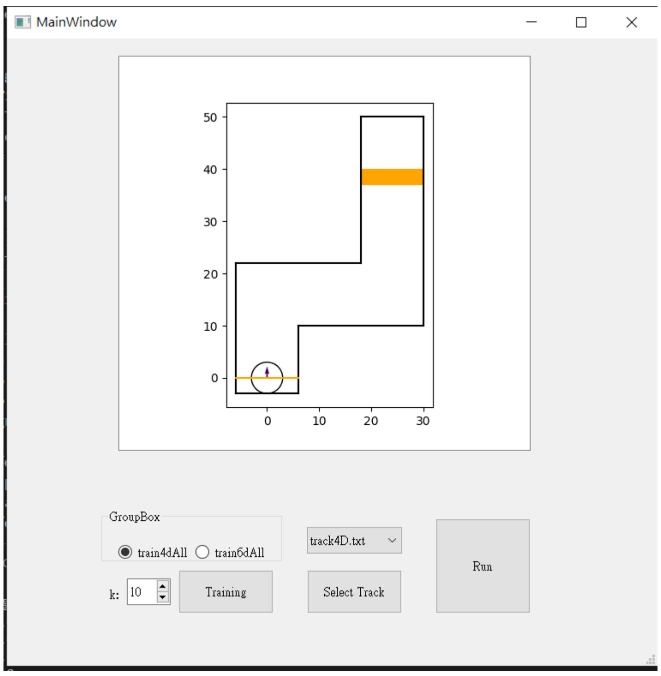
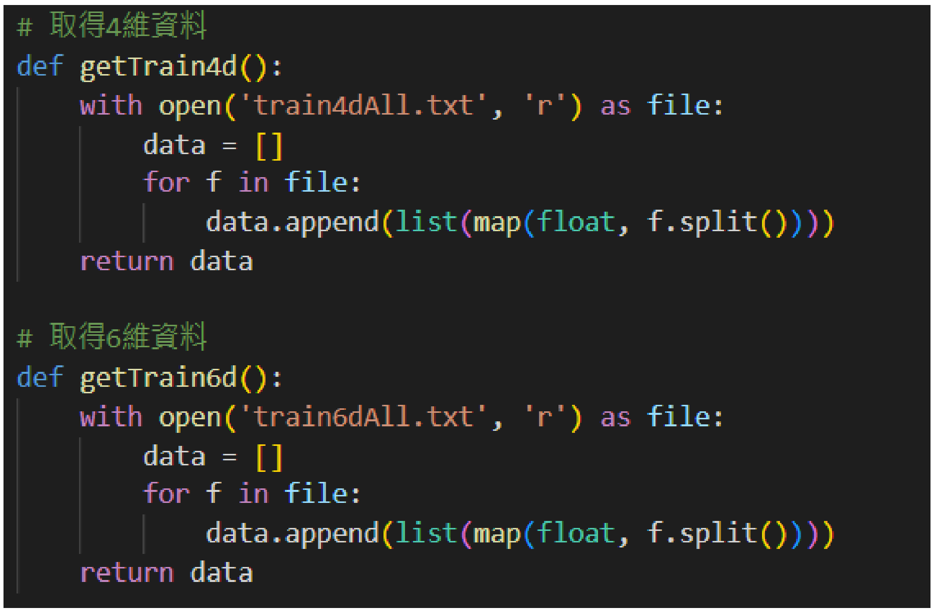
1. 程式介面說明



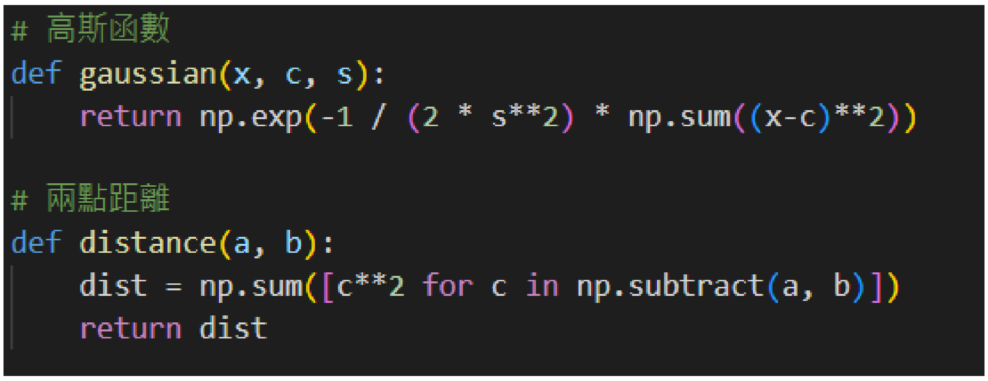
選擇欲訓練的資料集並更改欲分群的數目後，按下”Training”按鈕，等待 RBFN 模型訓練完成，後按下”Run”按鈕一次，便能執行預測結果，直至車子撞牆或是抵達終點，若發現車子撞牆、抵達終點或是純粹想再訓練一次模型可再次按下”Training” 待 RBFN 模型訓練完成後按下”Run”車子便會再次回到起點，或是可以選擇”track4D.txt”或”track6D.txt”後按下”Select Track”按鈕將過去成功走到終點的例子匯入，再按下”Run”按鈕一次，便會執行過去成功走到終點例子

