計算型智慧 作業二

110403518 資工 4B 林晉宇

一、程式介面說明

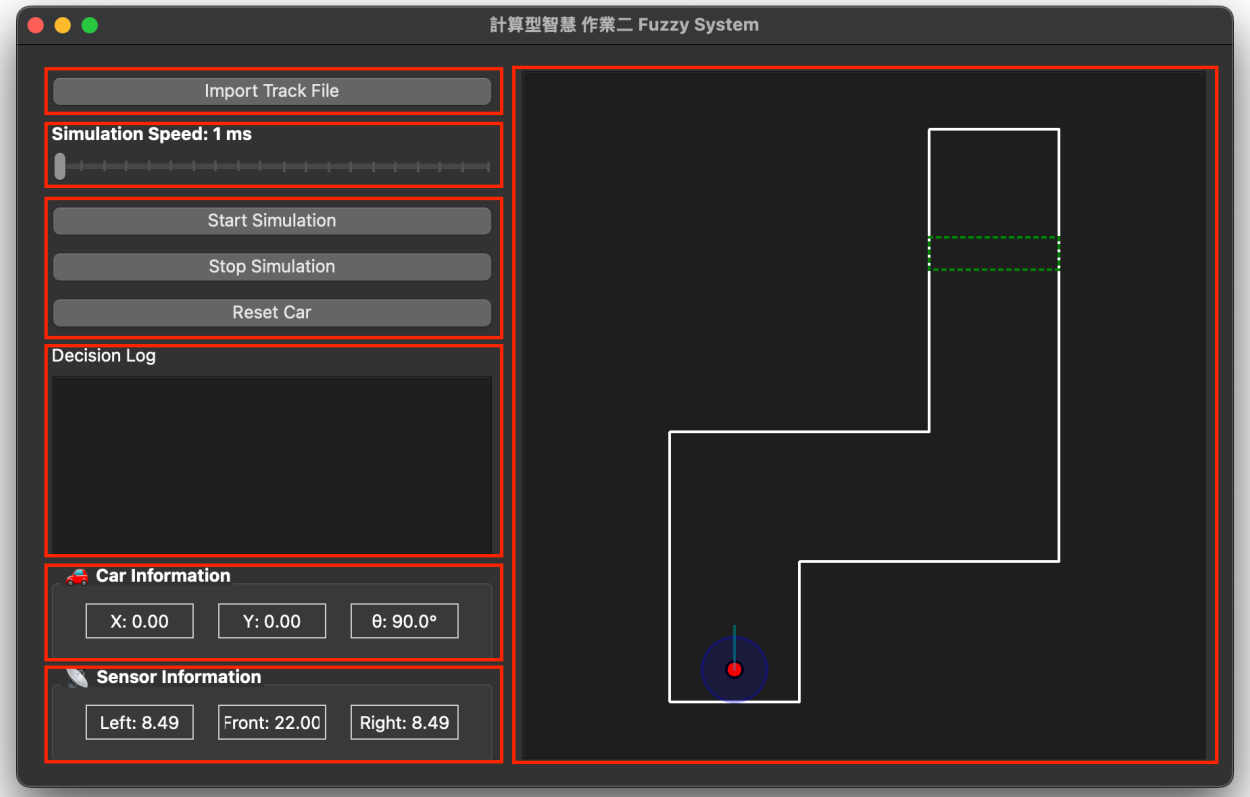
本次作業以 Python 的 PyQT5 開發互動式的 GUI 介面（圖一），整體介面分為左右兩區，操作方式為先匯入軌道座標檔，接著調整參數以及訓練速度，在按下「Start Training」。

左側功能區：

1. **匯入軌道**：點選「Import Track File」可讀入軌道.txt 檔案，並顯於右方顯示軌道
2. **訓練速度**：最低可設為 1ms 以達到快速訓練效果（加速動畫）。
3. **控制按鈕**：提供開始、暫停訓練（Stop Training）與重置車輛（Reset Car）。
4. **Decision Log**：記錄訓練過程中的決策狀態。
5. **車輛資訊**：即時顯示車輛的座標 (x, y) 及角度 θ。
6. **感測器資訊**：顯示三個方向（左前右）對邊界的偵測距離。

