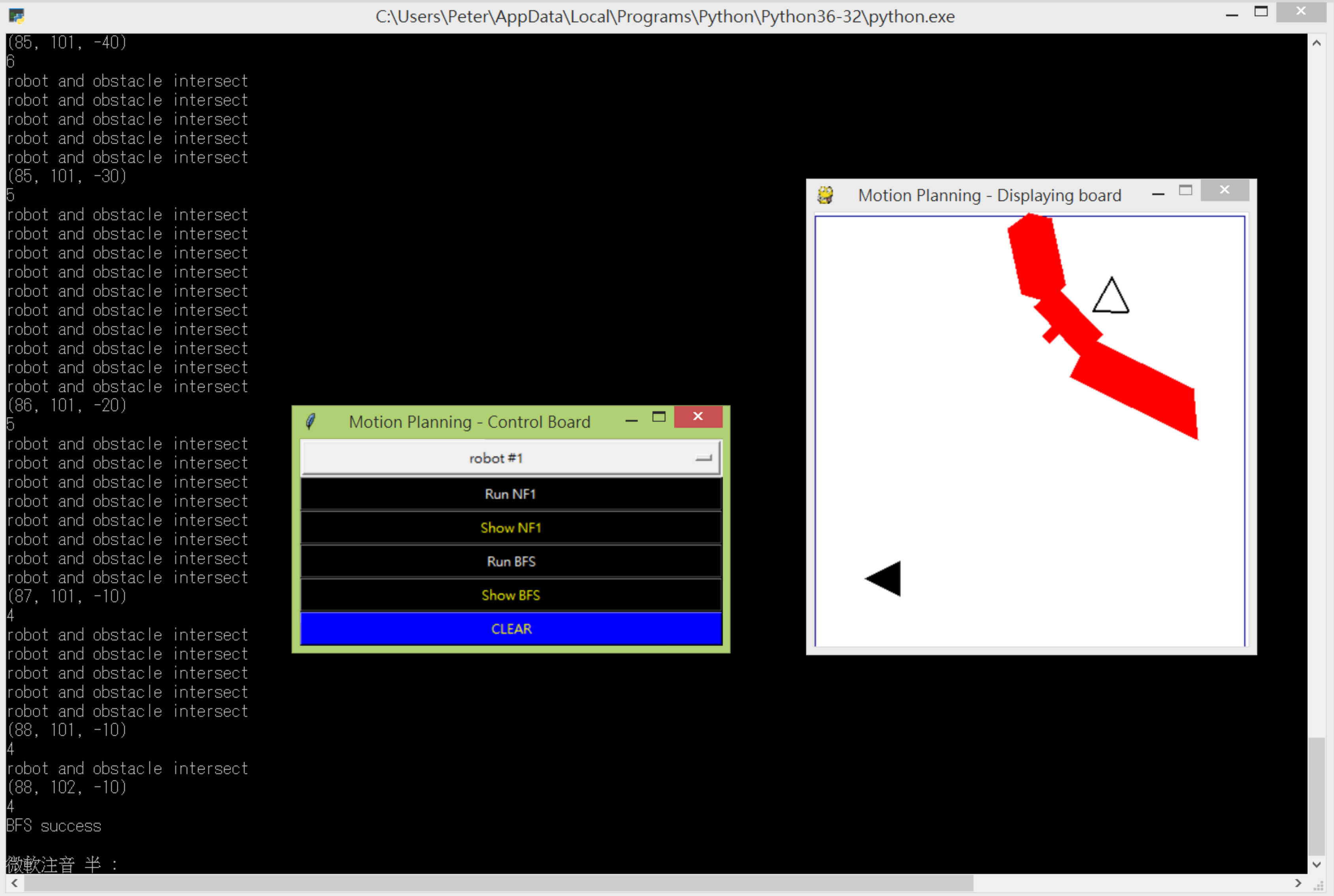
GRA – motion planning

GUI



C:\Users\Peter\AppData\Local\Programs\Python\Python36-32\python.exe

```
(85, 101, -40)
6
robot and obstacle intersect
robot and obstacle intersect
robot and obstacle intersect
robot and obstacle intersect
robot and obstacle intersect
(85, 101, -30)
5
robot and obstacle intersect
robot and obstacle intersect
robot and obstacle intersect
robot and obstacle intersect
robot and obstacle intersect
robot and obstacle intersect
robot and obstacle intersect
robot and obstacle intersect
robot and obstacle intersect
robot and obstacle intersect
(86, 101, -20)
5
robot and obstacle intersect
robot and obstacle intersect
robot and obstacle intersect
robot and obstacle intersect
robot and obstacle intersect
robot and obstacle intersect
robot and obstacle intersect
robot and obstacle intersect
(87, 101, -10)
4
robot and obstacle intersect
robot and obstacle intersect
robot and obstacle intersect
robot and obstacle intersect
robot and obstacle intersect
(88, 101, -10)
4
robot and obstacle intersect
(88, 102, -10)
4
BFS success

微軟注音 半 ：
```

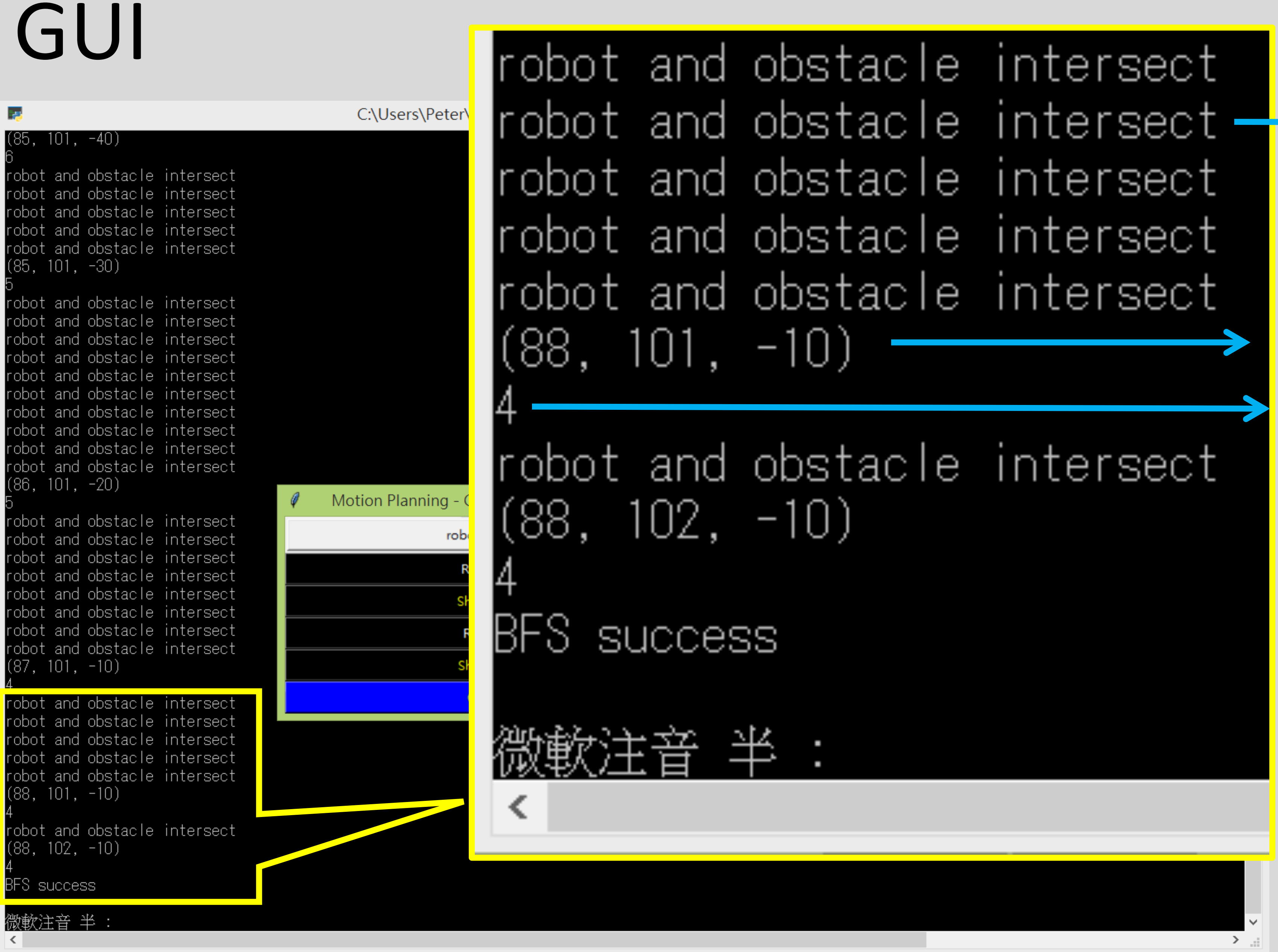
Motion Planning - Control Board
robot #1
Run NF1
Show NF1
Run BFS
Show BFS
CLEAR
Motion Planning - Displaying board

空心多邊形：robot_goal

實心多邊形：robot_recent

紅色多邊形：obstacle

GUI



```
(85, 101, -40)
6
robot and obstacle intersect
robot and obstacle intersect
robot and obstacle intersect
robot and obstacle intersect
robot and obstacle intersect
(85, 101, -30)
5
robot and obstacle intersect
robot and obstacle intersect
robot and obstacle intersect
robot and obstacle intersect
robot and obstacle intersect
robot and obstacle intersect
robot and obstacle intersect
robot and obstacle intersect
robot and obstacle intersect
robot and obstacle intersect
(86, 101, -20)
5
robot and obstacle intersect
robot and obstacle intersect
robot and obstacle intersect
robot and obstacle intersect
robot and obstacle intersect
robot and obstacle intersect
robot and obstacle intersect
robot and obstacle intersect
(87, 101, -10)
4
robot and obstacle intersect
robot and obstacle intersect
robot and obstacle intersect
robot and obstacle intersect
robot and obstacle intersect
(88, 101, -10)
4
robot and obstacle intersect
(88, 102, -10)
4
BFS success

微軟注音 半 :
```

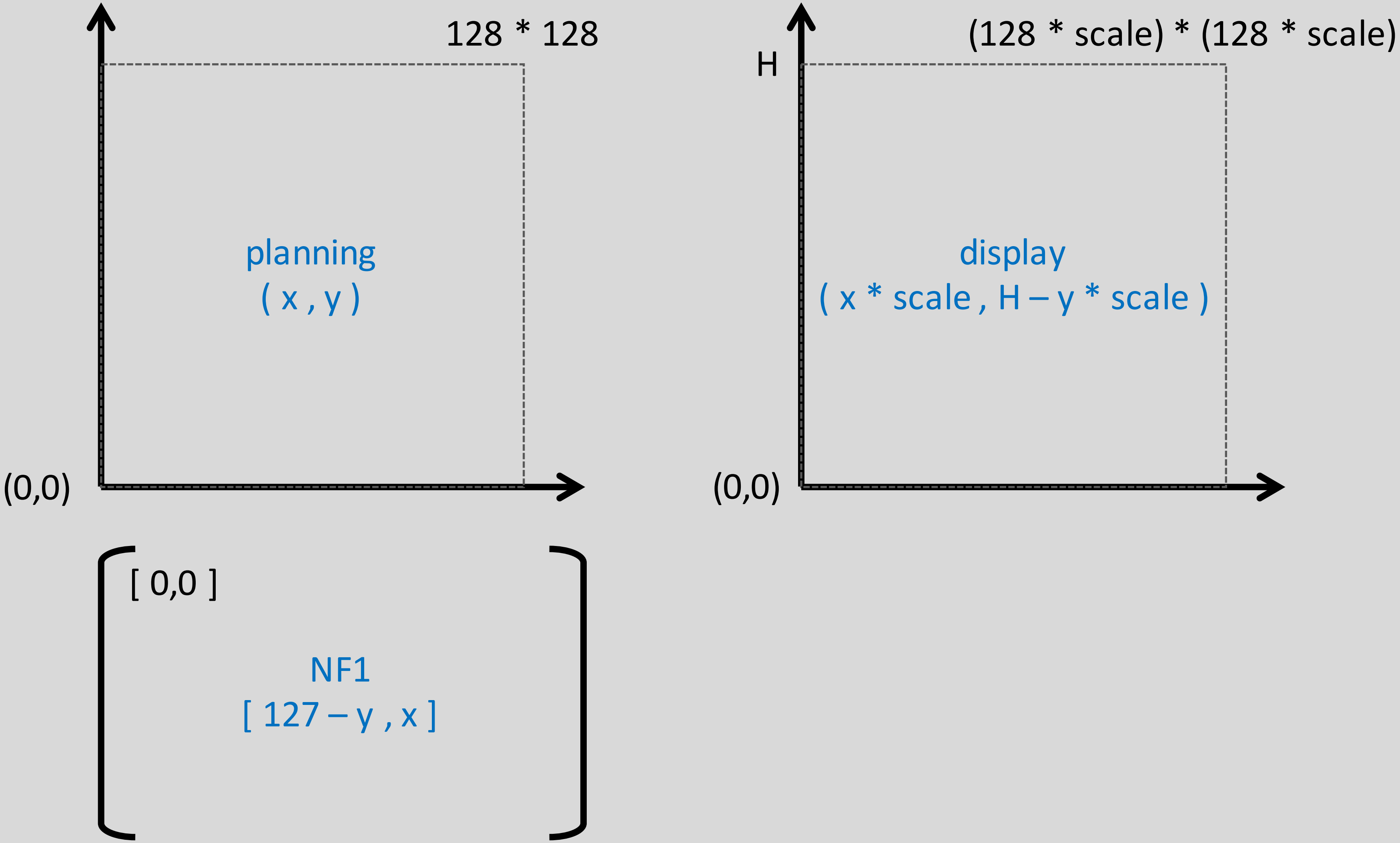
```
robot and obstacle intersect
robot and obstacle intersect
robot and obstacle intersect
robot and obstacle intersect
robot and obstacle intersect
(88, 101, -10)
4
robot and obstacle intersect
(88, 102, -10)
4
BFS success

微軟注音 半 :
```

with the neighbor configuration of FIRST in BFS OPEN, robot intersects with obstacles

configuration of FIRST in BFS OPEN

potential value of FIRST in BFS OPEN

兩張 map



128 * 128

(0,0)

planning

(x , y)