2. 程式碼說明



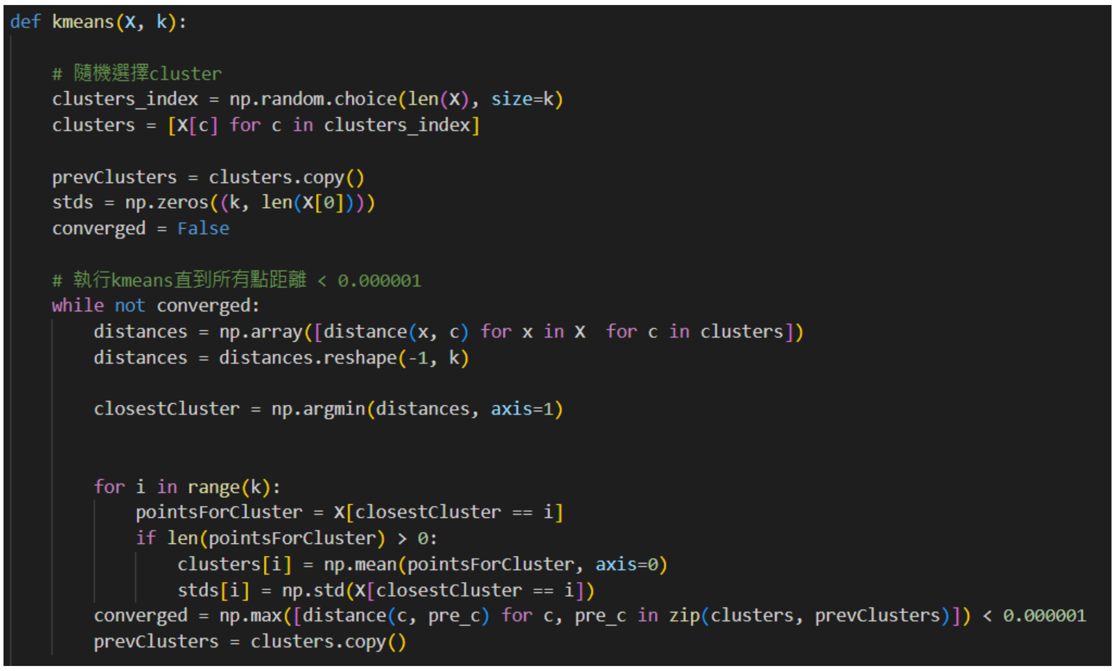
```python
# 取得4維資料
def getTrain4d():
    with open('train4dAll.txt', 'r') as file:
        data = []
        for f in file:
            data.append(list(map(float, f.split())))
    return data

# 取得6維資料
def getTrain6d():
    with open('train6dAll.txt', 'r') as file:
        data = []
        for f in file:
            data.append(list(map(float, f.split())))
    return data
```

取得訓練資料集



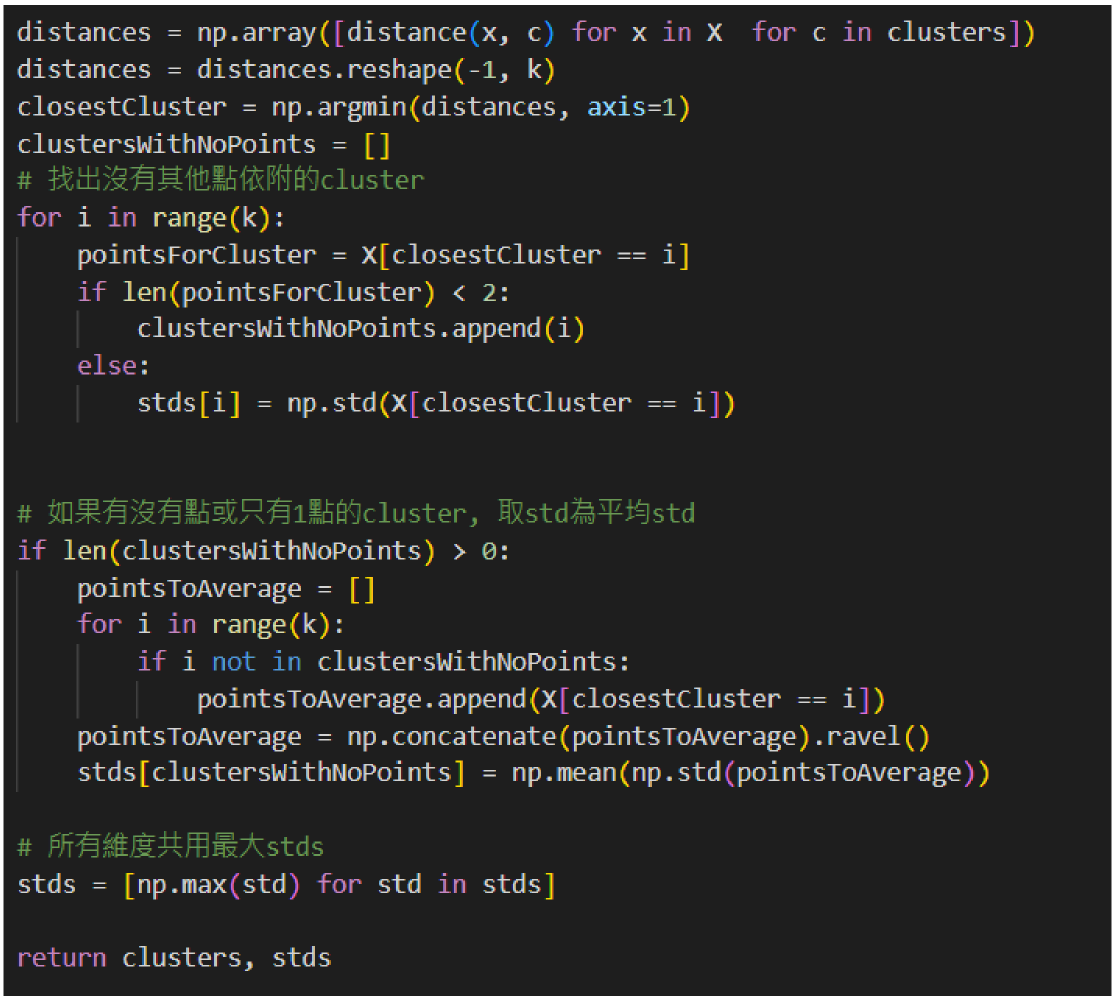
```python
# 高斯函數
def gaussian(x, c, s):
    return np.exp(-1 / (2 * s**2) * np.sum((x-c)**2))

# 兩點距離
def distance(a, b):
    dist = np.sum([c**2 for c in np.subtract(a, b)])
    return dist
```

