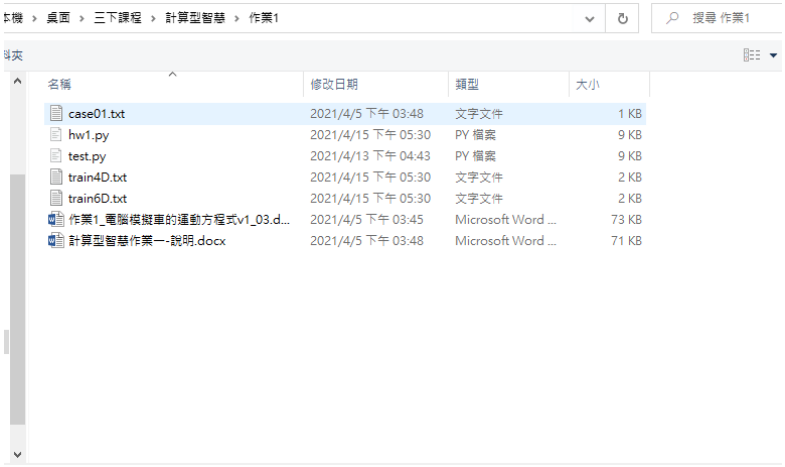
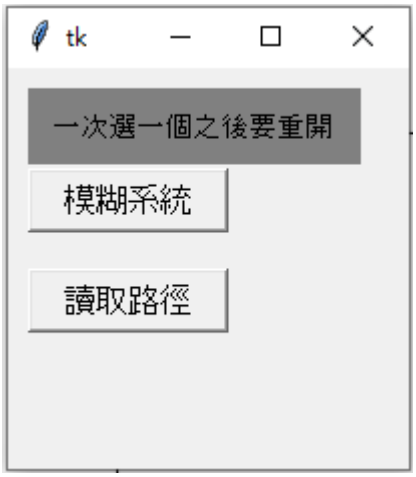
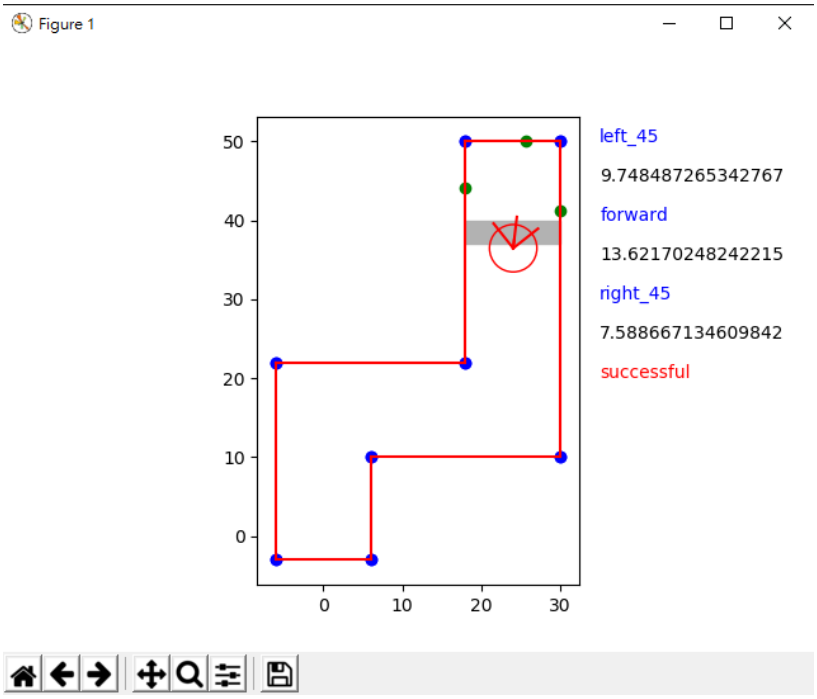
一、程式介面說明：

一、一開始會跳出檔案選擇介面，選擇你要的自走車文件。



二、跳出選擇介面，可以選擇要用模糊規則去跑自走車或是選擇路徑檔案按檔案路徑行走，兩個按鈕選一個按，之後必須重開才能再按。





GUI 介面會有自走車的動態移動顯示，右方顯示左 45 度、右 45 度、正前方的 sensor 分別測到的距離，如果能成功走到軌道終點，右下角會有紅字顯現 successful，如果失敗則會顯示 unsuccessful，然後車子撞到牆或走到終點都會停下。