[0,0]

NF1

[127 – y , x]

(128 * scale) * (128 * scale)

H

(0,0)

display

(x * scale , H – y * scale)

GRA_motion_planning.py Outline

- import package
- Background initialization
- Define function
- Define object – BFS_OPEN
  Define object – BFS_T
- Define function – NF1
- Define function – BFS
- Define object – robots
  Define object – obstacles
- Input row data
- tkinter initialization
- pygame initialization
- main()

Package we need :

import numpy

import math

import tkinter

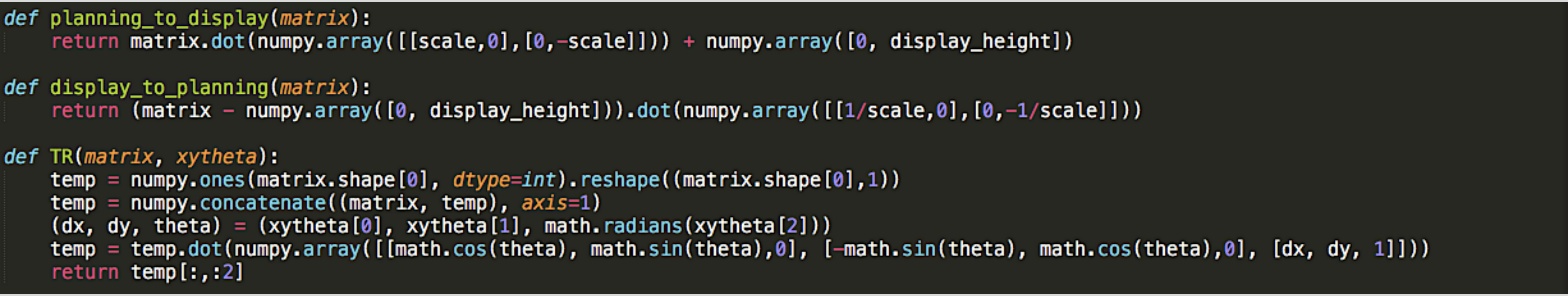
import pygame

import os

import platform

import matplotlib.pyplot

Function



```
def planning_to_display(matrix):
    return matrix.dot(numpy.array([[scale,0],[0,-scale]])) + numpy.array([0, display_height])

def display_to_planning(matrix):
    return (matrix - numpy.array([0, display_height])).dot(numpy.array([[1/scale,0],[0,-1/scale]]))

def TR(matrix, xytheta):
    temp = numpy.ones(matrix.shape[0], dtype=int).reshape((matrix.shape[0],1))
    temp = numpy.concatenate((matrix, temp), axis=1)
    (dx, dy, theta) = (xytheta[0], xytheta[1], math.radians(xytheta[2]))
    temp = temp.dot(numpy.array([[math.cos(theta), math.sin(theta),0], [-math.sin(theta), math.cos(theta),0], [dx, dy, 1]]))
    return temp[:,:2]
```

planning_to_display (matrix)

planning map 的 (x , y) 轉成 display map 的 (x’ , y’)

< Parameters >

matrix : numpy.ndarray (2D)

< Return > numpy.ndarray (2D)

display_to_planning (matrix)

display map 的 (x’ , y’) 轉成 planning map 的 (x , y)

< Parameters >

matrix : numpy.ndarray (2D)

< Return > numpy.ndarray (2D)

TR (matrix , xytheta)

對 vertices 做平移與旋轉

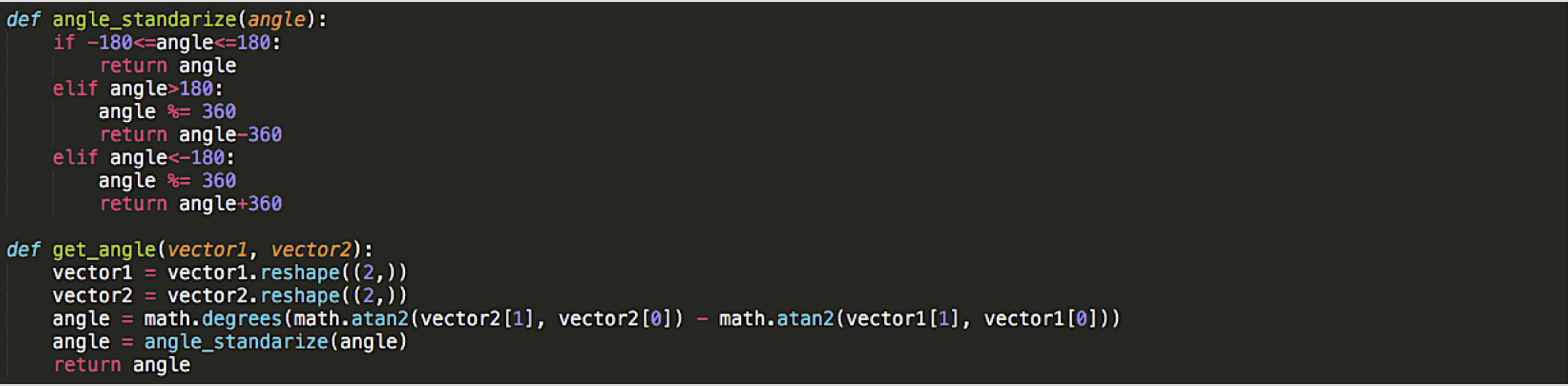
< Parameters >

matrix : numpy.ndarray (2D)

xytheta : list / numpy.ndarray (1D)

< Return > numpy.ndarray (2D)

Function



```python
def angle_standarize(angle):
    if -180<=angle<=180:
        return angle
    elif angle>180:
        angle %= 360
        return angle-360
    elif angle<-180:
        angle %= 360
        return angle+360

def get_angle(vector1, vector2):
    vector1 = vector1.reshape((2,))
    vector2 = vector2.reshape((2,))
    angle = math.degrees(math.atan2(vector2[1], vector2[0]) - math.atan2(vector1[1], vector1[0]))
    angle = angle_standarize(angle)
    return angle
```

angle_standarize (angle)

鎖定所有 configuration 的 theta 維持在 [-180 , 180] 之間

< Parameters >

angle : float

< Return > float

get_angle (vector1 , vector2)

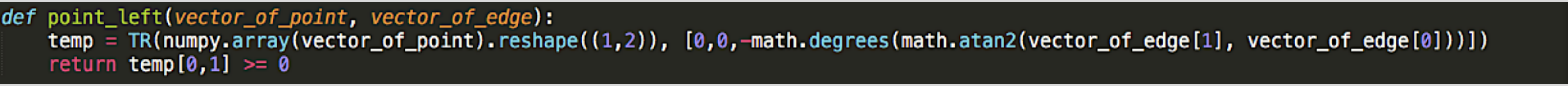
計算兩個向量之間的夾角

< Parameters >

vector1 : numpy.ndarray (1D/2D)

< Return > float (degrees)

Function



```
def point_left(vector_of_point, vector_of_edge):
    temp = TR(numpy.array(vector_of_point).reshape((1,2)), [0,0,-math.degrees(math.atan2(vector_of_edge[1], vector_of_edge[0]))])
    return temp[0,1] >= 0
```

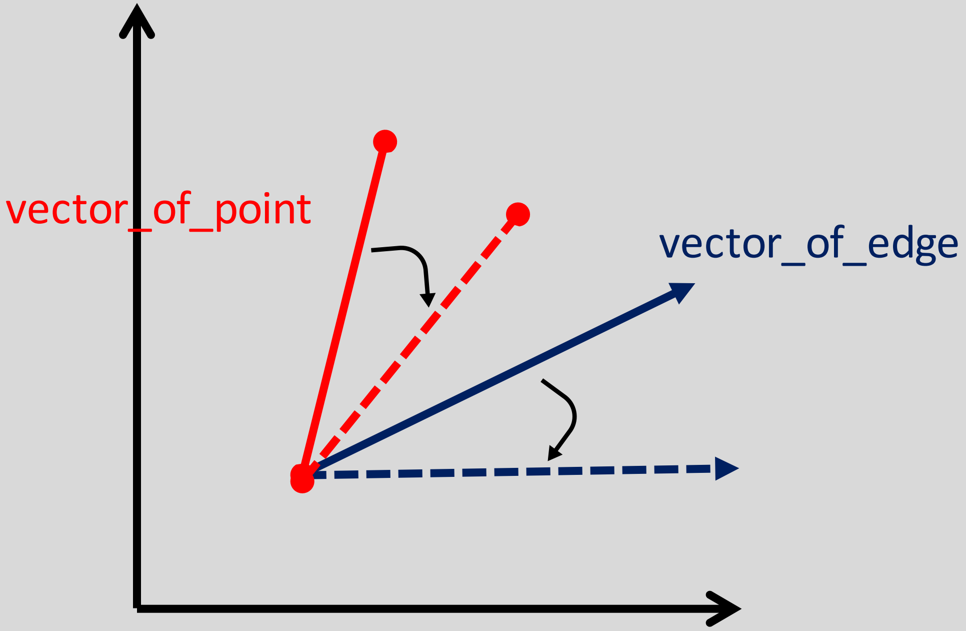
point_left (vector_of_point , vector_of_edge)

Point 是否在 edge 的左手邊 (根據edge的方向判斷)；
point 與 edge 均為向量，延 edge 的角度旋轉，若 point
的 y > 0 代表在 edge 的左手邊

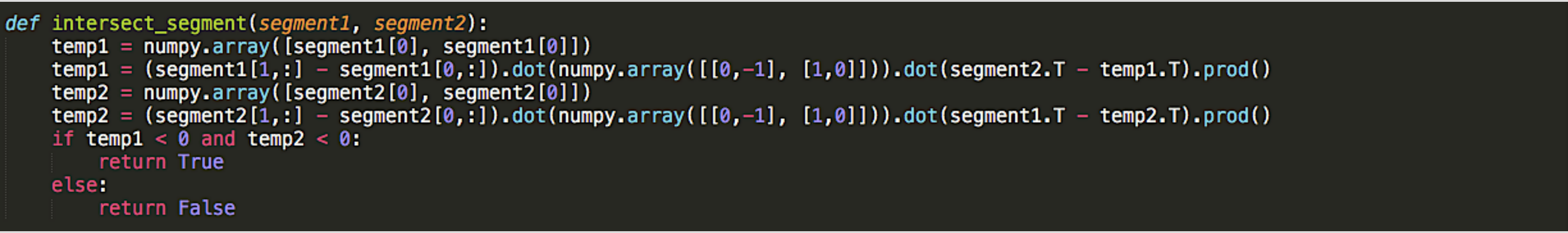
< Parameters >

vector_of_point : list / numpy.ndarray (1D)
Vector_of_edge : list / numpy.ndarray (1D)

< Return > boolean



Function



```
def intersect_segment(segment1, segment2):
    temp1 = numpy.array([segment1[0], segment1[0]])
    temp1 = (segment1[1,:] - segment1[0,:]).dot(numpy.array([[0,-1], [1,0]])).dot(segment2.T - temp1.T).prod()
    temp2 = numpy.array([segment2[0], segment2[0]])
    temp2 = (segment2[1,:] - segment2[0,:]).dot(numpy.array([[0,-1], [1,0]])).dot(segment1.T - temp2.T).prod()
    if temp1 < 0 and temp2 < 0:
        return True
    else:
        return False
```