高斯函數與計算兩點距離



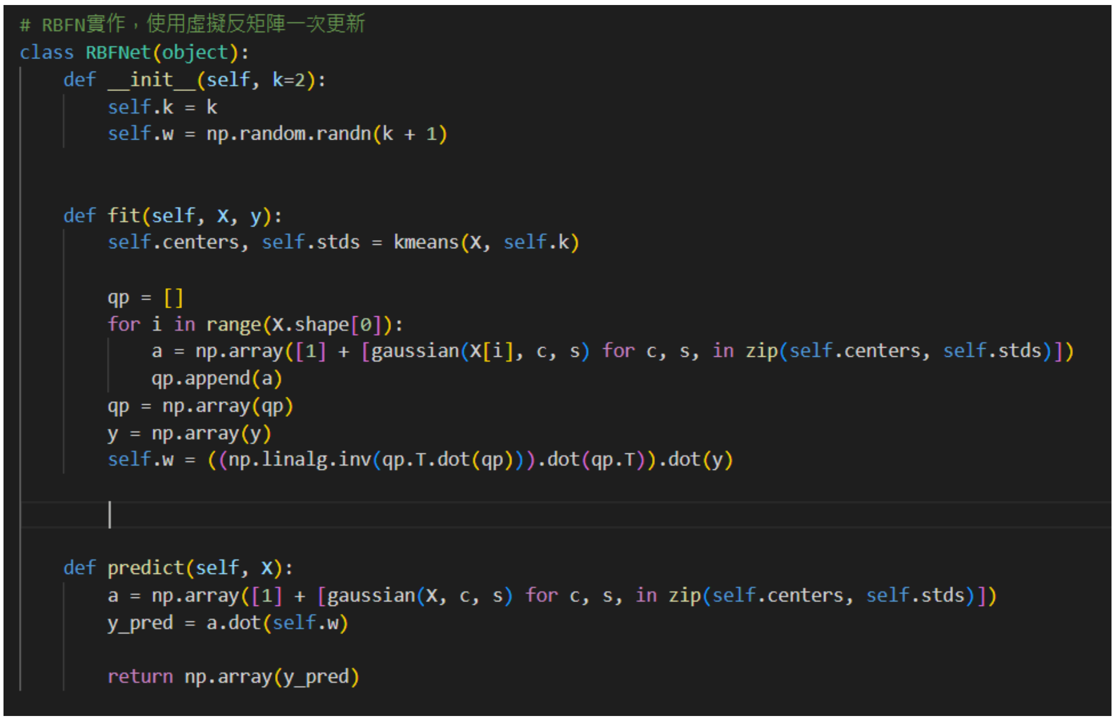
```python
def kmeans(X, k):

    # 隨機選擇cluster
    clusters_index = np.random.choice(len(X), size=k)
    clusters = [X[c] for c in clusters_index]

    prevClusters = clusters.copy()
    stds = np.zeros((k, len(X[0])))
    converged = False

    # 執行kmeans直到所有點距離 < 0.000001
    while not converged:
        distances = np.array([distance(x, c) for x in X  for c in clusters])
        distances = distances.reshape(-1, k)

        closestCluster = np.argmin(distances, axis=1)


        for i in range(k):
            pointsForCluster = X[closestCluster == i]
            if len(pointsForCluster) > 0:
                clusters[i] = np.mean(pointsForCluster, axis=0)
                stds[i] = np.std(X[closestCluster == i])
        converged = np.max([distance(c, pre_c) for c, pre_c in zip(clusters, prevClusters)]) < 0.000001
        prevClusters = clusters.copy()
```

計算 kmeans part1 直到所有點距離<0.000001 才停止



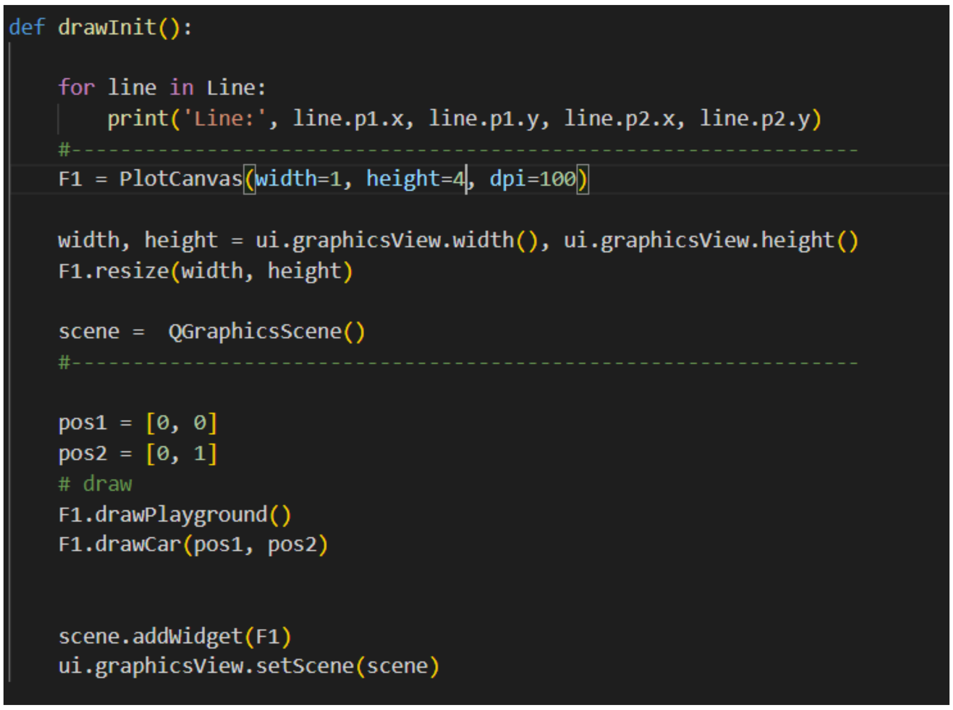
```python
distances = np.array([distance(x, c) for x in X  for c in clusters])
distances = distances.reshape(-1, k)
closestCluster = np.argmin(distances, axis=1)
clustersWithNoPoints = []
# 找出沒有其他點依附的cluster
for i in range(k):
    pointsForCluster = X[closestCluster == i]
    if len(pointsForCluster) < 2:
        clustersWithNoPoints.append(i)
    else:
        stds[i] = np.std(X[closestCluster == i])


# 如果有沒有點或只有1點的cluster, 取std為平均std
if len(clustersWithNoPoints) > 0:
    pointsToAverage = []
    for i in range(k):
        if i not in clustersWithNoPoints:
            pointsToAverage.append(X[closestCluster == i])
    pointsToAverage = np.concatenate(pointsToAverage).ravel()
    stds[clustersWithNoPoints] = np.mean(np.std(pointsToAverage))

# 所有維度共用最大stds
stds = [np.max(std) for std in stds]

return clusters, stds
```