二、程式碼說明

```
def distance(origin,head):
    global lists
    lenhead = 100
    x1 = origin[0]
    y1 = origin[1]
    x2 = head[0]
    y2 = head[1]
    if(x1 == x2):
        k1 = None
        b1=0
    else:
        k1 = (y2-y1)*1.0/(x2-x1)
        b1=y1*1.0-x1*k1*1.0
    X=0
    Y=0
    minx=0
    miny=0
    for i in range(3,len(lists)-1):
        flag = 0
        x3 = float(lists[i][0])
        x4 = float(lists[i+1][0])
        y3 = float(lists[i][1])
        y4 = float(lists[i+1][1])
        if(x3 == x4):
            flag = 1
        if(x4-x3) == 0:
            k2 = None
            b2 = 0
        else:
            k2=(y4-y3)*1.0/(x4-x3)
            b2=y3*1.0-x3*k2*1.0
        if(k1!=k2):
            if k2==None:
```

```
                X=x3
                Y=k1*X*1.0+b1*1.0
            elif k1==None:
                X=x1
                Y=k2*X*1.0+b2*1.0
            else:
                X=(b2-b1)*1.0/(k1-k2)
                Y=k1*X*1.0+b1*1.0
            if(flag == 0):
                if(X<=max(x3,x4) and X>=min(x3,x4)):
                    if (head[0] - X)**2+(head[1]-
Y)**2 < (origin[0] - X)**2+(origin[1]-Y)**2:
                        if(((origin[0] - X)**2+(origin[1]-
Y)**2)**0.5 < lenhead):
                            minx = X
                            miny = Y
                            lenhead = min(lenhead,((origin[0] - X)**2+(
origin[1]-Y)**2)**0.5)
            if (flag == 1):
                if Y>=min(y3,y4) and Y<=max(y3,y4):
                    if (head[0] - X)**2+(head[1]-
Y)**2 < (origin[0] - X)**2+(origin[1]-Y)**2:
                        if(((origin[0] - X)**2+(origin[1]-
Y)**2)**0.5 < lenhead):
                            minx = X
                            miny = Y
                            lenhead = min(lenhead,((origin[0] - X)**2+(
origin[1]-Y)**2)**0.5)

    plt.scatter(minx,miny,color = "g")
    return lenhead
```

這是我算出三個 sensor 與牆面距離的函式，依序 for 迴圈跑完每個牆壁，一開始先求牆壁與 sensor 公式再算出兩條直線的交點，如果交點超過牆壁的界線則不算，最後的距離是與所有牆面的最小距離，但這還有個問題是假設交點與前進方向是反方向呢？我的解決方法是如果牆面與 origin(原點)距離<=牆面與 head(方向點)的距離，才是合法距離。

```
if(len_of_line[2] >= 13 or len_of_line[0] <= 6):
        change_angle = 10
```

```
    elif(len_of_line[0]>= 13 or len_of_line[2] <= 6):
        change_angle = -10
    elif(len_of_line[2] >= 10 or len_of_line[0] <= 8):
        change_angle = 5
    elif(len_of_line[0]>= 10 or len_of_line[2] <= 8):
        change_angle = -5
```

算出距離後再套用模糊規則找出所需的改變角度

三、模糊規則設計

```
if((len_of_line[2] > 6 and len_of_line[2] < 13) and len_of_line[0] <= 6
):
            change_angle = 5
        elif((len_of_line[0] > 6 and len_of_line[0] < 13)and len_of_lin
e[2] <= 6):
            change_angle = -5
        elif(len_of_line[2] >= 13 or len_of_line[0] <= 6):
            change_angle = 10
        elif(len_of_line[0]>= 13 or len_of_line[2] <= 6):
            change_angle = -10
```

在這個軌道中沒有很窄的窄口所以兩側距離的最小值一定比前方距離小，所以我前方距離不考慮。

一、如果左方是 mid 且右方小：左轉 5 度

二、如果右方是 mid 且左方小：右轉 5 度

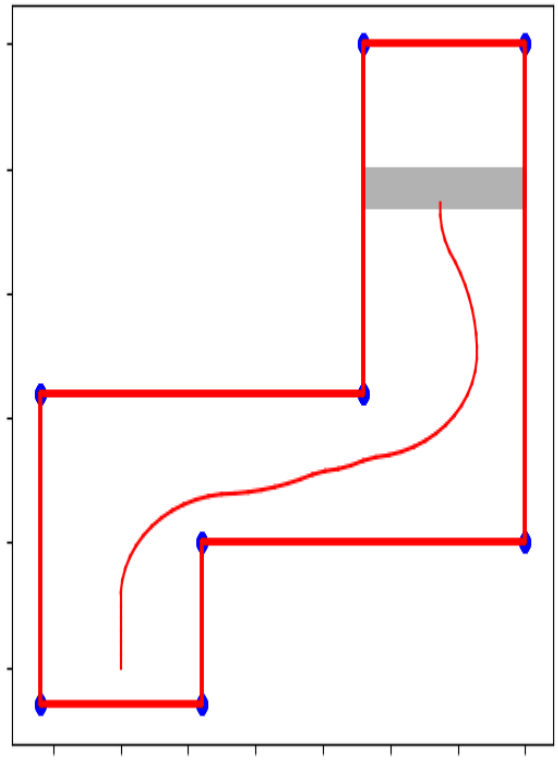
三、如果左方大或右方小：左轉 10 度

四、如果左方小或右方大：右轉 10 度

一開始有試過 40,30,20 發現角度都太大，所以後來調成 10 與 5。

四、實驗結果

再經過角度調整後已經可以用比較平滑得路徑到達終點，如果角度太小會撞牆，角度太大路徑會呈現鋸齒狀不好看。



五、分析

如果路徑更刁鑽時模糊規則必須更複雜才行，我的模糊規則只適用於路較大、角度小的路徑，如果一種距離有三種判斷 small、medium、large 則最多要有 27 種表示方式，模糊規則數量與路徑複雜度成正比。