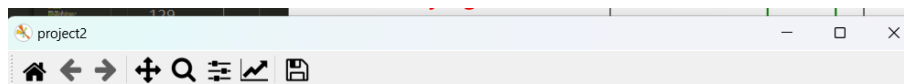
Rights:11.14242

Left:13.99842



```
deg = (deg - math.asin(2 * math.sin(predict)/10 )) % (math.pi * 2) #決定
```

若減緩得太多，又會導致車子轉彎的幅度太小，或者可以說反應太慢



train4dall.txt

Please try again!

x:27.90965

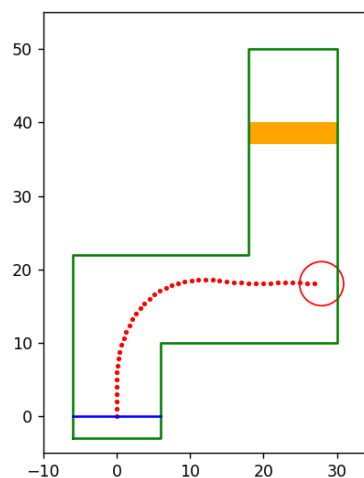
y:18.05321

Degree:356.16183

Front:3.09112

Rights:4.14885

Left:4.61942



4. 4D 與 6D 如何修改角度最好:

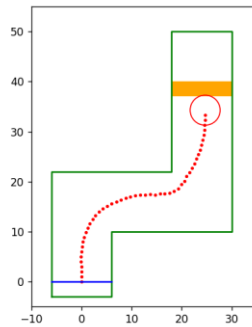
4D 只要在/2~/7 中成功率都是不錯的，而 6D 則是在/2~3 中，故預設為/3
這兩者在調整偏轉角度時的不同苛刻程度，可能的原因有: 6D 的輸入又接受到 x,y 這兩個輸入，又或者是訓練資料的特性，這或許使得他在預測角度時，需要更精確的區間來加以調整，4D 反之亦然

```
deg = (deg - math.asin(2 * math.sin(predict)/2 )) % (math.pi * 2) #
```

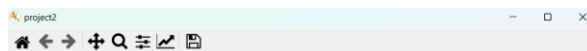
train6dall.txt

Success!

x:24.69285
y:34.26689
Degree:90.57573
Front:16.7331
Rigsht:9.49022
Left:7.48664



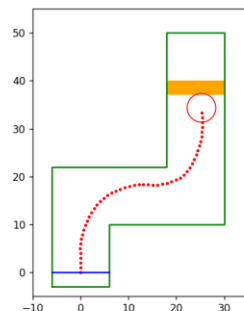
```
deg = (deg - math.asin(2 * math.sin(predict)/3 )) % (math.pi * 2) #決定
```



train6dall.txt

Success!

x:25.25596
y:34.3378
Degree:94.21245
Front:16.69635
Rigsht:9.73369
Left:7.09975



5. 4D 輸入該選擇甚麼樣不同的分群

實際上 K 群的抉擇需要套用到相關公式來求，但我只依照多次實驗的觀察，我發現到 4D 的這個數據集，並不需要太多的分群，便能執行足夠的效果，我認為這個數據集不存在太細緻或複雜的分群，甚至當我輸入 K=18 時，我有時會得出只有 17 個神經元，這代表有一個群集是沒有成員的。

既然對於群集分類的需求不高，此時我們可以選擇較小的 K 值，已達到更簡單有效率的模型，並且避免過度擬合、噪聲敏感性增加

```
4D.txt K: 10 Dsuceess rate: 18/20
```

```
4D.txt K: 15 Dsuceess rate: 17/20
```

6. 6D 輸入該選擇甚麼樣不同的分群

6D 輸入在選擇太小或太大(>15)的 K 值時，預測的效果有顯著的下降，分群太小

會過度簡化數據的結構，會讓車子在訊量數據上容易泛化，缺凡對複雜結構的適應，分群太大樣會造成過度擬合的問題，總之我們能透過 K 觀察到 4D 與 6D 在數據複雜度方面的差別。

```
6D.txt K: 10 Dsuceess rate: 13/20
*****
6D.txt K: 15 Dsuceess rate: 7/20
```

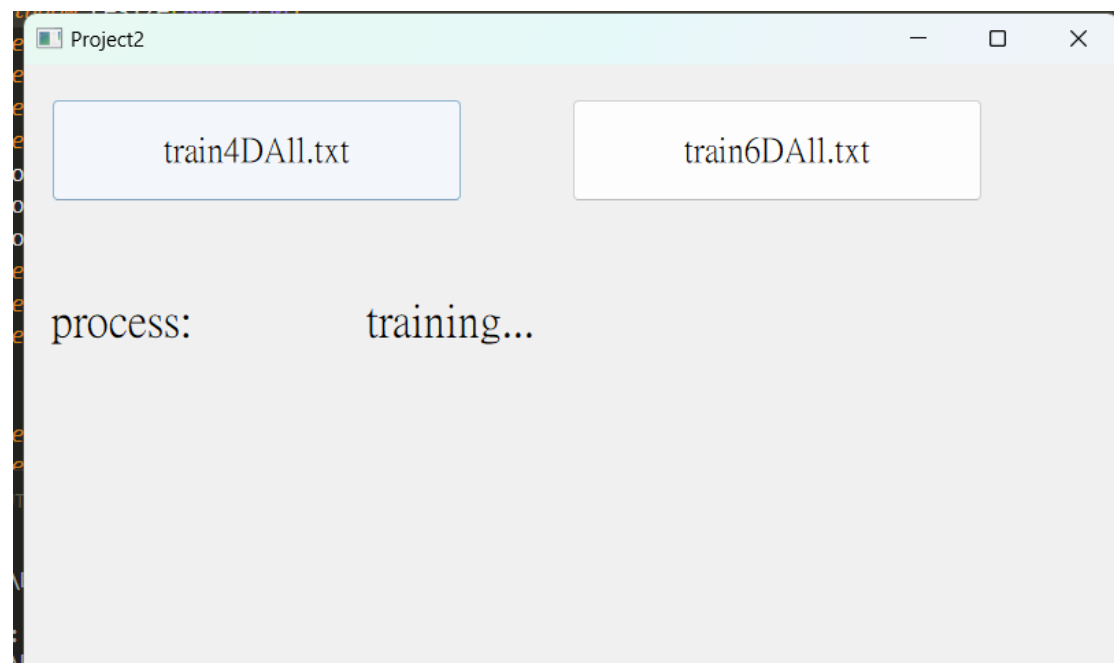
7. 4D 在總體的表現上成功率比 6D 高

6D 相比 4D 多了兩項輸入(X,Y)，表現的更差了，有以下幾種可能原因：

- (1) 輸入維度增加，樣本空間較大，導致樣本稀疏性增加。樣本點之間的距離可能變得極大，這使得模型難以找到有效的模式。
- (2) 新增的 (x, y) 輸入可能與現有特徵具有相關性，有可能會導致冗餘信息，不只會增加模型的複雜度，也可能降低本來的效能。
- (3) 模型的複雜度不足以處理高維數據。
- (4) 新增的特徵選擇不當，或許(X,Y)特徵對我們的訓練是反效果。

E. 失敗過程與嘗試:

原本打算做一個 GUI 讓使用者選擇檔案，並且看到程式執行的過程，如:初始分群、訓練、圖形產生中、並將結果顯示出。



但過程中無法成功 debug 多線程的問題:

```
QCoreApplication::exec: The event loop is already running
```

所以之後一律改為從終端機看其中過程