計算 kmeans part2 計算 cluster 的標準差，其中有些 cluster 可能 kmeans 分群設太多沒有其他點能計算標準差，就取平均標準差，最後把所有維度的標準差取最大者為準減少複雜度，回傳群聚中心與標準差



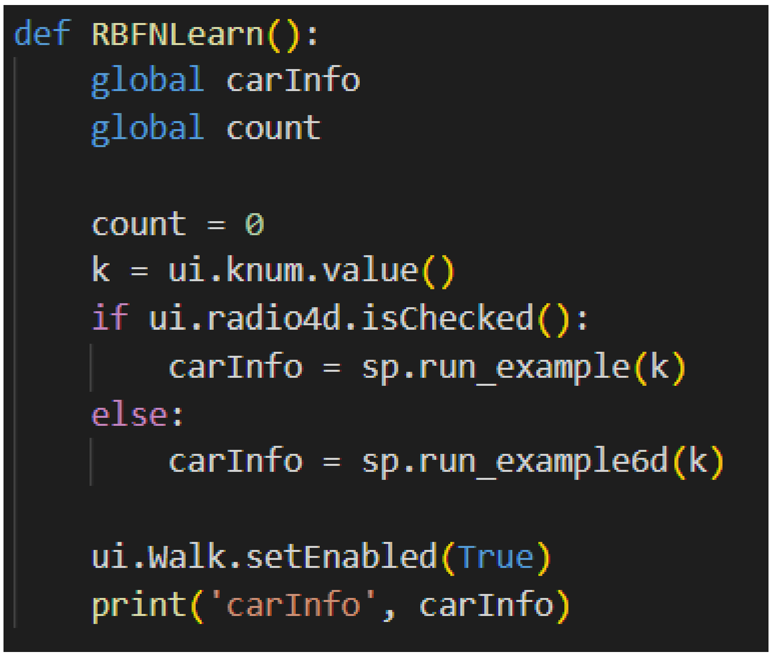
```python
# RBFN實作，使用虛擬反矩陣一次更新
class RBFNet(object):
    def __init__(self, k=2):
        self.k = k
        self.w = np.random.randn(k + 1)


    def fit(self, X, y):
        self.centers, self.stds = kmeans(X, self.k)

        qp = []
        for i in range(X.shape[0]):
            a = np.array([1] + [gaussian(X[i], c, s) for c, s, in zip(self.centers, self.stds)])
            qp.append(a)
        qp = np.array(qp)
        y = np.array(y)
        self.w = ((np.linalg.inv(qp.T.dot(qp))).dot(qp.T)).dot(y)



    def predict(self, X):
        a = np.array([1] + [gaussian(X, c, s) for c, s, in zip(self.centers, self.stds)])
        y_pred = a.dot(self.w)

        return np.array(y_pred)
```

實作 RBFN， fit() -> 使用虛擬反矩陣訓練權重
predict() -> 用訓練好的權重預測結果



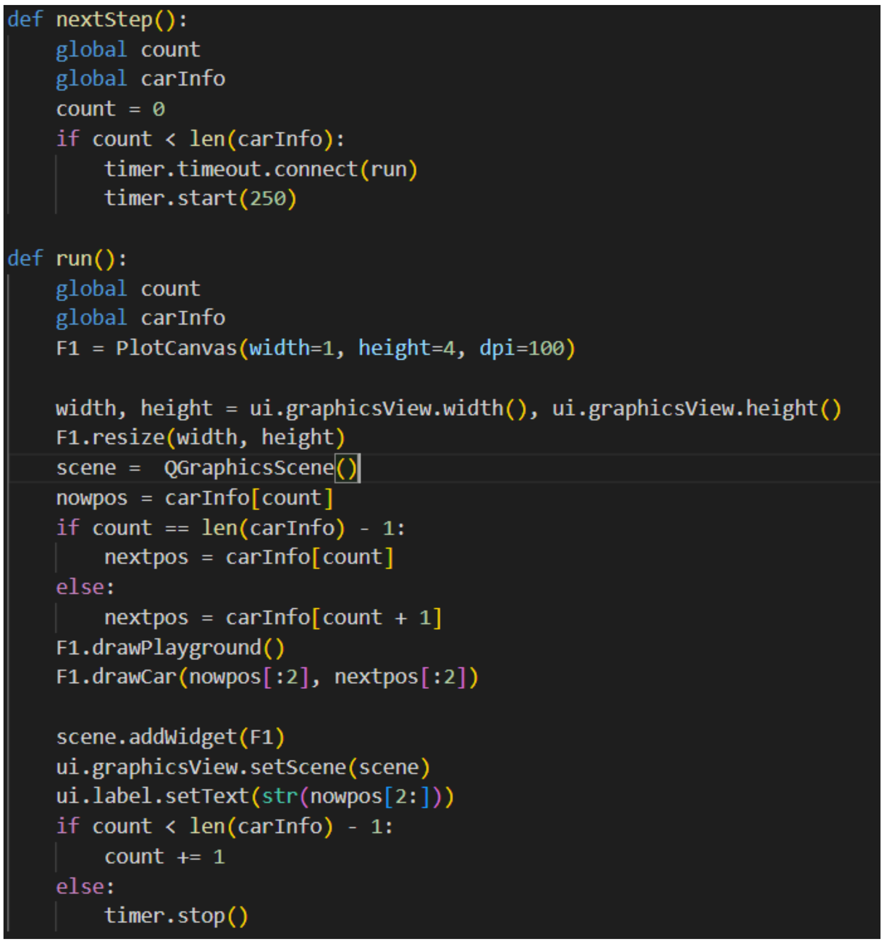
```python
def drawInit():

    for line in Line:
        print('Line:', line.p1.x, line.p1.y, line.p2.x, line.p2.y)
    #-----------------------------------------------------------------
    F1 = PlotCanvas(width=1, height=4, dpi=100)

    width, height = ui.graphicsView.width(), ui.graphicsView.height()
    F1.resize(width, height)

    scene =  QGraphicsScene()
    #-----------------------------------------------------------------

    pos1 = [0, 0]
    pos2 = [0, 1]
    # draw
    F1.drawPlayground()
    F1.drawCar(pos1, pos2)


    scene.addWidget(F1)
    ui.graphicsView.setScene(scene)
```