右側畫布區：

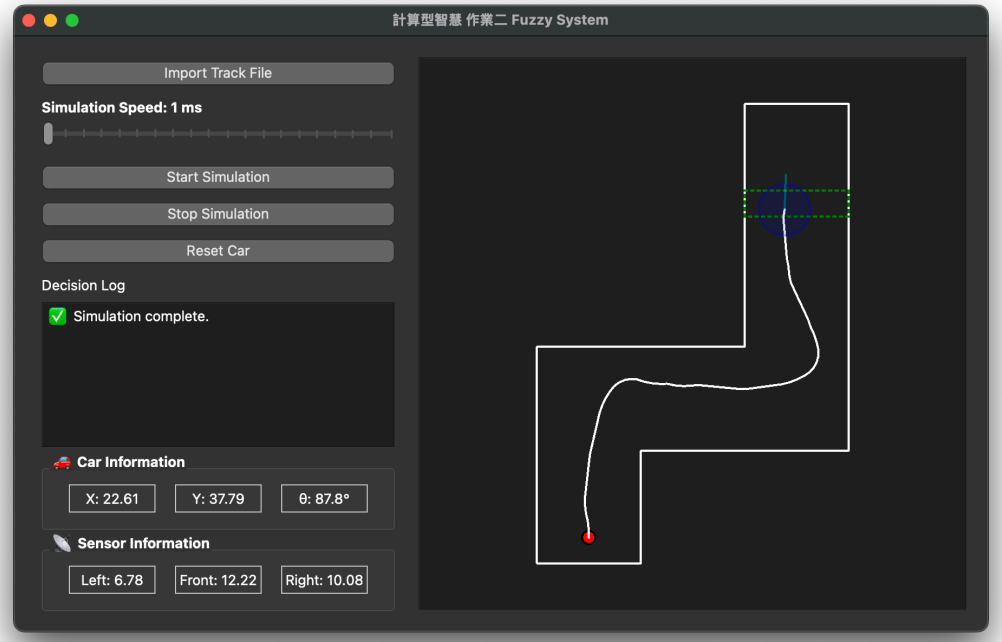
1. 顯示匯入的軌道圖形，包含白色邊界線、紅色起點、綠色終點區域。
2. 車輛以圓形表示，朝向以藍色線段表示。
3. 訓練過程中，車輛行經路徑會留下白色軌跡。



