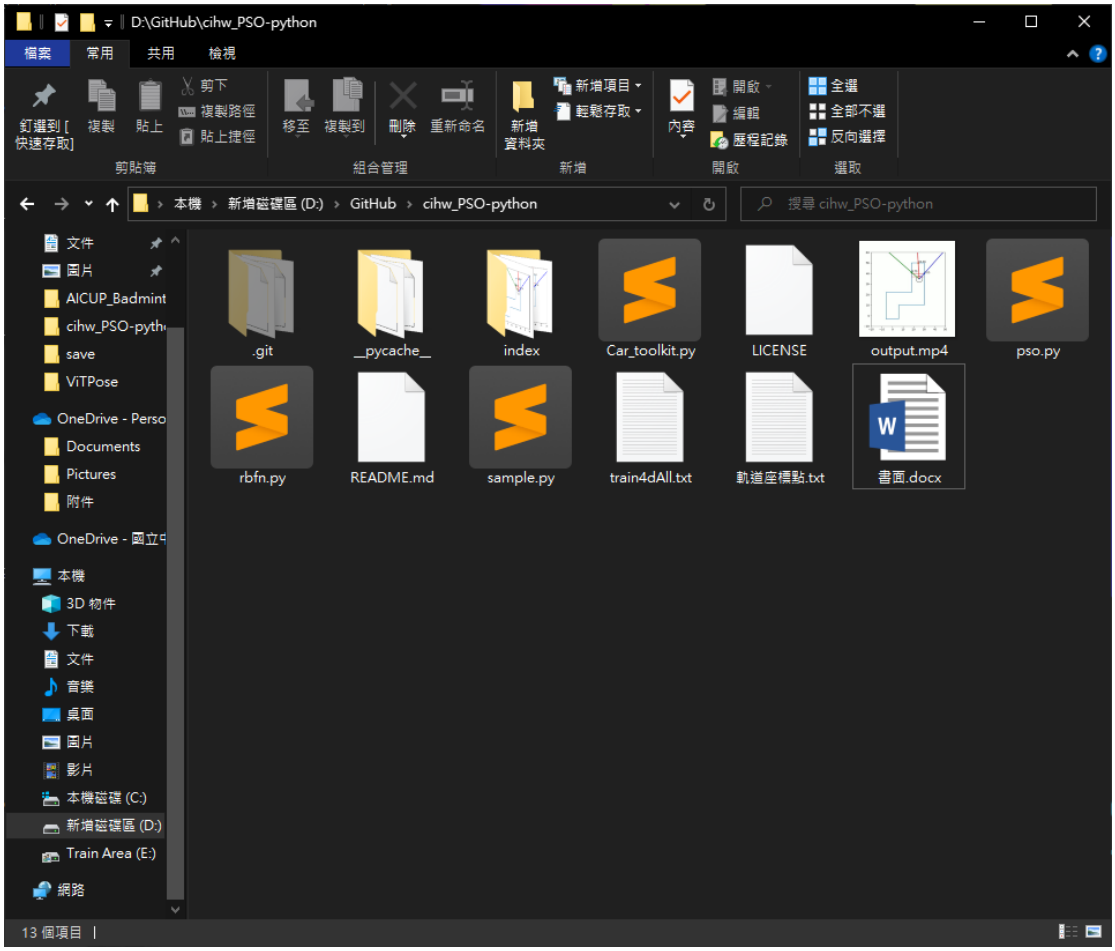
PSO

一、介面說明:

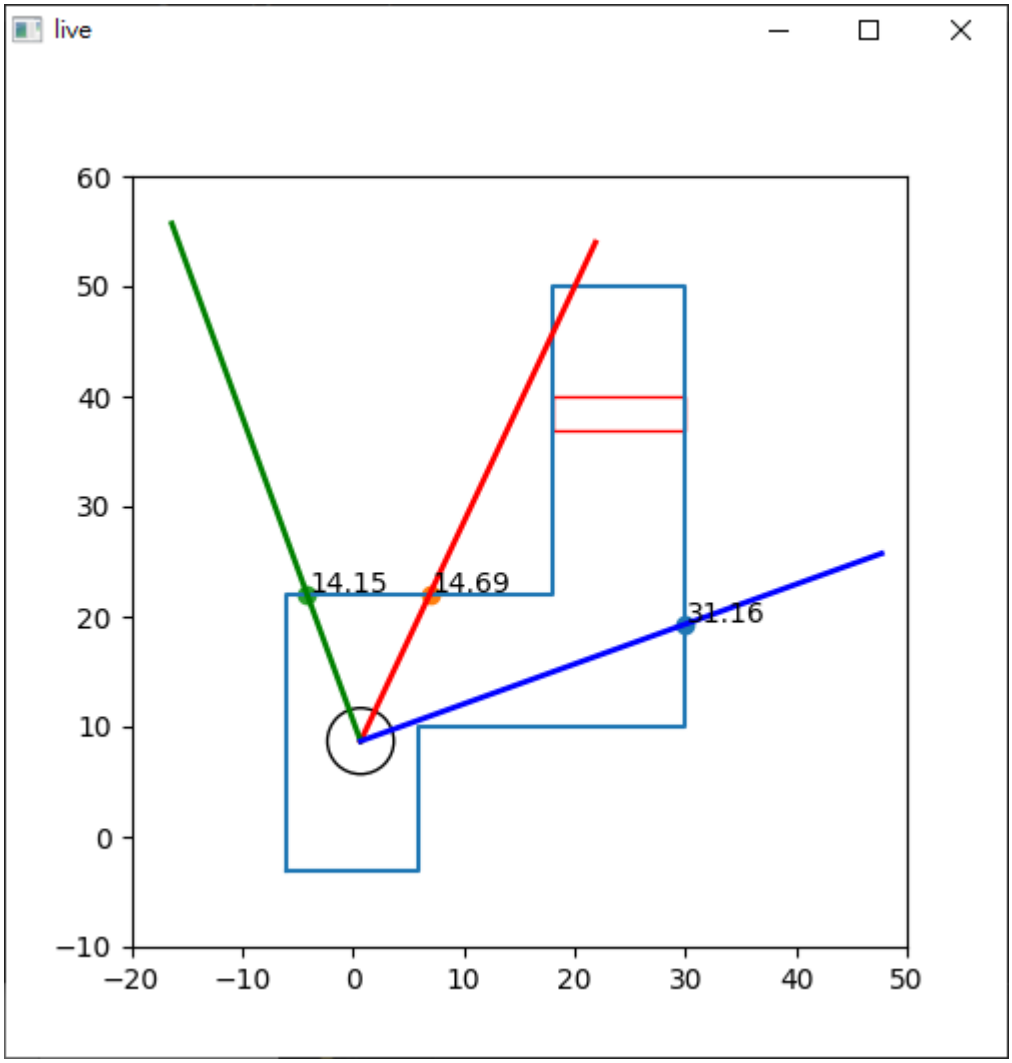
執行檔:sample.exe

截圖存放目錄:index

影片存放目錄:output.mp4



執行後會自動進入基因演算法，運算完成後將車輛軌跡顯示於 UI。



二、實驗結果

詳見圖片目錄./index 與影片目錄./output.mp4

三、基因演算法實作細節

RBFN 程式碼:

```python
class RBFN:
    def __init__(self, input_dim, hidden_dim, output_dim, sigma=40):
        self.input_dim = input_dim
        self.hidden_dim = hidden_dim
        self.output_dim = output_dim
        self.sigma = sigma

        self.centers = None
        self.weights = None

    def _rbf(self, X, centers):
        distances = np.linalg.norm(X[:, None, :] - centers[None, :, :], axis=-1)
        return np.exp(-distances**2 / (2*self.sigma**2))

    def train(self, X, y):
        idx = np.random.choice(len(X), self.hidden_dim, replace=False)
        self.centers = X[idx]
        RBF_outputs = self._rbf(X, self.centers)
        self.weights = np.linalg.lstsq(RBF_outputs, y, rcond=None)[0]

    def getDNA(self):
        self.DNA = np.concatenate((self.weights, np.reshape(self.centers, (30,)) ))
        return self.DNA

    def setDNA(self, DNA):
        self.weights, self.centers = np.split(DNA, [10])
        self.centers = np.reshape(self.centers, (10, 3))

    def predict(self, X):
        RBF_outputs = self._rbf(X, self.centers)
        y_pred = np.dot(RBF_outputs, self.weights)
        return y_pred
```