畫出初始畫面



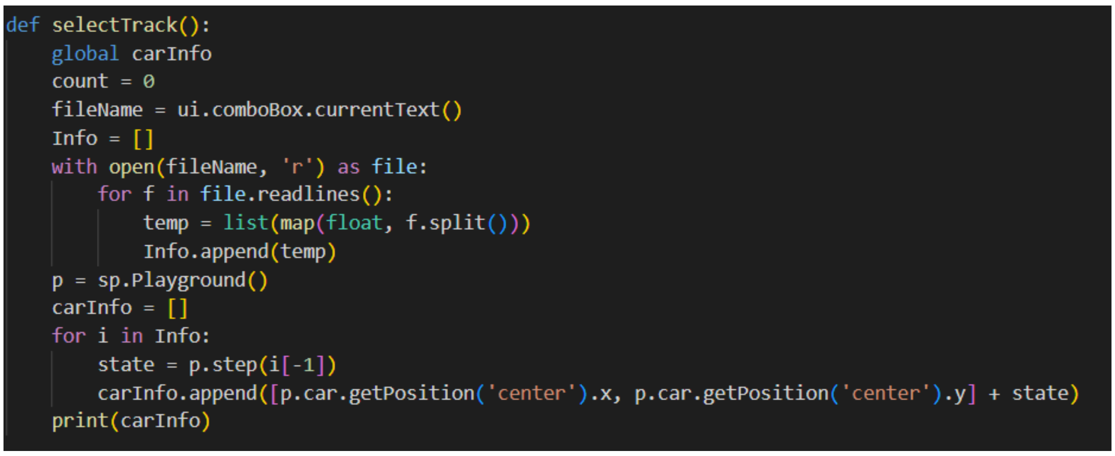
```python
def RBFNLearn():
    global carInfo
    global count

    count = 0
    k = ui.knum.value()
    if ui.radio4d.isChecked():
        carInfo = sp.run_example(k)
    else:
        carInfo = sp.run_example6d(k)

    ui.Walk.setEnabled(True)
    print('carInfo', carInfo)
```

“Training”按鈕實作，根據使用者選擇欲跑的資料集，以及取得欲分群數目，訓練 RBFN 模型



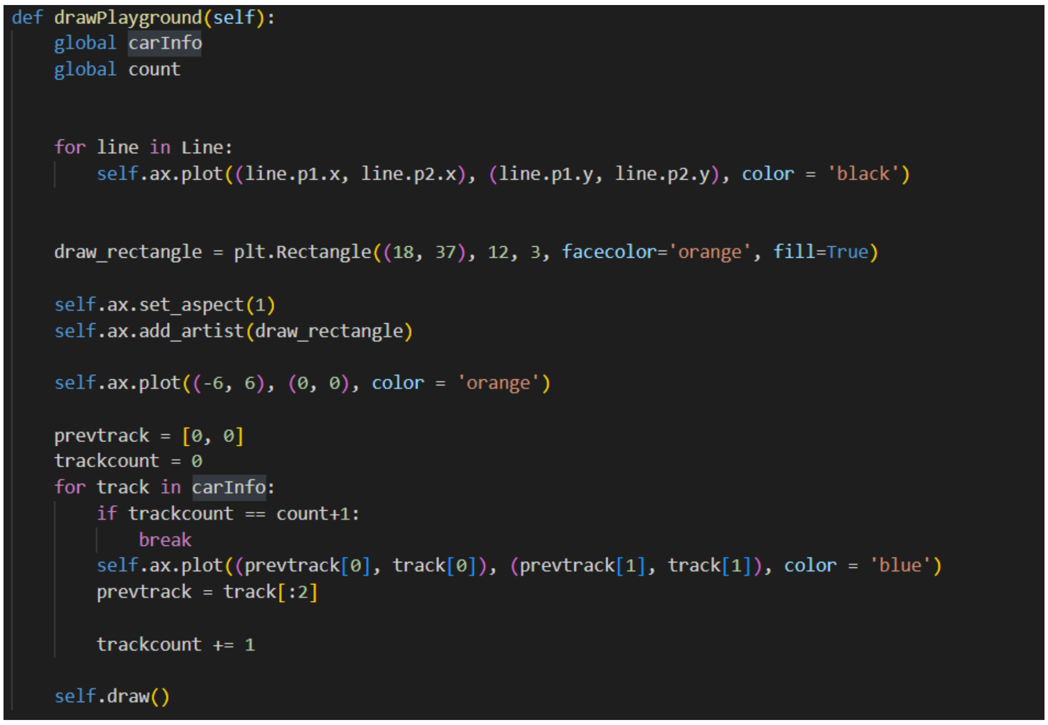
```python
def nextStep():
    global count
    global carInfo
    count = 0
    if count < len(carInfo):
        timer.timeout.connect(run)
        timer.start(250)

def run():
    global count
    global carInfo
    F1 = PlotCanvas(width=1, height=4, dpi=100)

    width, height = ui.graphicsView.width(), ui.graphicsView.height()
    F1.resize(width, height)
    scene =  QGraphicsScene()
    nowpos = carInfo[count]
    if count == len(carInfo) - 1:
        nextpos = carInfo[count]
    else:
        nextpos = carInfo[count + 1]
    F1.drawPlayground()
    F1.drawCar(nowpos[:2], nextpos[:2])

    scene.addWidget(F1)
    ui.graphicsView.setScene(scene)
    ui.label.setText(str(nowpos[2:]))
    if count < len(carInfo) - 1:
        count += 1
    else:
        timer.stop()
```