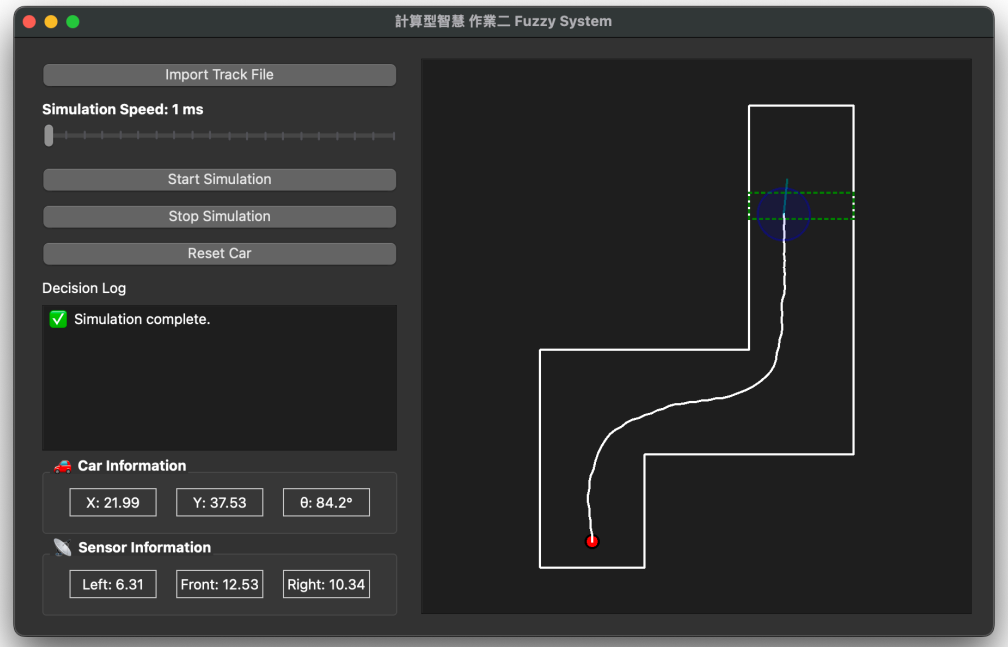
圖一、GUI 介面展示

二、實驗結果

經過大量實驗後，最終發現比較容易達到終點的設定列於本說明文件第三段，基於這個設定，測試十次的失敗次數頂多一次。在這個段落，我會描述我如何找到這個參數的，下方將一開始使用的參數以及最終使用參數做實驗對照，將修改前後的移動軌跡圖放於下方，圖二為修改前（small, medium, large 的閾值比較小），圖三為修改後（閾值比較大，也就是在距離比較大的情況，就收到警訊，提前轉彎），由下圖可以看出修改後的有提早轉彎的趨勢，不會出現像圖二有點“驚險”的情況：



圖二、移動軌跡截圖（修改前）

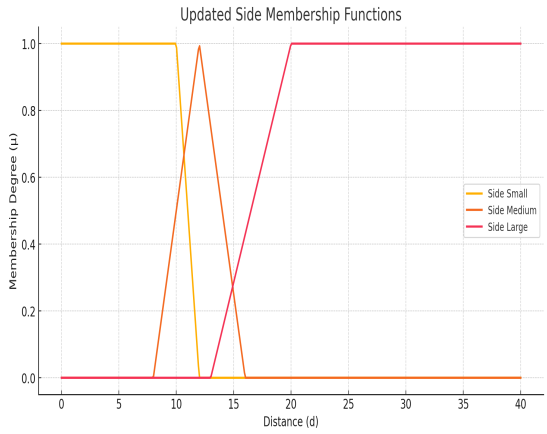
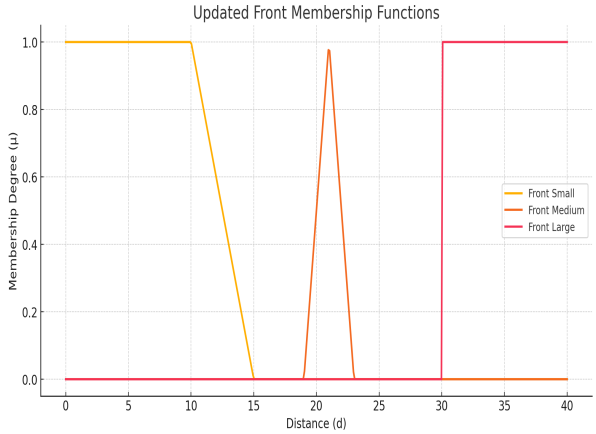


圖三、移動軌跡截圖（修改後）

三、模糊系統設計（歸屬函數）

模糊系統分為以下幾個部分：

1. **模糊化機構（歸屬函數）**：歸屬函數的設計主要分為兩個部分： side（左右傳感器）和 front（前方傳感器），這兩個部分又各自包含三個部分：*small*, *medium*, *large*，然後選擇歸屬度最大的作為該方向傳感器的程度（s, m, l）。

	side	***front***
small	$\begin{cases} \frac{12-d}{2}, & \text{if } 10 \le d < 12 \\ 0, & \text{if } d \ge 12 \end{cases}$	$\begin{cases} 1, & \text{if } d < 10 \\ \frac{15-d}{5}, & \text{if } 10 \le d < 15 \\ 0, & \text{if } d \ge 15 \end{cases}$
medium	$\begin{cases} \frac{d-8}{4}, & \text{if } 8 < d \le 12 \\ \frac{16-d}{4}, & \text{if } 12 < d \le 16 \\ 0, & \text{otherwise} \end{cases}$	$\begin{cases} \frac{d-19}{2}, & \text{if } 19 < d \le 21 \\ \frac{23-d}{2}, & \text{if } 21 < d \le 23 \\ 0, & \text{otherwise} \end{cases}$
large	$\begin{cases} \frac{d-13}{7}, & \text{if } 13 < d \le 20 \\ 1, & \text{if } d > 20 \\ 0, & \text{if } d \le 13 \end{cases}$	$\begin{cases} 1, & \text{if } d > 30 \\ 0, & \text{if } d \le 30 \end{cases}$
座標圖		