RBFN 輸入為(3,)，layer 為 10，故共有 weights(10,)與 center(3, 10)，兩個參數陣列，由於基引演算法需要使用一維陣列做計算，故此程式在基於 RBFN 的基礎上新增了 getDNA 與 setDNA 兩個 function。

getDNA()用於將上述參數(weight, center)平坦化{ weights 不變，center reshape 為(30,) }組合後回傳。{ weights(10,)+flatten center(30,) -> DNA(40,) }

setDNA()用於將優化後的基因還原為原始的 weights 與 center，為 getDNA()的反函數。

PSO 程式碼:

```python
class PSO:
    def __init__(self, objective_func, DNA, max_iter, Vmin, Vmax):
        self.objective_func = objective_func
        self.num_particles = DNA.shape[0]
        self.num_dimensions = DNA.shape[1]
        self.Vmin = Vmin
        self.Vmax = Vmax
        self.max_iter = max_iter
        self.particles_position = DNA
        self.particles_velocity = np.zeros(DNA.shape)
        self.particles_best_position = self.particles_position.copy()
        self.particles_best_value = np.zeros(self.num_particles)
        self.global_best_position = np.zeros(self.num_dimensions)
        self.global_best_value = float('inf')

    def optimize(self):
        for i in range(self.max_iter):
            for j in range(self.num_particles):
                self.particles_best_value[j] = self.objective_func(self.particles_best_position[j])

            for j in range(self.num_particles):
                if self.particles_best_value[j] < self.global_best_value:
                    self.global_best_value = self.particles_best_value[j]
                    self.global_best_position = self.particles_best_position[j].copy()

            for j in range(self.num_particles):
                r1 = np.random.random(self.num_dimensions)
                r2 = np.random.random(self.num_dimensions)
                self.particles_velocity[j] = self.particles_velocity[j] + 2.0 * r1 * (self.particles_best_position[j] - self.particles_position[j]) \
                                             + 2.0 * r2 * (self.global_best_position - self.particles_position[j])

                self.particles_velocity[j] = np.clip(self.particles_velocity[j], self.Vmin, self.Vmax)
                self.particles_position[j] = self.particles_position[j] + self.particles_velocity[j]



        return self.global_best_position, self.global_best_value

def fitness_func(individual):
    return computeScore(individual)

def getbestDNA(DNA, max_iter, Vmin, Vmax):
    pso = PSO(fitness_func, DNA, max_iter, Vmin, Vmax)
    best_position, best_value = pso.optimize()
    print('best position:')
    print(best_position)
    return best_position
```

此實驗的，max_iter 為 10，Vmin 為 -10，Vmax 為 10。

```python
def computeScore(DNA):
    rbfn = RBFN(input_dim=3, hidden_dim=10, output_dim=1)
    rbfn.setDNA(DNA)

    square, car_pos, endline = ct.getSquare("軌道座標點.txt")
    frame=0
    s=0
    while (not(ct.inBox(car_pos[0], car_pos[1])))and frame<100:
        sX = square[:, 0]; sY = square[:, 1]
        try:
            sensors, min_ds, min_ds_point = ct.draw_sensors(car_pos[0], car_pos[1], car_pos[2], square)
        except:
            break
        rd, fd, ld = min_ds
        sensors_np = np.array([[fd, rd, ld]])
        theta = rbfn.predict(sensors_np)[0]
        car_pos = ct.nextPos(car_pos[0], car_pos[1], car_pos[2], theta)
        #d = rd+fd+ld
        #s+=d

        if ct.inBox(car_pos[0], car_pos[1]):
            return(300-frame)
        frame+=1
    return frame
```

Fitness function 基於車輛行駛的總步數設計，由於此實驗將步數上限訂為 99，完整 function 如下：

令 f 代表該基因行駛的總步數
Score = 300-f,若成功抵達終點
Score = f, 若未成功抵達終點

此函數能很好的在車輛未成功抵達終點時增加存活時間，並在成功抵達終點時縮短行駛步數。

取得最佳基因程式:

```
for i in range(population_size):
    rbfn = RBFN(input_dim=3, hidden_dim=10, output_dim=1)
    rbfn.train(X, y)
    wList.append(rbfn.getDNA().tolist())

w_arr = np.array(wList)
best_w = getbestDNA(w_arr, 10, -10, 10)
rbfn = RBFN(input_dim=3, hidden_dim=10, output_dim=1)
rbfn.setDNA(best_w)
```

綜上所述，設 population_size 為 10，定義 population_size 個 layer 為 10 的 RBFN 網路(為節省時間預先進行了 pretrain)，並使用 getDNA()將基因取出，之後使用 PSO 得到 bestDNA，最後使用 setDNA 將該基因重組為 RBFN 網路即完成。