“Run”按鈕實作，畫出軌跡與更新車子位置



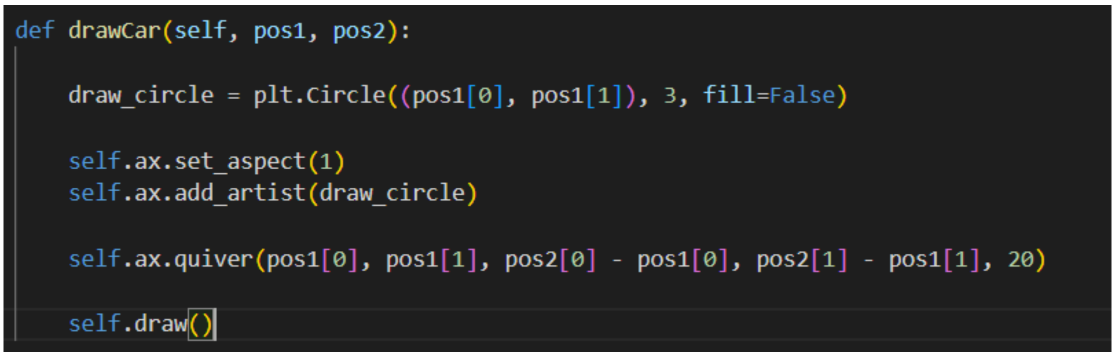
```python
def selectTrack():
    global carInfo
    count = 0
    fileName = ui.comboBox.currentText()
    Info = []
    with open(fileName, 'r') as file:
        for f in file.readlines():
            temp = list(map(float, f.split()))
            Info.append(temp)
    p = sp.Playground()
    carInfo = []
    for i in Info:
        state = p.step(i[-1])
        carInfo.append([p.car.getPosition('center').x, p.car.getPosition('center').y] + state)
    print(carInfo)
```

“Select track”按鈕實作，選擇欲顯示的成功走到終點紀錄



```python
def drawPlayground(self):
    global carInfo
    global count


    for line in Line:
        self.ax.plot((line.p1.x, line.p2.x), (line.p1.y, line.p2.y), color = 'black')


    draw_rectangle = plt.Rectangle((18, 37), 12, 3, facecolor='orange', fill=True)

    self.ax.set_aspect(1)
    self.ax.add_artist(draw_rectangle)

    self.ax.plot((-6, 6), (0, 0), color = 'orange')

    prevtrack = [0, 0]
    trackcount = 0
    for track in carInfo:
        if trackcount == count+1:
            break
        self.ax.plot((prevtrack[0], track[0]), (prevtrack[1], track[1]), color = 'blue')
        prevtrack = track[:2]

        trackcount += 1

    self.draw()
```

畫出跑道

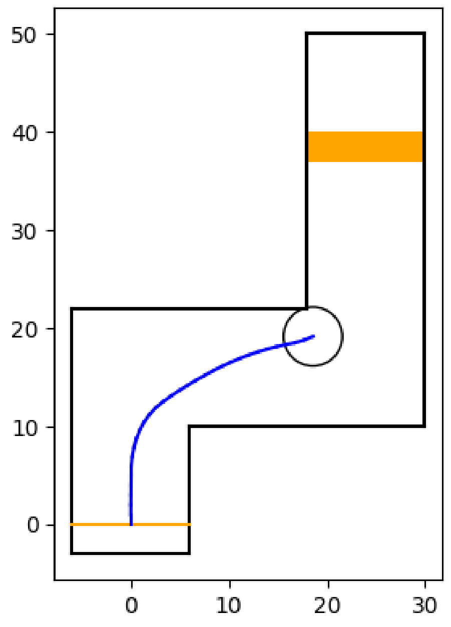


```python
def drawCar(self, pos1, pos2):

    draw_circle = plt.Circle((pos1[0], pos1[1]), 3, fill=False)

    self.ax.set_aspect(1)
    self.ax.add_artist(draw_circle)

    self.ax.quiver(pos1[0], pos1[1], pos2[0] - pos1[0], pos2[1] - pos1[1], 20)

    self.draw()
```