程式碼	```class Fuzzifier:
	    @staticmethod
	    def to_level(s, m, l):
	        return Level([s, m, l].index(max([s, m, l])))
	
	    @staticmethod
	    def l_point(distance):
	        s = MembershipFunctions.side_small(distance)
	        m = MembershipFunctions.side_medium(distance)
	        l = MembershipFunctions.side_large(distance)```	

```
        return Fuzzifier.to_level(s, m, l)

    @staticmethod
    def r_point(distance):
        s = MembershipFunctions.side_small(distance)
        m = MembershipFunctions.side_medium(distance)
        l = MembershipFunctions.side_large(distance)
        return Fuzzifier.to_level(s, m, l)

    @staticmethod
    def c_point(distance):
        s = MembershipFunctions.front_small(distance)
        m = MembershipFunctions.front_medium(distance)
        l = MembershipFunctions.front_large(distance)
        return Fuzzifier.to_level(s, m, l)
```

2. **模糊推論引擎與去模糊化機構**：模糊推論引擎包含 FuzzyController 以及 Rules 的部分，在這邊模糊規則算是跟去模糊話直接結合在一起，所以兩者皆在這個部分呈現。

FuzzyController

```
class FuzzyController:
    def __init__(self):
        self.fuzzifier = Fuzzifier()

    def decide_action(self, sensor_data):
        right, front, left = sensor_data
        l_point = self.fuzzifier.l_point(left)
        c_point = self.fuzzifier.c_point(front)
        r_point = self.fuzzifier.r_point(right)
        return Rules.apply(l_point, c_point,
r_point)
```

Rules

```
class Rules:
    @staticmethod
    def apply(l_point, c_point, r_point):
        if r_point == Level.SMALL:
            return -40
        if l_point == Level.SMALL:
            return 40
        if r_point == Level.MEDIUM and c_point
== Level.SMALL:
            return -20
        if l_point == Level.MEDIUM and c_point
== Level.SMALL:
            return 20
        return 0
```

四、分析與探討

針對這次作業，我主要有以下幾點觀察與思考：

1. **不同歸屬函數對車子行走軌跡的影響：**在實驗的部分，針對歸屬函數使用許多不同參數進行測試，起初車子仍不受控制去撞牆，慢慢經過調整後，車子開始能達到終點，然後過程十分“驚險”，如：已經快要碰撞時才轉彎等等，所以後續又把歸屬函數的數值調高，代表在還有一段距離時，就先轉彎，也正是最終版本，可以看出它可以“安穩”的抵達終點。

2. **與強化式學習的差異：**相比第一次作業使用 Q-learning，需要花上大量時間調整參數，在進行數百次甚至數千次 episode 的訓練，耗費大量時間，這次作業使用模糊系統，只需要設計好規則後，不用進行任何訓練，讓他實際模擬一次就可以知道結果，節省不少時間，讓我十分快樂（。然而，經過思考，我也了解這個作業設計的情境並不是非常複雜，所以使用模糊系統可以更快速的應用，而強化式學習則可以運用在更複雜的環境之中。以下我也透過表格統整出模糊系統與強化式學習的差異。

	強化式學習	**模糊系統**
訓練 / 計算成本	訓練時間成本高，計算成本也高。	不用訓練，計算成本低。
解釋性 / 邏輯	難以觀察策略的演變過程以及不可解釋性。	規則是專家人類自己訂定的，可解釋性高。
環境	可以適應複雜多變的環境。	適合簡單的環境，可以用語言描述的場景。