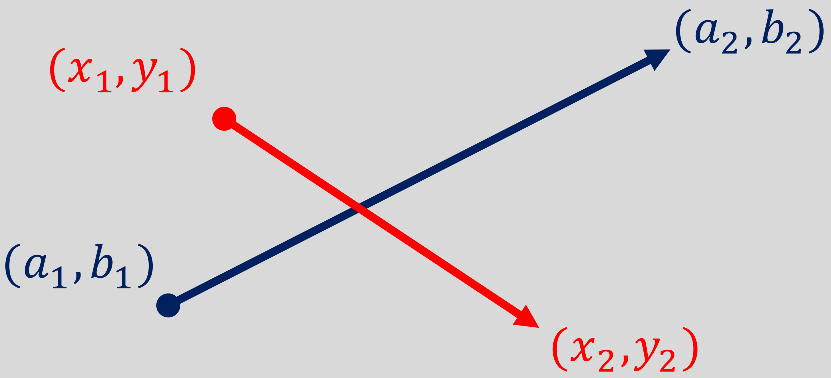
intersect_segment (segment1 , segment2)

兩個線段是否相交

< Parameters >

segment1 : numpy.ndarray (2 vertices in 2D)

< Return > boolean



$$V_{12} = [x_2 - x_1 \quad y_2 - y_1]$$

$$V'_{12} = [y_2 - y_1 \quad x_2 - x_1]$$

$$= ([x_2 \;\; y_2] - [x_1 \;\; y_1]) \times \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix}$$

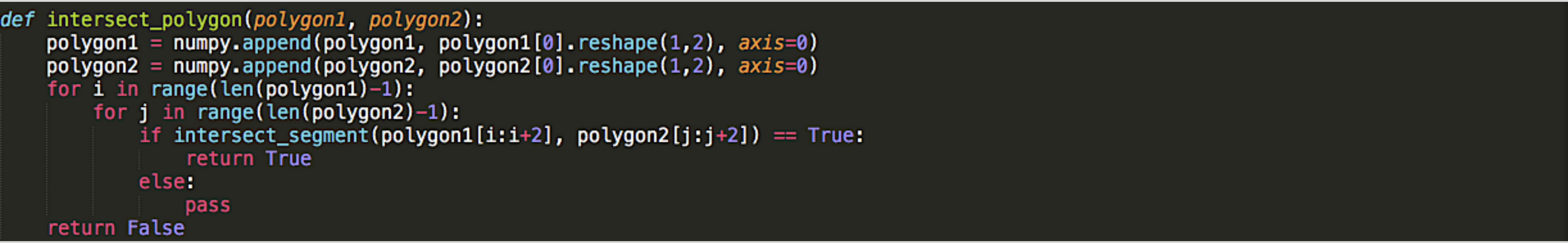
$$[V'_{12} \cdot V_{13} \quad V'_{12} \cdot V_{14}]$$

$$= ([x_2 \;\; y_2] - [x_1 \;\; y_1]) \times \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix} \times \left(\begin{bmatrix} a_1 & b_1 \\ a_2 & b_2 \end{bmatrix}^T - \begin{bmatrix} x_1 & y_1 \\ x_1 & y_1 \end{bmatrix}^T \right)$$

$$[V'_{34} \cdot V_{31} \quad V'_{34} \cdot V_{32}]$$

$$= ([a_2 \;\; b_2] - [a_1 \;\; b_1]) \times \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix} \times \left(\begin{bmatrix} x_1 & y_1 \\ x_2 & y_2 \end{bmatrix}^T - \begin{bmatrix} a_1 & b_1 \\ a_1 & b_1 \end{bmatrix}^T \right)$$

Function



```python
def intersect_polygon(polygon1, polygon2):
    polygon1 = numpy.append(polygon1, polygon1[0].reshape(1,2), axis=0)
    polygon2 = numpy.append(polygon2, polygon2[0].reshape(1,2), axis=0)
    for i in range(len(polygon1)-1):
        for j in range(len(polygon2)-1):
            if intersect_segment(polygon1[i:i+2], polygon2[j:j+2]) == True:
                return True
            else:
                pass
    return False
```

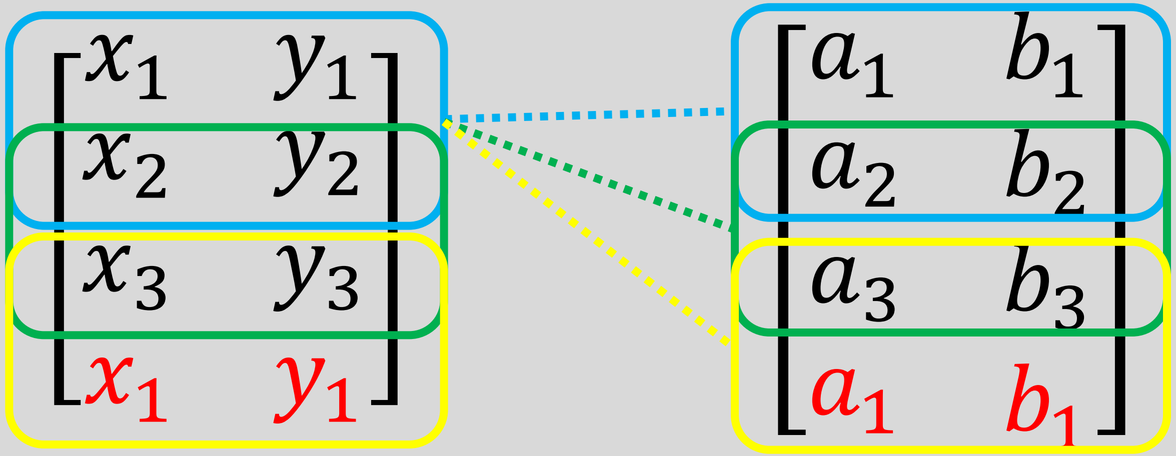
intersect_polygon (polygon1 , polygon2)

兩個多邊形是否相交，檢查所有兩兩的線段

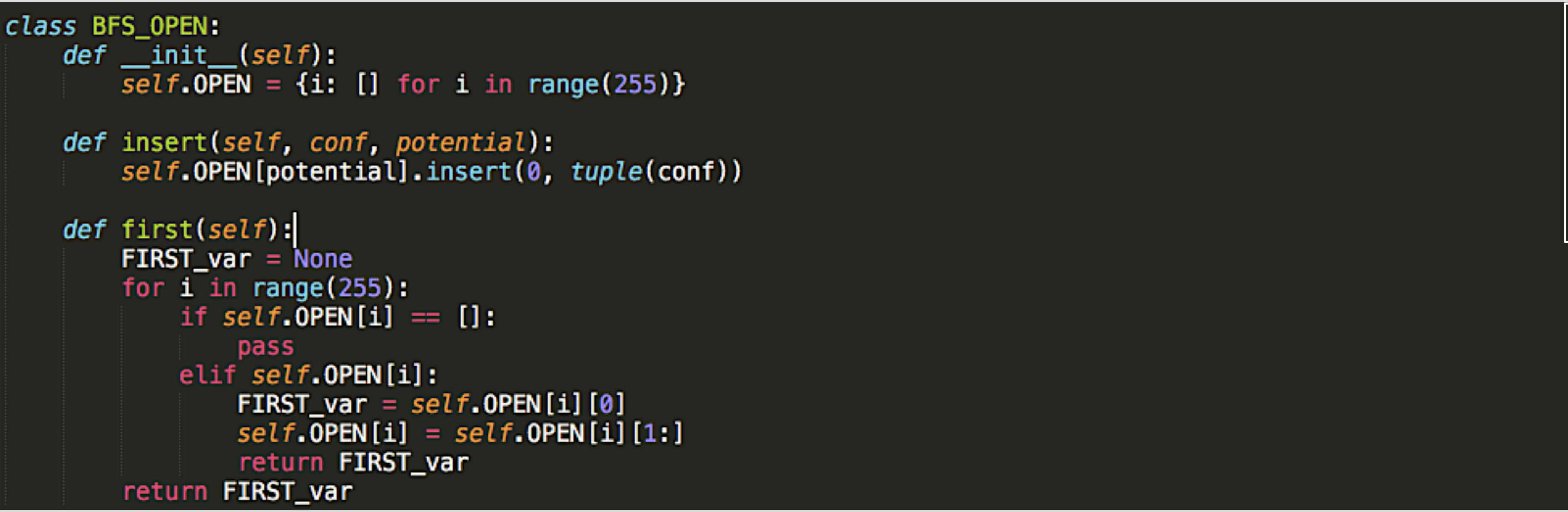
< Parameters >

polygon1 : numpy.ndarray (2D)

< Return > boolean



Object – BFS_OPEN



```
class BFS_OPEN:
    def __init__(self):
        self.OPEN = {i: [] for i in range(255)}

    def insert(self, conf, potential):
        self.OPEN[potential].insert(0, tuple(conf))

    def first(self):
        FIRST_var = None
        for i in range(255):
            if self.OPEN[i] == []:
                pass
            elif self.OPEN[i]:
                FIRST_var = self.OPEN[i][0]
                self.OPEN[i] = self.OPEN[i][1:]
                return FIRST_var
        return FIRST_var
```

BFS_OPEN.OPEN =
{ 0 : [ ] ,
 1 : [(conf) , (conf)] ,
 …… ,
 254 : [(conf) , (conf)] }

BFS_OPEN.insert (conf , potential)

未被拜訪過的 configuration 根據 potential (整數) 值加入 OPEN

< Parameters >

conf : tuple

potential : int

< Return > None

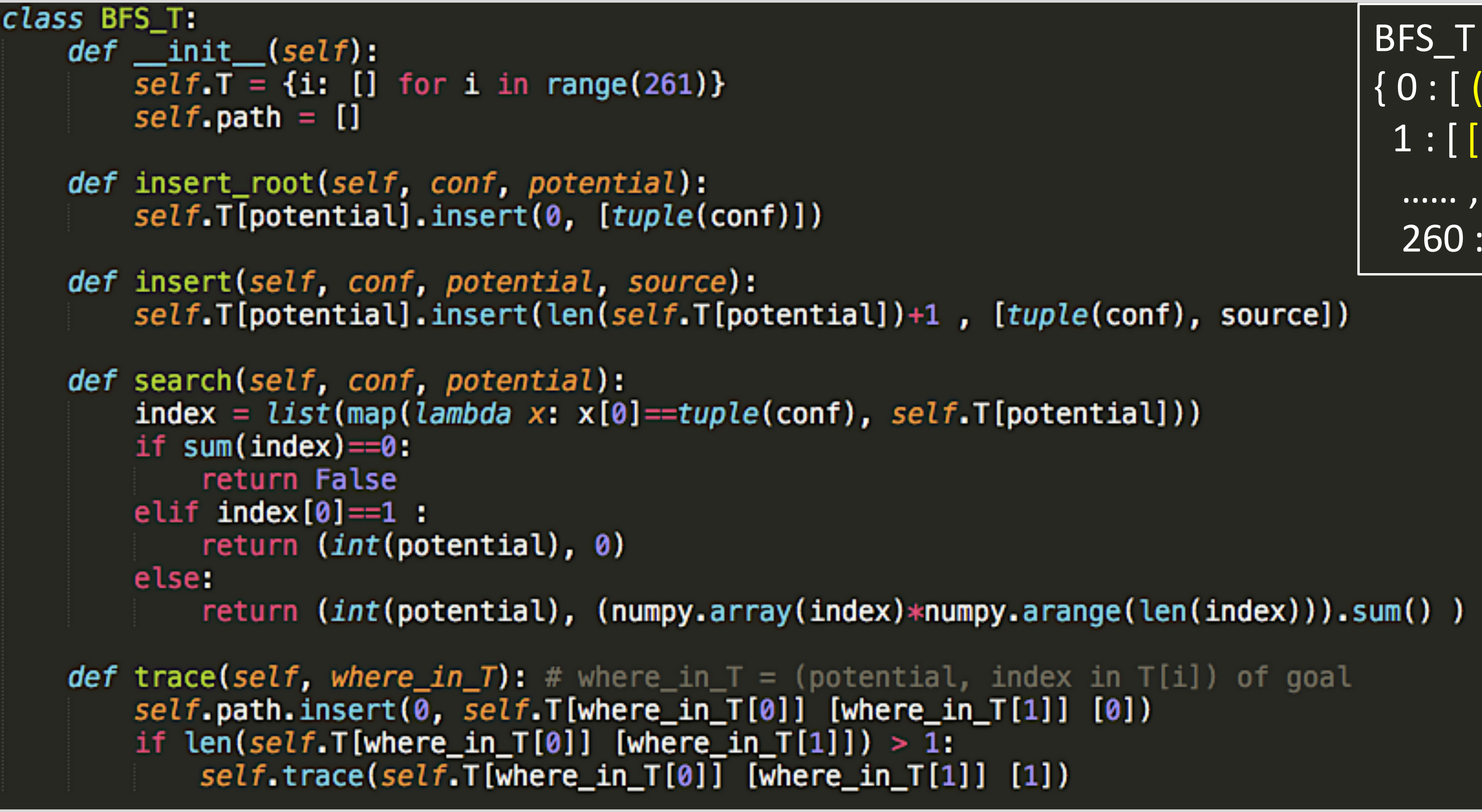
BFS_OPEN.first ()