畫出小車

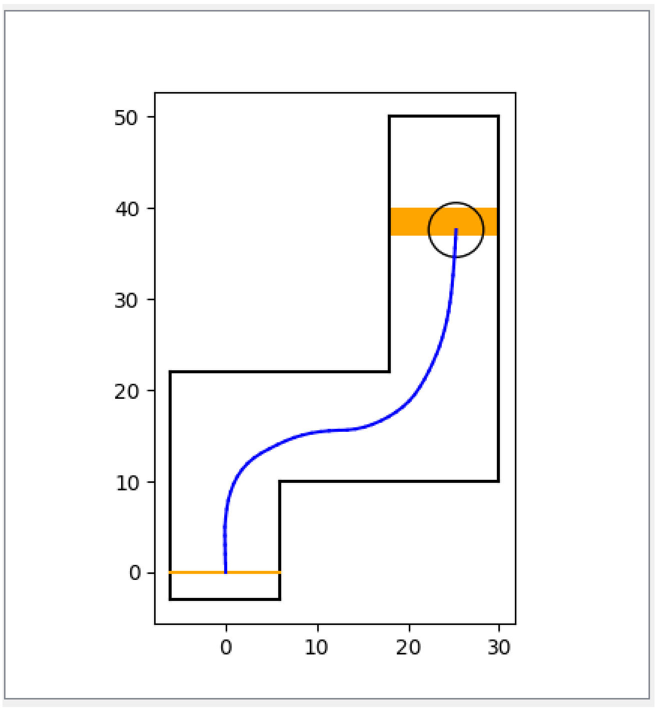
3. 實驗結果

k = 5, train4d



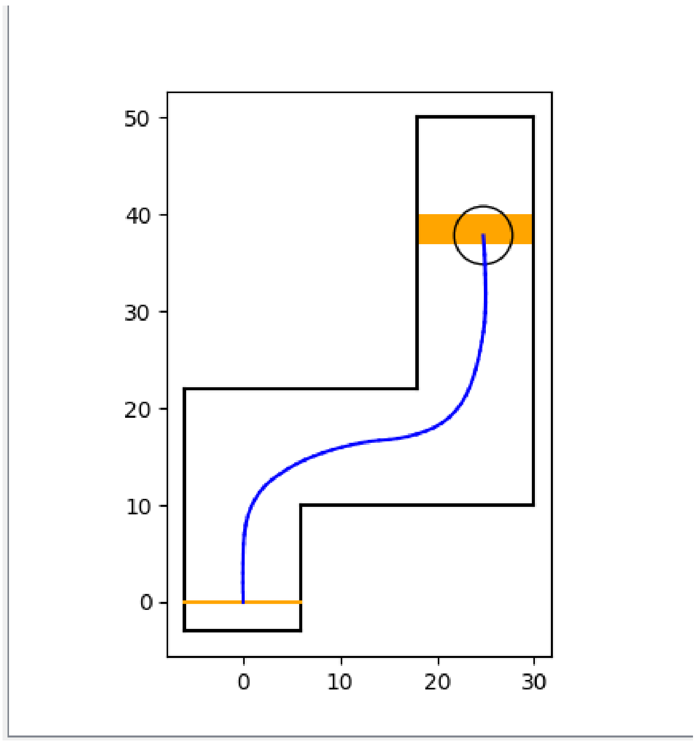
車子很容易撞牆，幾乎無法走到終點

k = 5, train6d



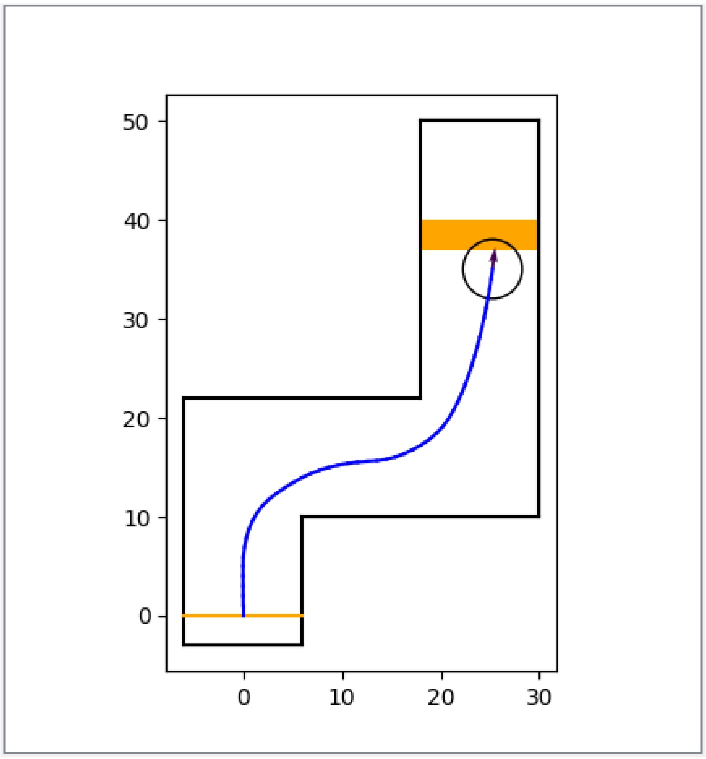
令人意外地相較於 train4d 當 k=5 時 train6d 有高於一半的機會能訓練成功