若 OPEN 非空集合，取出 potential 最小的 conf；
若 OPEN 為空，回傳 None

< Parameters >

< Return > tuple / None

Object – BFS_T



```python
class BFS_T:
    def __init__(self):
        self.T = {i: [] for i in range(261)}
        self.path = []

    def insert_root(self, conf, potential):
        self.T[potential].insert(0, [tuple(conf)])

    def insert(self, conf, potential, source):
        self.T[potential].insert(len(self.T[potential])+1 , [tuple(conf), source])

    def search(self, conf, potential):
        index = list(map(lambda x: x[0]==tuple(conf), self.T[potential]))
        if sum(index)==0:
            return False
        elif index[0]==1 :
            return (int(potential), 0)
        else:
            return (int(potential), (numpy.array(index)*numpy.arange(len(index))).sum() )

    def trace(self, where_in_T): # where_in_T = (potential, index in T[i]) of goal
        self.path.insert(0, self.T[where_in_T[0]] [where_in_T[1]] [0])
        if len(self.T[where_in_T[0]] [where_in_T[1]]) > 1:
            self.trace(self.T[where_in_T[0]] [where_in_T[1]] [1])
```

BFS_T . T =
{ 0 : [(conf)] ,
 1 : [[(conf) , (source)] , [(conf) , (source)]]
 …… ,
 260 : [[(conf) , (source)] , [(conf) , (source)]] }

(source) = 來源在 T tree 所在的位置

Object – BFS_T

BFS_T.insert_root (conf , potential)

control point 的起點加入T tree

< Parameters >

conf : tuple

potential : int

< Return > None

BFS_T.search (conf , potential)

neighbor 是否存在 T tree 裡面，有的話回傳位置，沒有的話回傳 False；同時作為 visited or not 的判斷

< Parameters >

conf : tuple

potential : int

< Return > tuple / False

BFS_T.insert (conf , potential , source)

未被拜訪過的 neighbor 加入T tree

< Parameters >

conf : tuple

potential : int

source : tuple

< Return > None

BFS_T.trace (where_in_T)

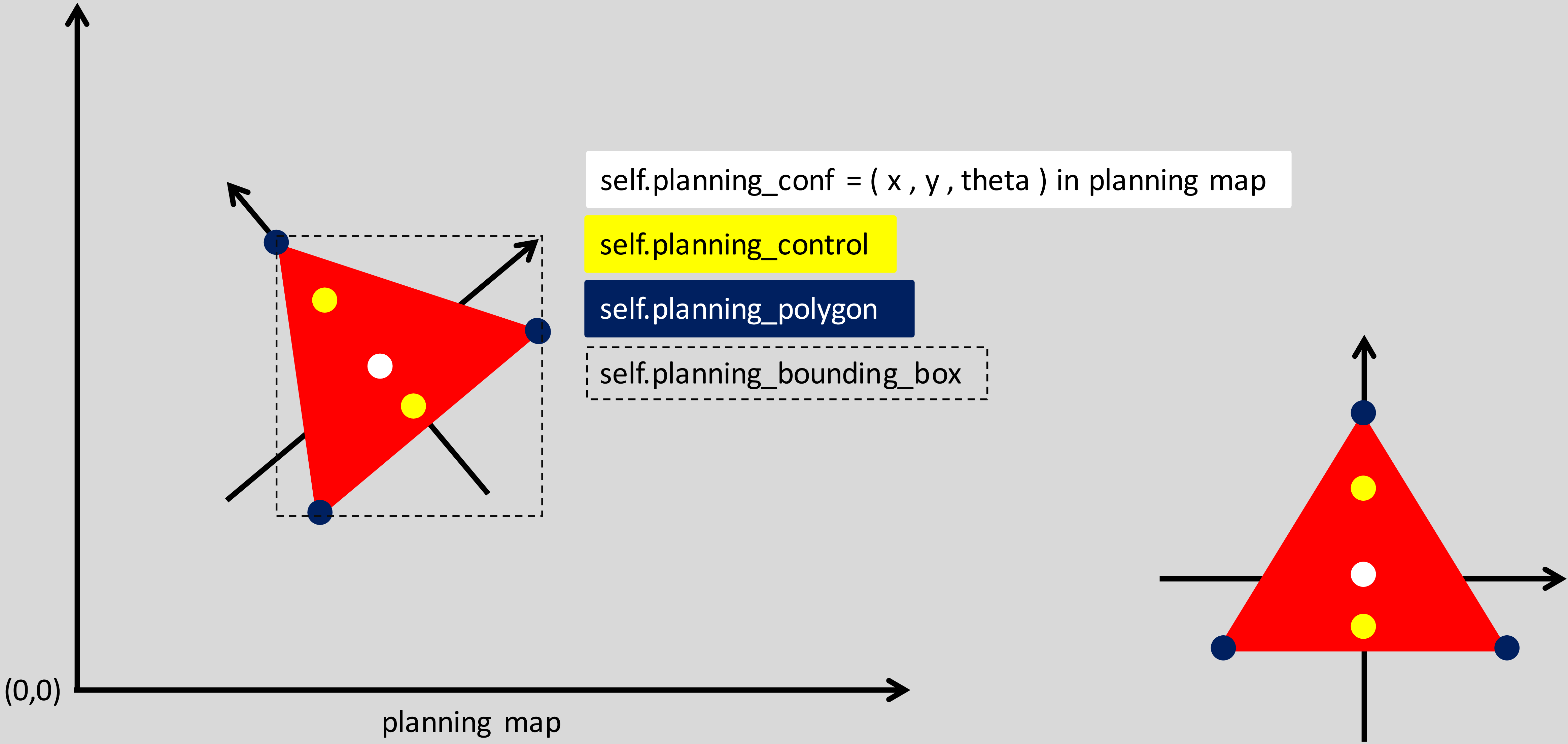
search 到 goal 的時候，根據所在的 T tree 位置，回傳 path 陣列

< Parameters >

where_in_T : tuple

< Return > list

Robots – Class



self.planning_conf = (x , y , theta) in planning map

self.planning_control

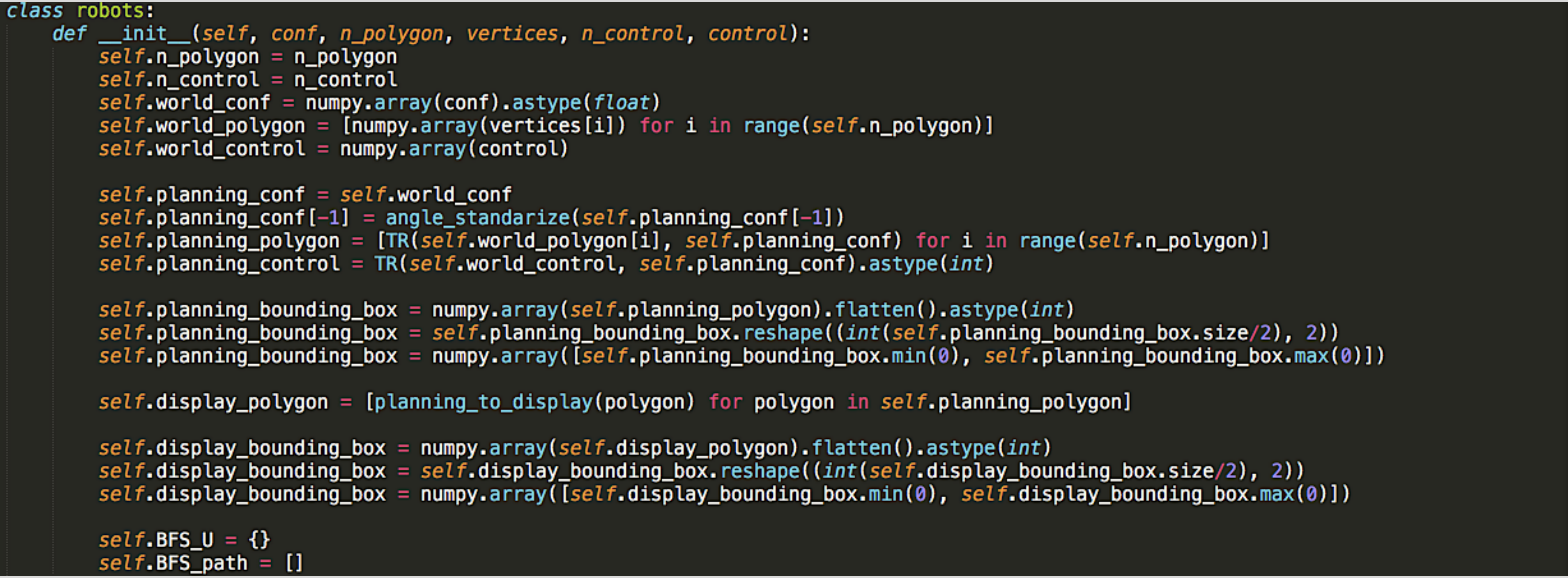
self.planning_polygon

self.planning_bounding_box

Robots – Class

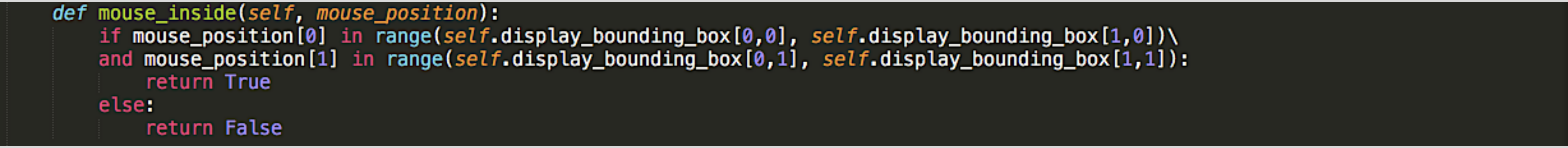
<Parameters>

- conf : tuple/list
- n_polygon : int
- vertices : list and counterclockwise ordered (3D)
- n_control : int
- control : list (2D)



```
class robots:
    def __init__(self, conf, n_polygon, vertices, n_control, control):
        self.n_polygon = n_polygon
        self.n_control = n_control
        self.world_conf = numpy.array(conf).astype(float)
        self.world_polygon = [numpy.array(vertices[i]) for i in range(self.n_polygon)]
        self.world_control = numpy.array(control)

        self.planning_conf = self.world_conf
        self.planning_conf[-1] = angle_standarize(self.planning_conf[-1])
        self.planning_polygon = [TR(self.world_polygon[i], self.planning_conf) for i in range(self.n_polygon)]
        self.planning_control = TR(self.world_control, self.planning_conf).astype(int)

        self.planning_bounding_box = numpy.array(self.planning_polygon).flatten().astype(int)
        self.planning_bounding_box = self.planning_bounding_box.reshape((int(self.planning_bounding_box.size/2), 2))
        self.planning_bounding_box = numpy.array([self.planning_bounding_box.min(0), self.planning_bounding_box.max(0)])

        self.display_polygon = [planning_to_display(polygon) for polygon in self.planning_polygon]

        self.display_bounding_box = numpy.array(self.display_polygon).flatten().astype(int)
        self.display_bounding_box = self.display_bounding_box.reshape((int(self.display_bounding_box.size/2), 2))
        self.display_bounding_box = numpy.array([self.display_bounding_box.min(0), self.display_bounding_box.max(0)])

        self.BFS_U = {}
        self.BFS_path = []
```

Robots – Class.function



```python
def mouse_inside(self, mouse_position):
    if mouse_position[0] in range(self.display_bounding_box[0,0], self.display_bounding_box[1,0])\
    and mouse_position[1] in range(self.display_bounding_box[0,1], self.display_bounding_box[1,1]):
        return True
    else:
        return False
```

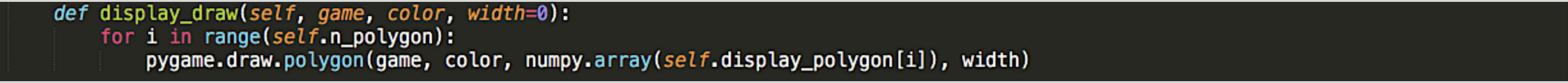
robots.mouse_inside (mouse_position)

游標是否落在 robot 的bounding box裡面；使用在物件拖移

< Parameters >

mouse_position : list (1D)

< Return > Boolean



```python
def display_draw(self, game, color, width=0):
    for i in range(self.n_polygon):
        pygame.draw.polygon(game, color, numpy.array(self.display_polygon[i]), width)
```

robots.displaty_draw (game , color , width = 0)