k = 8, train4d



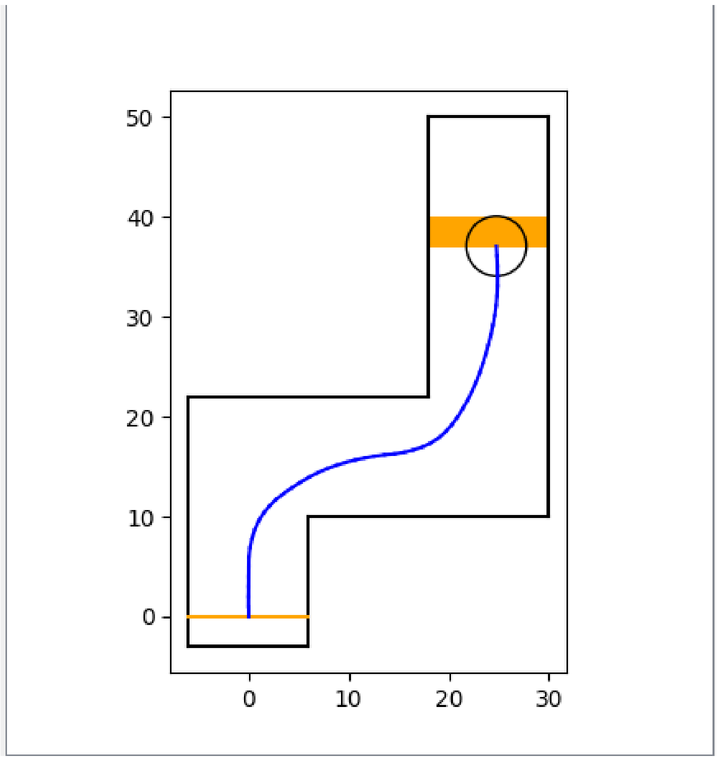
成功率要 k = 5 時高出許多，但仍有機會撞牆

k = 8, train6d



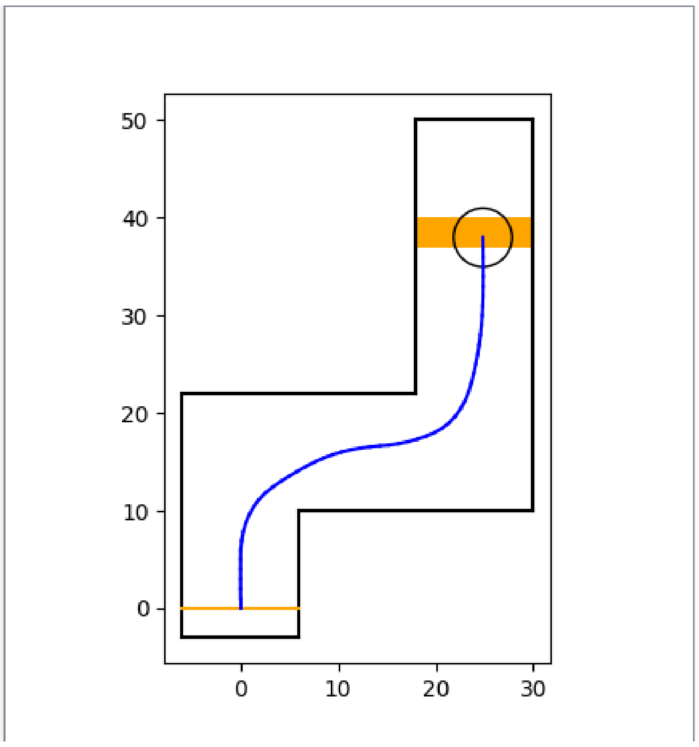
成功率要 k = 5 時高，已經幾乎不會撞牆

k = 10, train4d



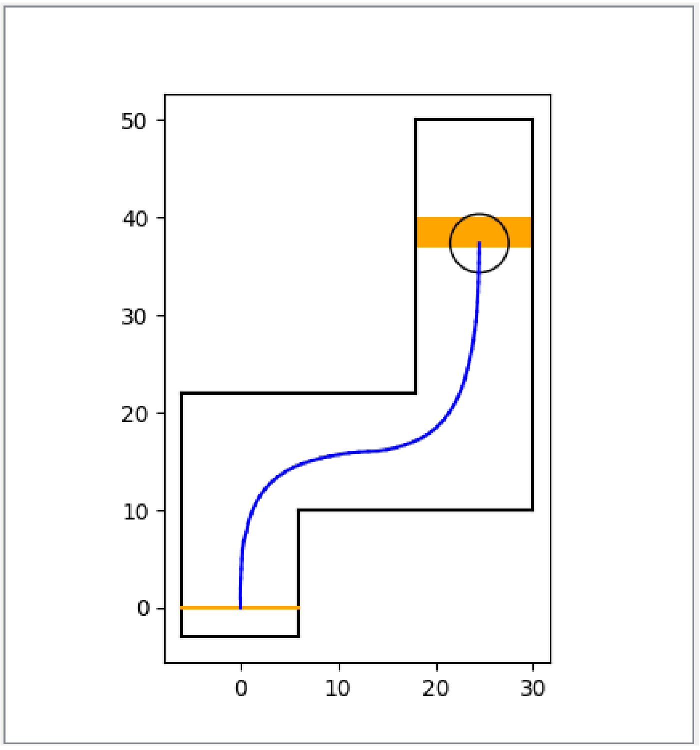
已經幾乎不會撞牆

k = 10, train6d



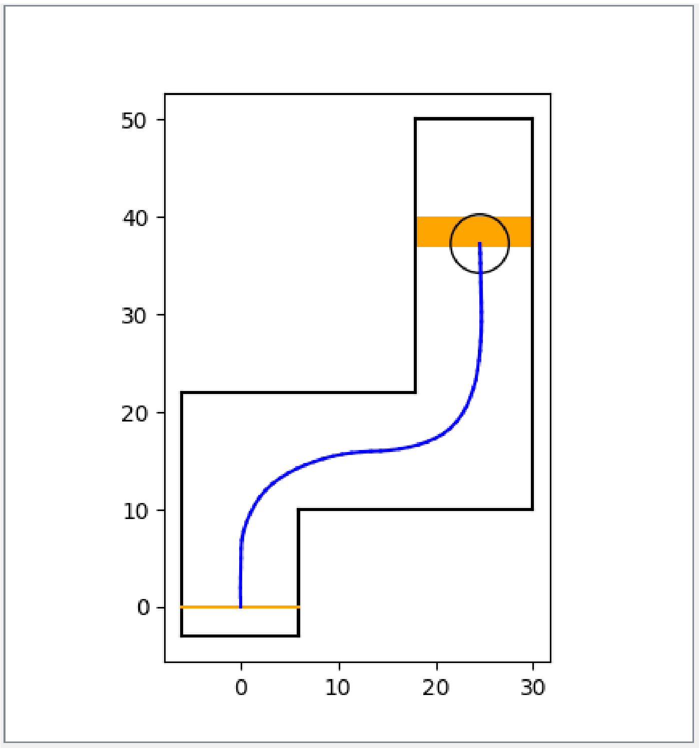
已經幾乎不會撞牆

k = 60, train4d



隨著 k 的增大，計算時間也愈來愈久，同樣已經幾乎不會撞牆

k = 60, train4d



同 4d 時的情況，隨著 k 的增大，計算時間也愈來愈久，同樣已經幾乎不會撞牆

4. 分析

選用一次性更新的虛擬反矩陣作為 RBFN 的訓練方式時，程式的時間複雜度都落在 kmeans 的運作上，而 kmeans 的分群數量亦會很大程度上的影響訓練的準確度與訓練時間，因此對欲訓練的資料集的觀察與分群數目的選擇對能否訓練好 RBFN 來說是相當重要的。