物件畫在 display map

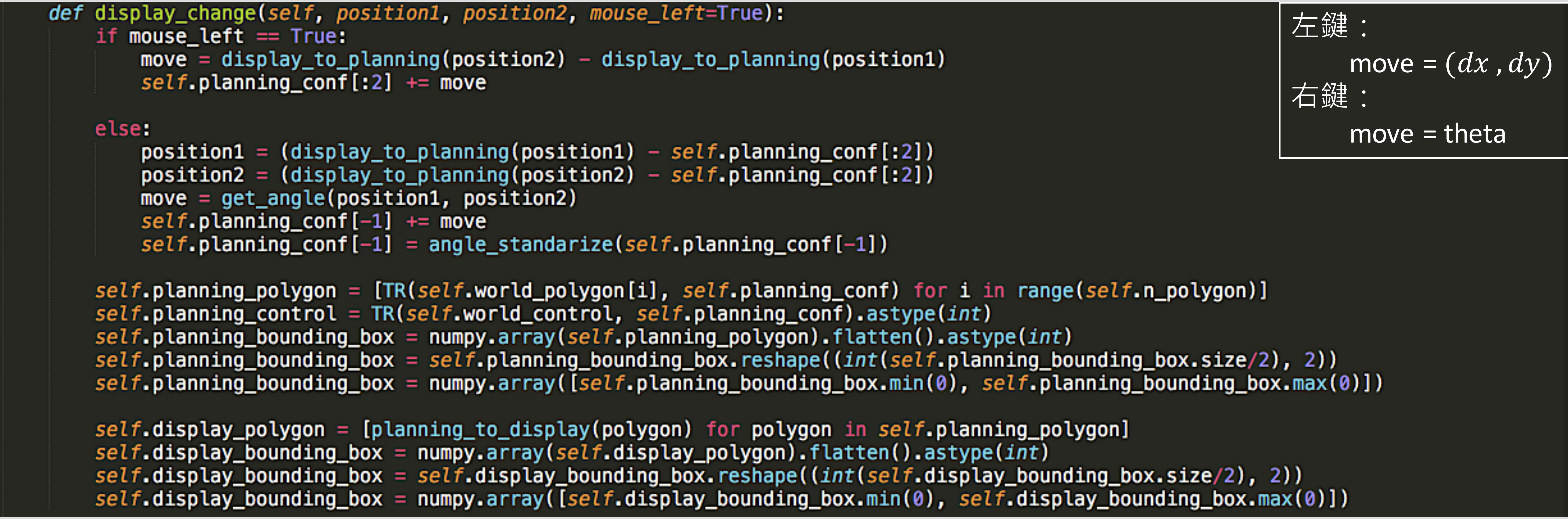
< Parameters >

game : pygame.display

color : tuple

< Return > None

Robots – Class.function



```
    def display_change(self, position1, position2, mouse_left=True):
        if mouse_left == True:
            move = display_to_planning(position2) - display_to_planning(position1)
            self.planning_conf[:2] += move

        else:
            position1 = (display_to_planning(position1) - self.planning_conf[:2])
            position2 = (display_to_planning(position2) - self.planning_conf[:2])
            move = get_angle(position1, position2)
            self.planning_conf[-1] += move
            self.planning_conf[-1] = angle_standarize(self.planning_conf[-1])

        self.planning_polygon = [TR(self.world_polygon[i], self.planning_conf) for i in range(self.n_polygon)]
        self.planning_control = TR(self.world_control, self.planning_conf).astype(int)
        self.planning_bounding_box = numpy.array(self.planning_polygon).flatten().astype(int)
        self.planning_bounding_box = self.planning_bounding_box.reshape((int(self.planning_bounding_box.size/2), 2))
        self.planning_bounding_box = numpy.array([self.planning_bounding_box.min(0), self.planning_bounding_box.max(0)])

        self.display_polygon = [planning_to_display(polygon) for polygon in self.planning_polygon]
        self.display_bounding_box = numpy.array(self.display_polygon).flatten().astype(int)
        self.display_bounding_box = self.display_bounding_box.reshape((int(self.display_bounding_box.size/2), 2))
        self.display_bounding_box = numpy.array([self.display_bounding_box.min(0), self.display_bounding_box.max(0)])
```

左鍵：

move = $(dx\ , dy)$

右鍵：

move = theta

robots.displaty_change (position1 , position2 , mouse_left = True)

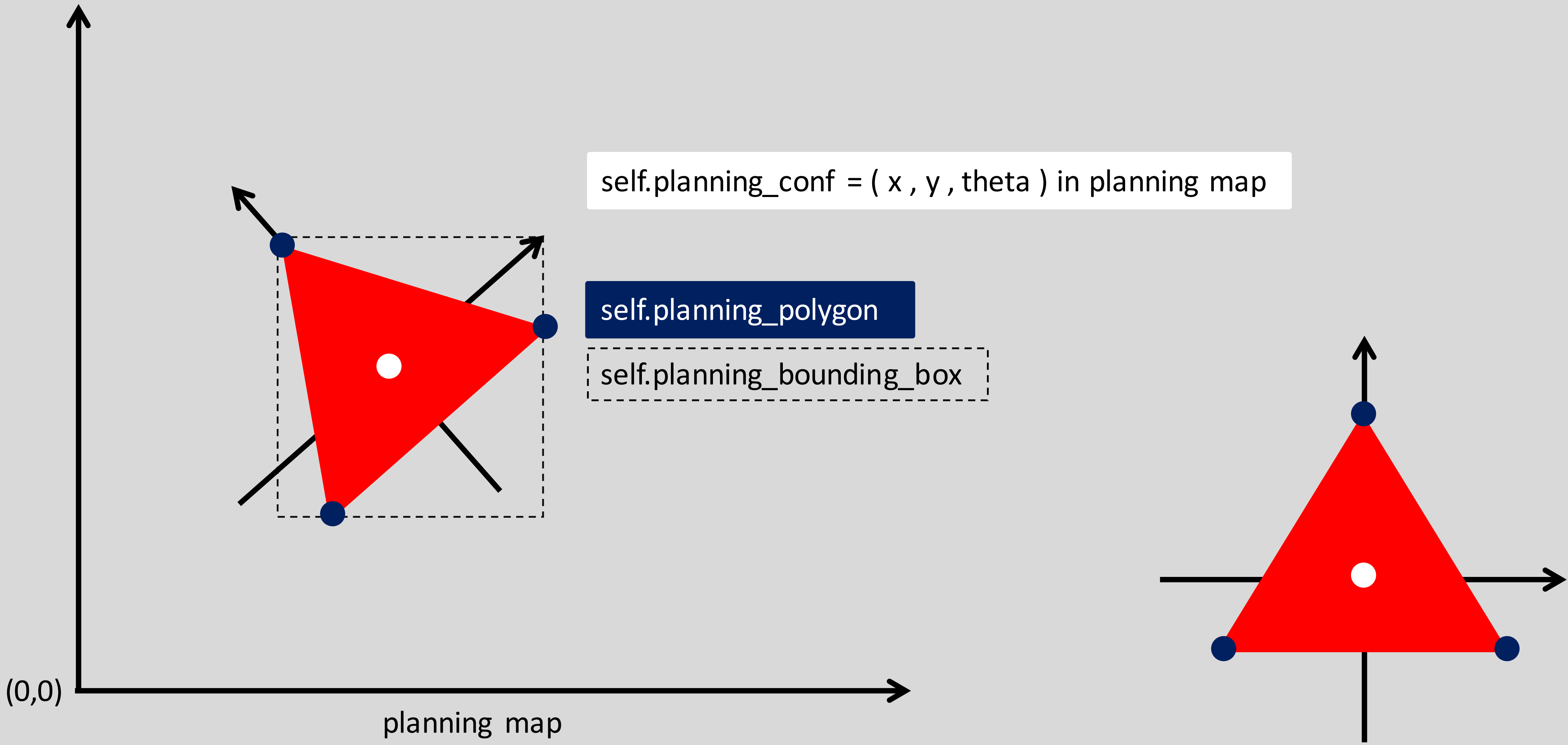
物件被拖移時，先更改 planning map 的屬性，再反映在 display map

< Parameters >

position1 : numpy.ndarray (1D)

< Return > None

Obstacles – Class



self.planning_conf = (x , y , theta) in planning map

self.planning_polygon

self.planning_bounding_box

(0,0)

planning map

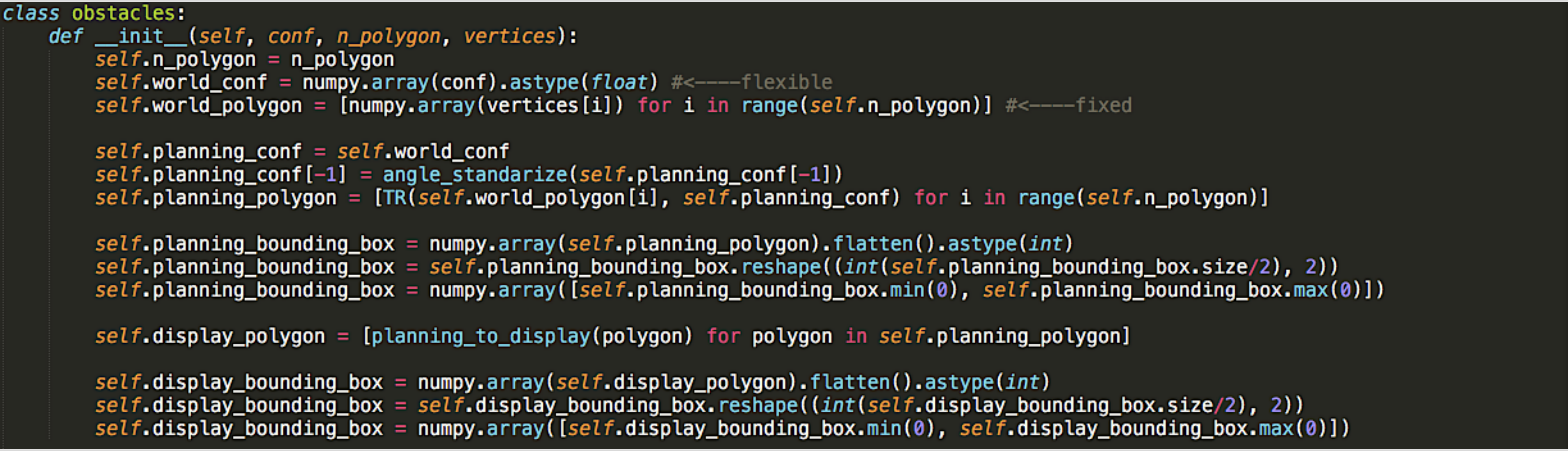
Obstacles – Class

<Parameters>

conf : tuple/list

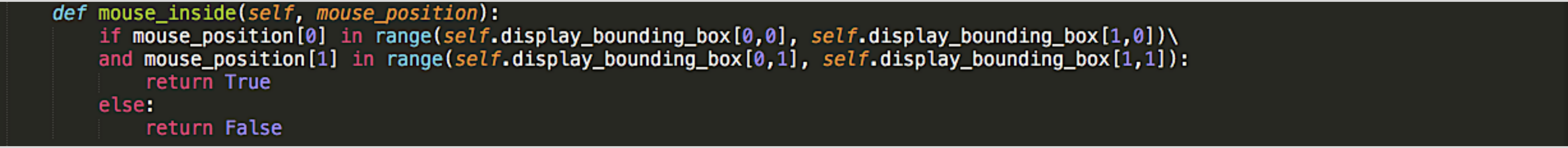
n_polygon : int

vertices : list and counterclockwise ordered (3D)



```
class obstacles:
    def __init__(self, conf, n_polygon, vertices):
        self.n_polygon = n_polygon
        self.world_conf = numpy.array(conf).astype(float) #<----flexible
        self.world_polygon = [numpy.array(vertices[i]) for i in range(self.n_polygon)] #<----fixed

        self.planning_conf = self.world_conf
        self.planning_conf[-1] = angle_standarize(self.planning_conf[-1])
        self.planning_polygon = [TR(self.world_polygon[i], self.planning_conf) for i in range(self.n_polygon)]

        self.planning_bounding_box = numpy.array(self.planning_polygon).flatten().astype(int)
        self.planning_bounding_box = self.planning_bounding_box.reshape((int(self.planning_bounding_box.size/2), 2))
        self.planning_bounding_box = numpy.array([self.planning_bounding_box.min(0), self.planning_bounding_box.max(0)])

        self.display_polygon = [planning_to_display(polygon) for polygon in self.planning_polygon]

        self.display_bounding_box = numpy.array(self.display_polygon).flatten().astype(int)
        self.display_bounding_box = self.display_bounding_box.reshape((int(self.display_bounding_box.size/2), 2))
        self.display_bounding_box = numpy.array([self.display_bounding_box.min(0), self.display_bounding_box.max(0)])
```

Obstacles – Class.function



```
    def mouse_inside(self, mouse_position):
        if mouse_position[0] in range(self.display_bounding_box[0,0], self.display_bounding_box[1,0])\
        and mouse_position[1] in range(self.display_bounding_box[0,1], self.display_bounding_box[1,1]):
            return True
        else:
            return False
```

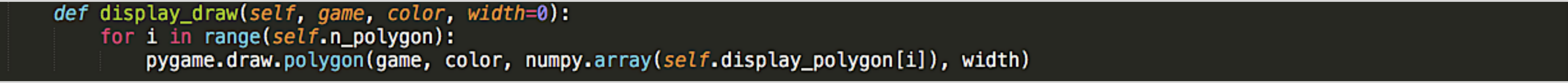
obstacles.mouse_inside (mouse_position)

游標是否落在 obstacle 的bounding box裡面；使用在物件拖移

< Parameters >

mouse_position : list (1D)

< Return > Boolean



```
    def display_draw(self, game, color, width=0):
        for i in range(self.n_polygon):
            pygame.draw.polygon(game, color, numpy.array(self.display_polygon[i]), width)
```

obstacles.displaty_draw (game , color , width = 0)

物件畫在 display map

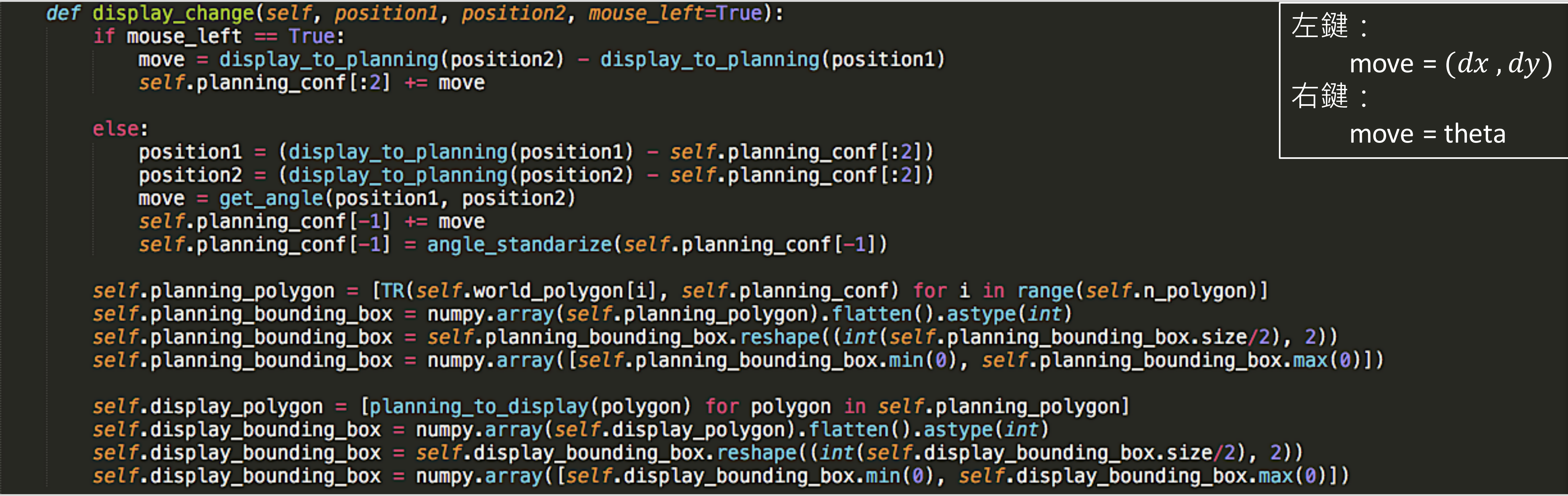
< Parameters >

game : pygame.display

color : tuple

< Return > None

Obstacles – Class.function



```python
def display_change(self, position1, position2, mouse_left=True):
    if mouse_left == True:
        move = display_to_planning(position2) - display_to_planning(position1)
        self.planning_conf[:2] += move

    else:
        position1 = (display_to_planning(position1) - self.planning_conf[:2])
        position2 = (display_to_planning(position2) - self.planning_conf[:2])
        move = get_angle(position1, position2)
        self.planning_conf[-1] += move
        self.planning_conf[-1] = angle_standarize(self.planning_conf[-1])

    self.planning_polygon = [TR(self.world_polygon[i], self.planning_conf) for i in range(self.n_polygon)]
    self.planning_bounding_box = numpy.array(self.planning_polygon).flatten().astype(int)
    self.planning_bounding_box = self.planning_bounding_box.reshape((int(self.planning_bounding_box.size/2), 2))
    self.planning_bounding_box = numpy.array([self.planning_bounding_box.min(0), self.planning_bounding_box.max(0)])

    self.display_polygon = [planning_to_display(polygon) for polygon in self.planning_polygon]
    self.display_bounding_box = numpy.array(self.display_polygon).flatten().astype(int)
    self.display_bounding_box = self.display_bounding_box.reshape((int(self.display_bounding_box.size/2), 2))
    self.display_bounding_box = numpy.array([self.display_bounding_box.min(0), self.display_bounding_box.max(0)])
```

左鍵：

move = $(dx\ ,dy)$

右鍵：

move = theta

obstacles.displaty_change (position1 , position2 , mouse_left = True)

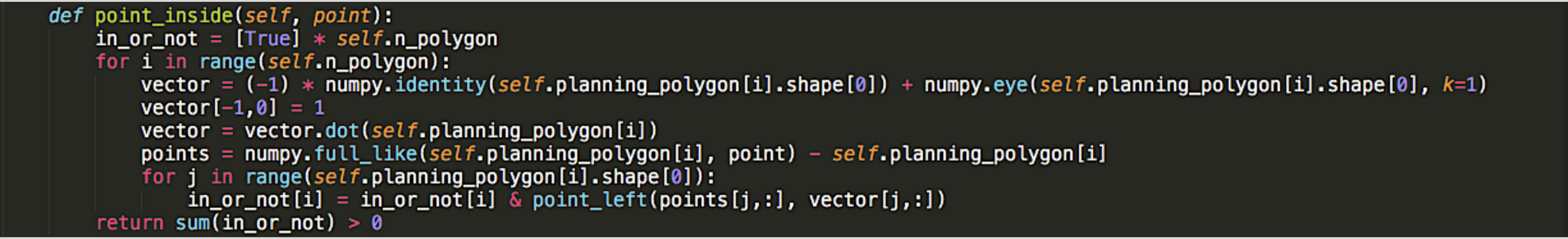
物件被拖移時，先更改 planning map 的屬性，再反映在 display map

< Parameters >

position1 ：numpy.ndarray (1D)

< Return > None

Obstacles – Class.function



```
    def point_inside(self, point):
        in_or_not = [True] * self.n_polygon
        for i in range(self.n_polygon):
            vector = (-1) * numpy.identity(self.planning_polygon[i].shape[0]) + numpy.eye(self.planning_polygon[i].shape[0], k=1)
            vector[-1,0] = 1
            vector = vector.dot(self.planning_polygon[i])
            points = numpy.full_like(self.planning_polygon[i], point) - self.planning_polygon[i]
            for j in range(self.planning_polygon[i].shape[0]):
                in_or_not[i] = in_or_not[i] & point_left(points[j,:], vector[j,:])
        return sum(in_or_not) > 0
```

obstacles.point_inside (point)

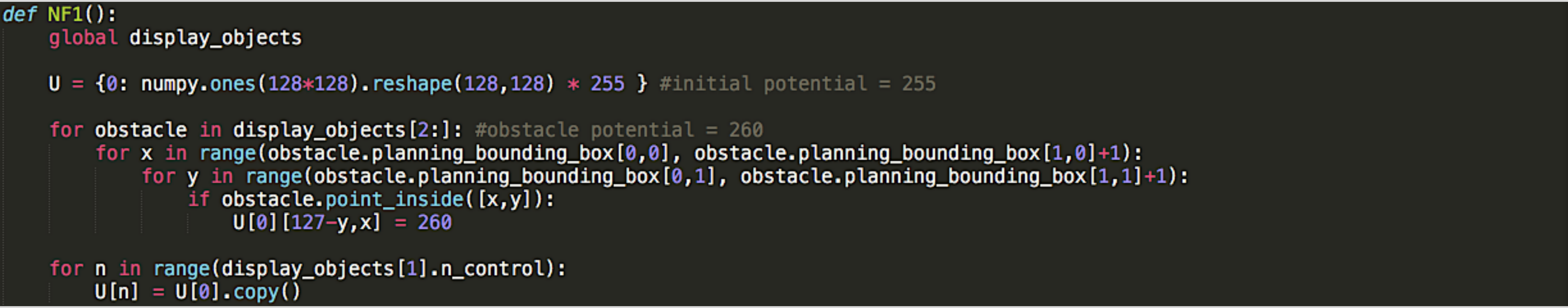
計算 potential field 時，要把 obstacle 內部的點都填滿 (potential ← 260)，做為判斷的function

< Parameters >

point : tuple/list (1D)

< Return > Boolean

Function – NF1

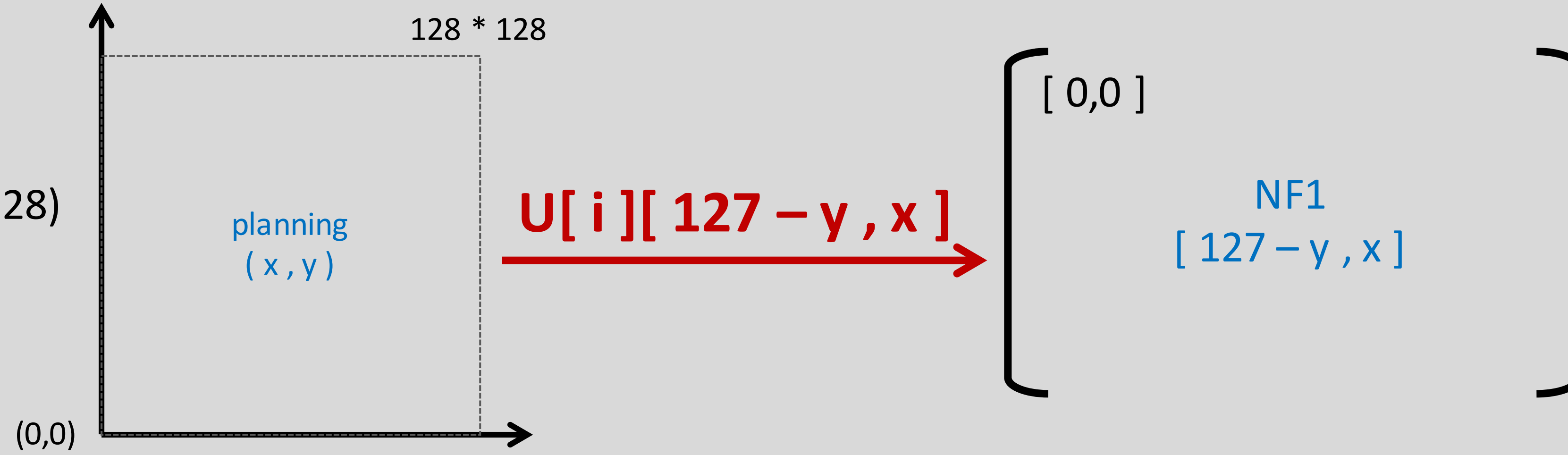


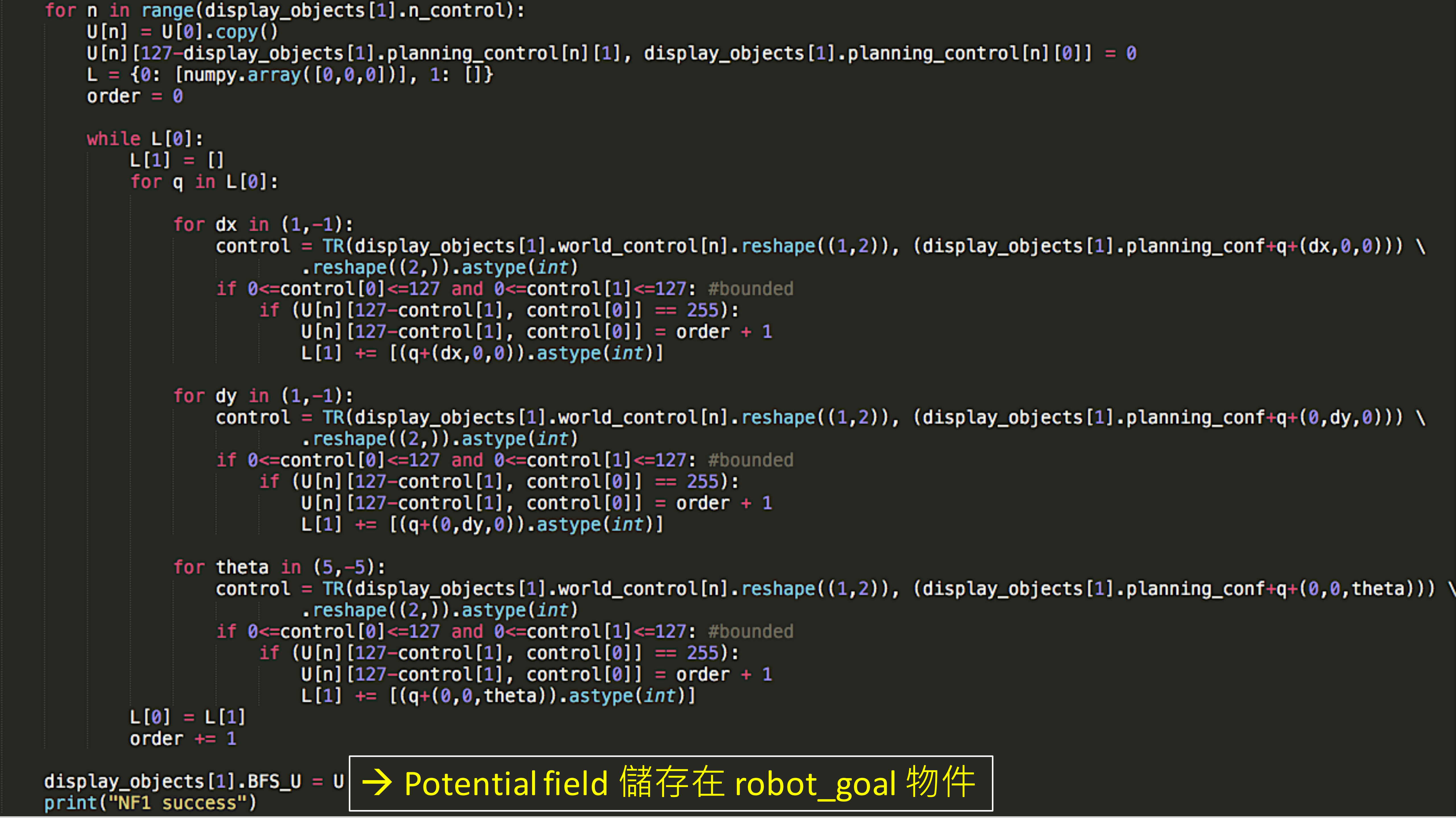
```
def NF1():
    global display_objects

    U = {0: numpy.ones(128*128).reshape(128,128) * 255 } #initial potential = 255

    for obstacle in display_objects[2:]: #obstacle potential = 260
        for x in range(obstacle.planning_bounding_box[0,0], obstacle.planning_bounding_box[1,0]+1):
            for y in range(obstacle.planning_bounding_box[0,1], obstacle.planning_bounding_box[1,1]+1):
                if obstacle.point_inside([x,y]):
                    U[0][127-y,x] = 260

    for n in range(display_objects[1].n_control):
        U[n] = U[0].copy()
```

U = { 0 : numpy.ndarray ,
    1 : numpy.ndarray }

Order of control point

Potential field (shape = 128 * 128)
initial = 255
obstacle = 260

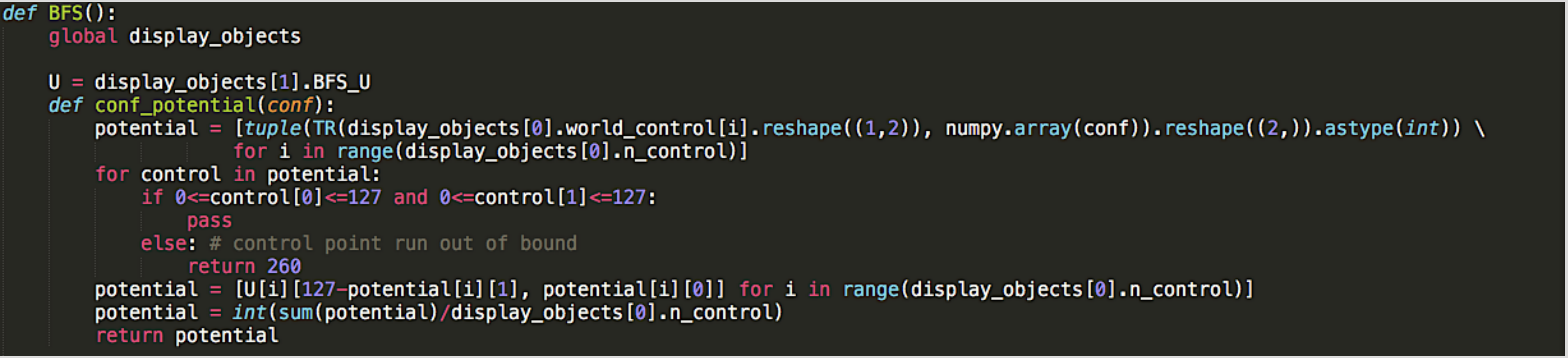




```
for n in range(display_objects[1].n_control):
    U[n] = U[0].copy()
    U[n][127-display_objects[1].planning_control[n][1], display_objects[1].planning_control[n][0]] = 0
    L = {0: [numpy.array([0,0,0])], 1: []}
    order = 0

    while L[0]:
        L[1] = []
        for q in L[0]:

            for dx in (1,-1):
                control = TR(display_objects[1].world_control[n].reshape((1,2)), (display_objects[1].planning_conf+q+(dx,0,0))) \
                        .reshape((2,)).astype(int)
                if 0<=control[0]<=127 and 0<=control[1]<=127: #bounded
                    if (U[n][127-control[1], control[0]] == 255):
                        U[n][127-control[1], control[0]] = order + 1
                        L[1] += [(q+(dx,0,0)).astype(int)]

            for dy in (1,-1):
                control = TR(display_objects[1].world_control[n].reshape((1,2)), (display_objects[1].planning_conf+q+(0,dy,0))) \
                        .reshape((2,)).astype(int)
                if 0<=control[0]<=127 and 0<=control[1]<=127: #bounded
                    if (U[n][127-control[1], control[0]] == 255):
                        U[n][127-control[1], control[0]] = order + 1
                        L[1] += [(q+(0,dy,0)).astype(int)]

            for theta in (5,-5):
                control = TR(display_objects[1].world_control[n].reshape((1,2)), (display_objects[1].planning_conf+q+(0,0,theta))) \
                        .reshape((2,)).astype(int)
                if 0<=control[0]<=127 and 0<=control[1]<=127: #bounded
                    if (U[n][127-control[1], control[0]] == 255):
                        U[n][127-control[1], control[0]] = order + 1
                        L[1] += [(q+(0,0,theta)).astype(int)]
        L[0] = L[1]
        order += 1

display_objects[1].BFS_U = U
print("NF1 success")
```

→ Potential field 儲存在 robot_goal 物件

Function – BFS



```
def BFS():
    global display_objects

    U = display_objects[1].BFS_U
    def conf_potential(conf):
        potential = [tuple(TR(display_objects[0].world_control[i].reshape((1,2)), numpy.array(conf)).reshape((2,)).astype(int)) \
                    for i in range(display_objects[0].n_control)]
        for control in potential:
            if 0<=control[0]<=127 and 0<=control[1]<=127:
                pass
            else: # control point run out of bound
                return 260
        potential = [U[i][127-potential[i][1], potential[i][0]] for i in range(display_objects[0].n_control)]
        potential = int(sum(potential)/display_objects[0].n_control)
        return potential
```

conf_potential (conf)

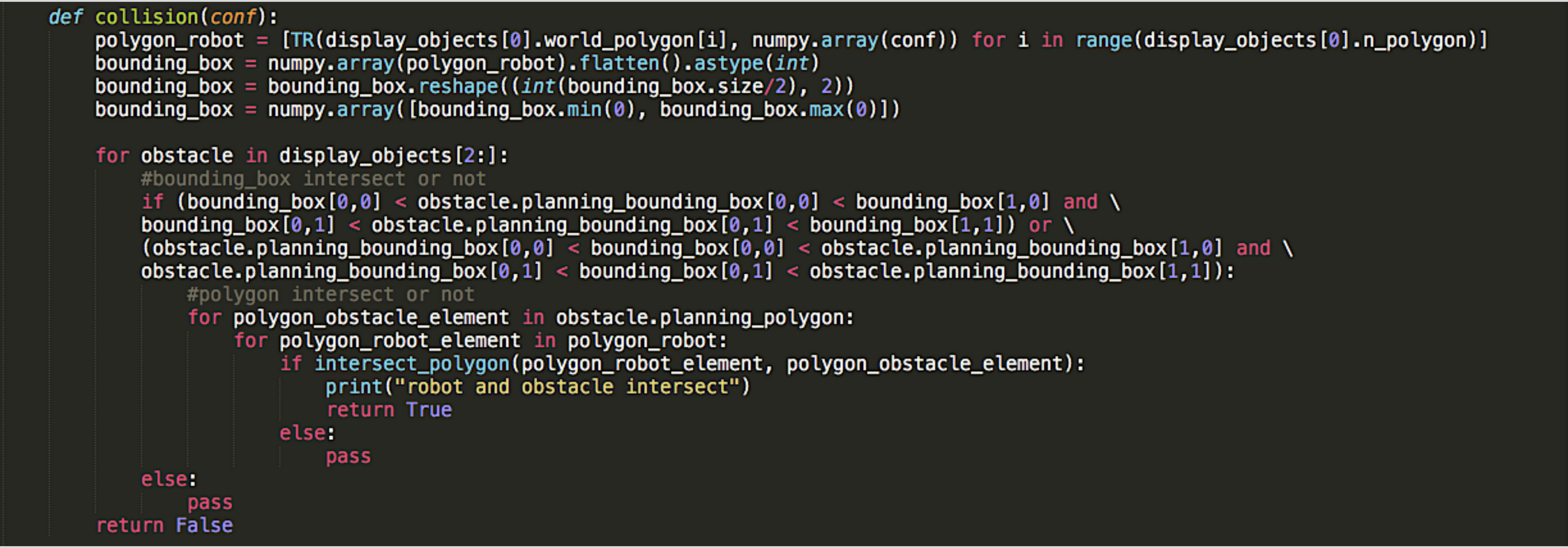
根據 configuration 找出 planning_control 的位置，再找儲存在 robot_goal 裡面的 potential field，用平均的方式 (**sum(potential) / n_control**)得到 potential 值

< Parameters >

conf : tuple

< Return > int

Function – BFS



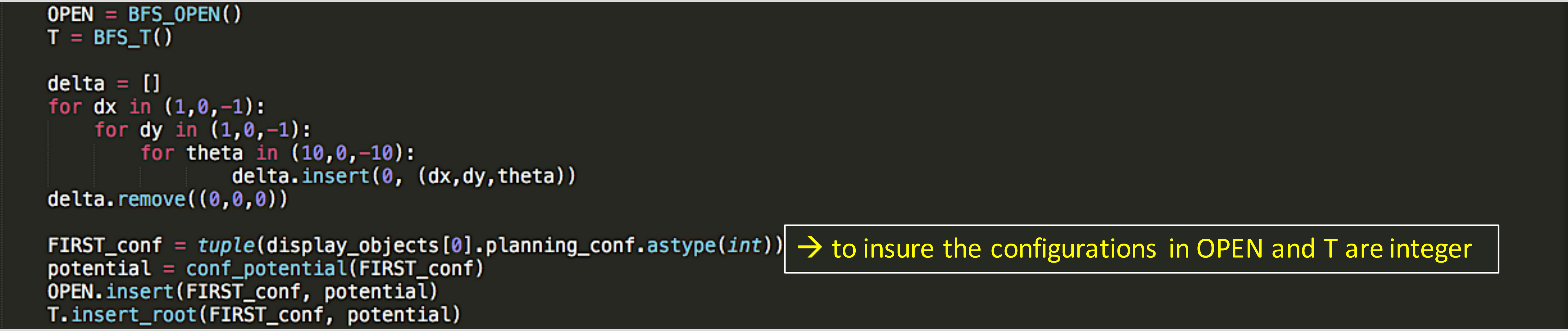
```
    def collision(conf):
        polygon_robot = [TR(display_objects[0].world_polygon[i], numpy.array(conf)) for i in range(display_objects[0].n_polygon)]
        bounding_box = numpy.array(polygon_robot).flatten().astype(int)
        bounding_box = bounding_box.reshape((int(bounding_box.size/2), 2))
        bounding_box = numpy.array([bounding_box.min(0), bounding_box.max(0)])

        for obstacle in display_objects[2:]:
            #bounding_box intersect or not
            if (bounding_box[0,0] < obstacle.planning_bounding_box[0,0] < bounding_box[1,0] and \
            bounding_box[0,1] < obstacle.planning_bounding_box[0,1] < bounding_box[1,1]) or \
            (obstacle.planning_bounding_box[0,0] < bounding_box[0,0] < obstacle.planning_bounding_box[1,0] and \
            obstacle.planning_bounding_box[0,1] < bounding_box[0,1] < obstacle.planning_bounding_box[1,1]):
                #polygon intersect or not
                for polygon_obstacle_element in obstacle.planning_polygon:
                    for polygon_robot_element in polygon_robot:
                        if intersect_polygon(polygon_robot_element, polygon_obstacle_element):
                            print("robot and obstacle intersect")
                            return True
                        else:
                            pass
            else:
                pass
        return False
```

collision (conf)

根據 configuration 找出 polygon 的位置，檢查是否有碰撞

< Return > Boolean

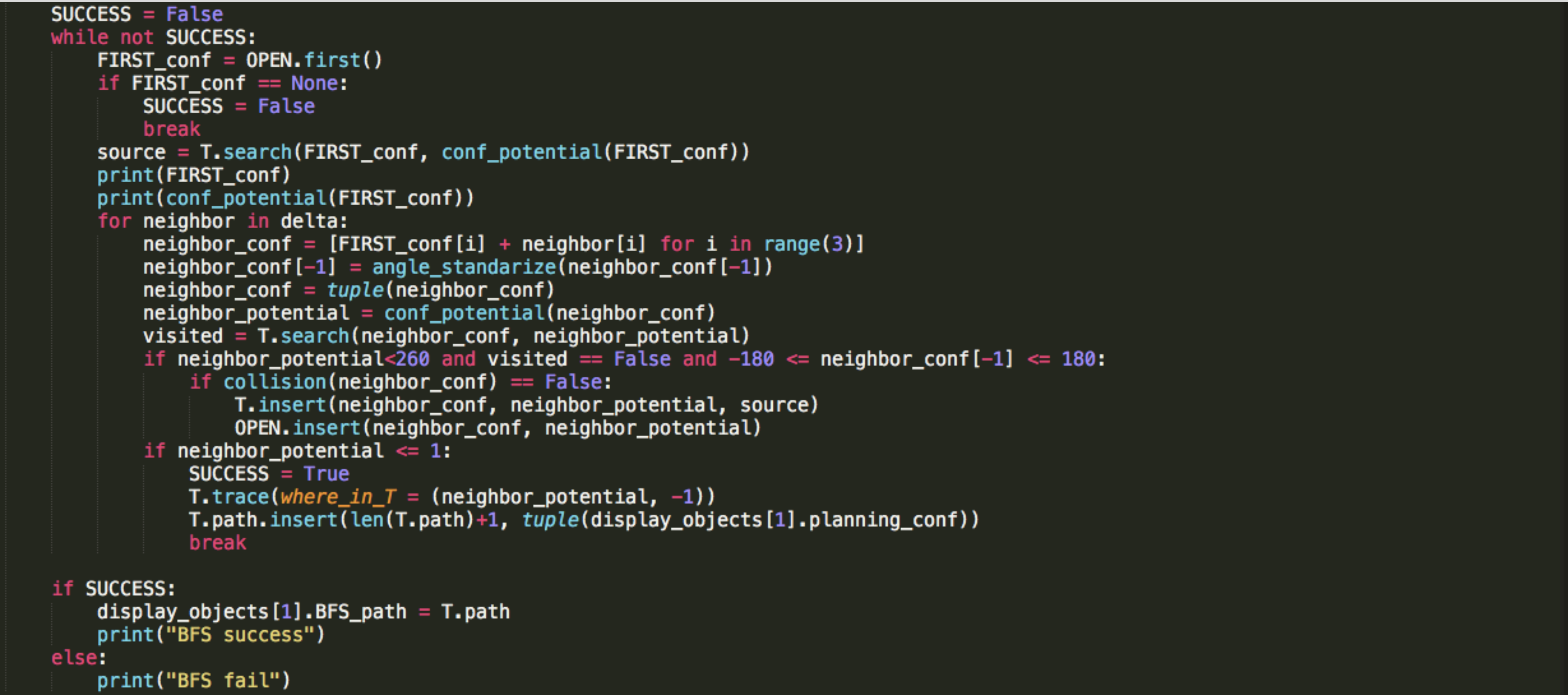
Function – BFS



```
OPEN = BFS_OPEN()
T = BFS_T()

delta = []
for dx in (1,0,-1):
    for dy in (1,0,-1):
        for theta in (10,0,-10):
                delta.insert(0, (dx,dy,theta))
delta.remove((0,0,0))

FIRST_conf = tuple(display_objects[0].planning_conf.astype(int))
potential = conf_potential(FIRST_conf)
OPEN.insert(FIRST_conf, potential)
T.insert_root(FIRST_conf, potential)
```

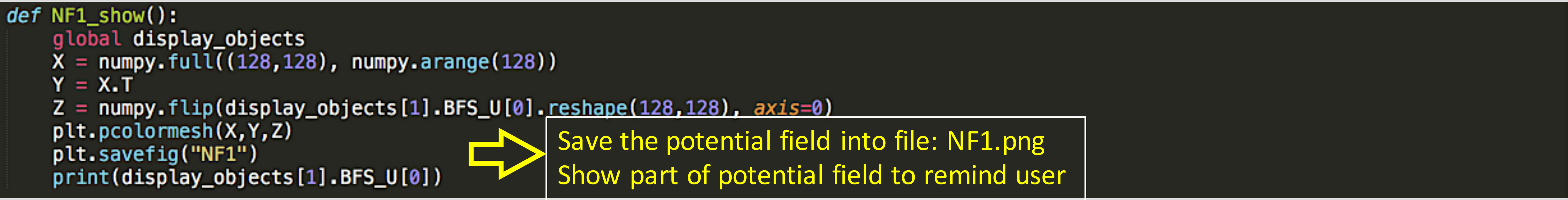
→ to insure the configurations in OPEN and T are integer

Function – BFS



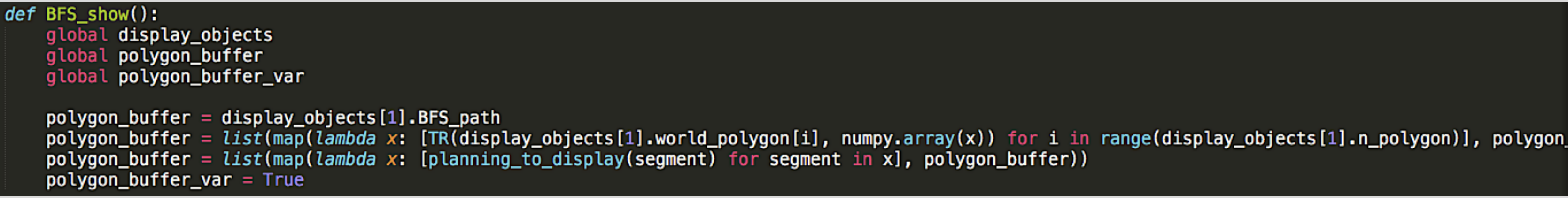
```python
SUCCESS = False
while not SUCCESS:
    FIRST_conf = OPEN.first()
    if FIRST_conf == None:
        SUCCESS = False
        break
    source = T.search(FIRST_conf, conf_potential(FIRST_conf))
    print(FIRST_conf)
    print(conf_potential(FIRST_conf))
    for neighbor in delta:
        neighbor_conf = [FIRST_conf[i] + neighbor[i] for i in range(3)]
        neighbor_conf[-1] = angle_standarize(neighbor_conf[-1])
        neighbor_conf = tuple(neighbor_conf)
        neighbor_potential = conf_potential(neighbor_conf)
        visited = T.search(neighbor_conf, neighbor_potential)
        if neighbor_potential<260 and visited == False and -180 <= neighbor_conf[-1] <= 180:
            if collision(neighbor_conf) == False:
                T.insert(neighbor_conf, neighbor_potential, source)
                OPEN.insert(neighbor_conf, neighbor_potential)
        if neighbor_potential <= 1:
            SUCCESS = True
            T.trace(where_in_T = (neighbor_potential, -1))
            T.path.insert(len(T.path)+1, tuple(display_objects[1].planning_conf))
            break

if SUCCESS:
    display_objects[1].BFS_path = T.path
    print("BFS success")
else:
    print("BFS fail")
```

Function – NF1_show , BFS_show



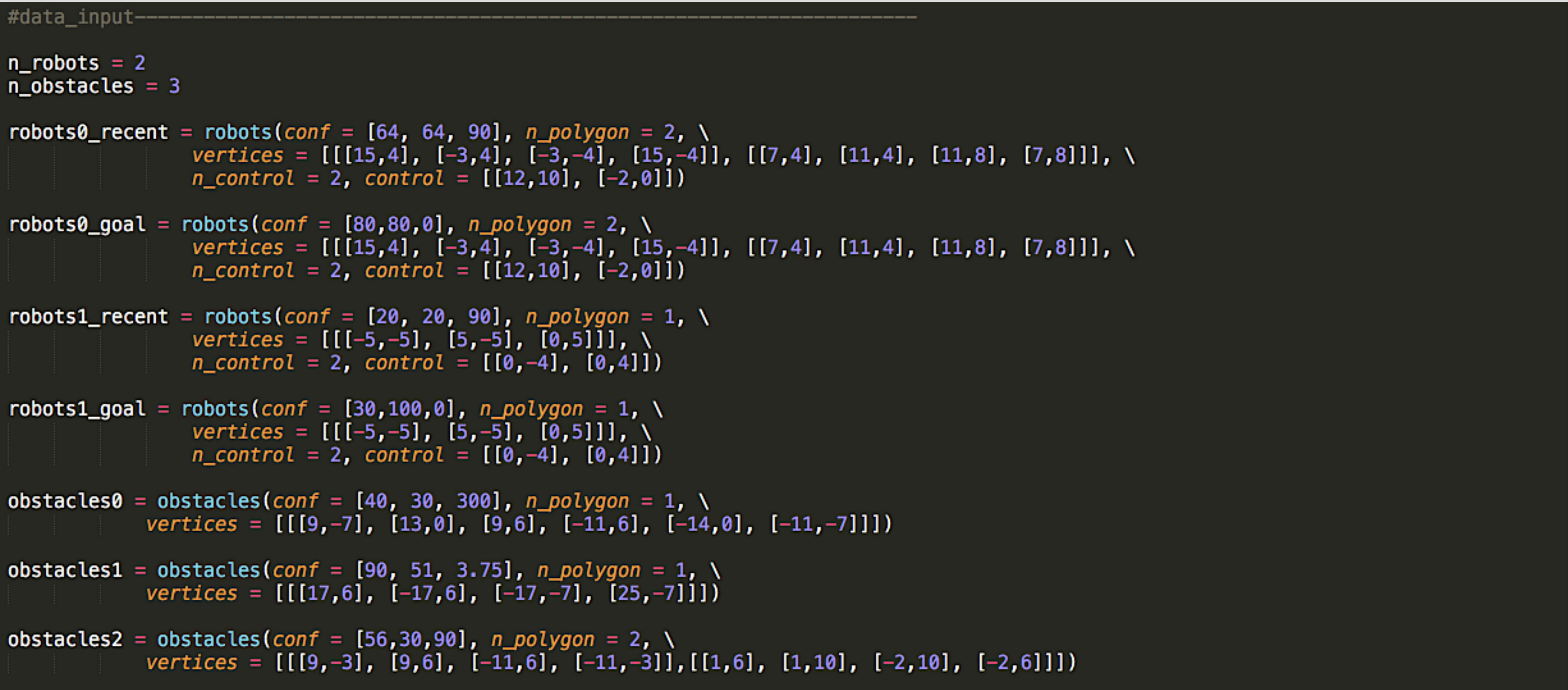
```
def NF1_show():
    global display_objects
    X = numpy.full((128,128), numpy.arange(128))
    Y = X.T
    Z = numpy.flip(display_objects[1].BFS_U[0].reshape(128,128), axis=0)
    plt.pcolormesh(X,Y,Z)
    plt.savefig("NF1")
    print(display_objects[1].BFS_U[0])
```

Save the potential field into file: NF1.png
Show part of potential field to remind user



```
def BFS_show():
    global display_objects
    global polygon_buffer
    global polygon_buffer_var

    polygon_buffer = display_objects[1].BFS_path
    polygon_buffer = list(map(lambda x: [TR(display_objects[1].world_polygon[i], numpy.array(x)) for i in range(display_objects[1].n_polygon)], polygon_
    polygon_buffer = list(map(lambda x: [planning_to_display(segment) for segment in x], polygon_buffer))
    polygon_buffer_var = True
```

Input Data



```
#data_input--------------------------------------------------------------------

n_robots = 2
n_obstacles = 3

robots0_recent = robots(conf = [64, 64, 90], n_polygon = 2, \
                vertices = [[[15,4], [-3,4], [-3,-4], [15,-4]], [[7,4], [11,4], [11,8], [7,8]]], \
                n_control = 2, control = [[12,10], [-2,0]])

robots0_goal = robots(conf = [80,80,0], n_polygon = 2, \
                vertices = [[[15,4], [-3,4], [-3,-4], [15,-4]], [[7,4], [11,4], [11,8], [7,8]]], \
                n_control = 2, control = [[12,10], [-2,0]])

robots1_recent = robots(conf = [20, 20, 90], n_polygon = 1, \
                vertices = [[[-5,-5], [5,-5], [0,5]]], \
                n_control = 2, control = [[0,-4], [0,4]])

robots1_goal = robots(conf = [30,100,0], n_polygon = 1, \
                vertices = [[[-5,-5], [5,-5], [0,5]]], \
                n_control = 2, control = [[0,-4], [0,4]])

obstacles0 = obstacles(conf = [40, 30, 300], n_polygon = 1, \
            vertices = [[[9,-7], [13,0], [9,6], [-11,6], [-14,0], [-11,-7]]])

obstacles1 = obstacles(conf = [90, 51, 3.75], n_polygon = 1, \
            vertices = [[[17,6], [-17,6], [-17,-7], [25,-7]]])

obstacles2 = obstacles(conf = [56,30,90], n_polygon = 2, \
            vertices = [[[9,-3], [9,6], [-11,6], [-11,-3]],[[1,6], [1,10], [-2,10], [-2,6]]])
```