Étude de l'emplacement de détecteurs de fumée dans une pièce

Elina FALIZE
Candidate n° 27000

TIPE

Session 2022

Présentation

La dangerosité de la fumée

Départ de fumée dans une pièce

Crédit : Conrado/Shutterstock

Présentation

Les détecteurs de fumée

Détecteur de fumée photoélectrique

Crédit : Nikkytok/Shutterstock

Peut-on déterminer un emplacement minimisant le temps de réponse d'un détecteur de fumée pour une pièce donnée ?

But : Implémenter un algorithme permettant de trouver l'emplacement idéal d'un détecteur

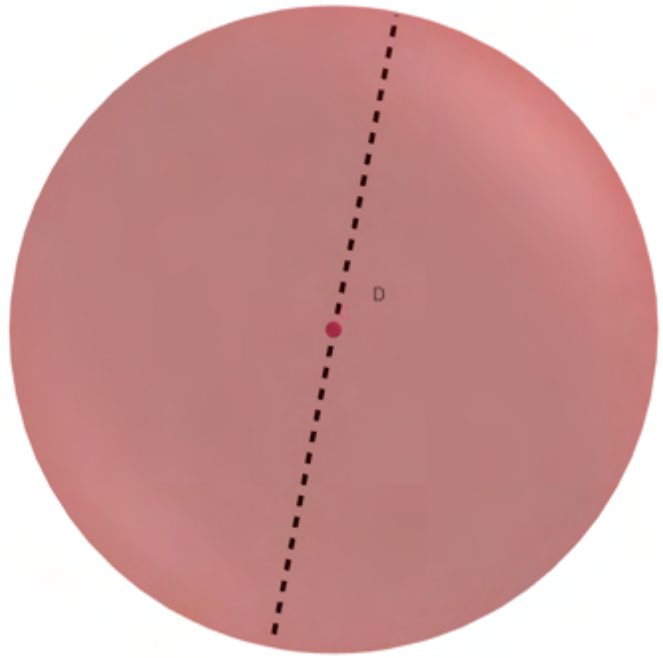
Plan

- Existence mathématique d'un point minimisant la distance aux autres points d'un volume
- Pièce rectangulaire à 1 détecteur
- Pièce rectangulaire à plusieurs détecteurs
- Pièce à géométrie quelconque à 1 détecteur

Point minimisant la distance aux autres points d'un volume

Problème

Volume V supposé : borné
fermé
dans $\mathbb{R}^3$



Norme :
$\forall M \in \mathbb{R}^3,\ \|M\| = \sqrt{x^2 + y^2 + z^2}$

Point minimisant la distance aux autres points d'un volume

Problème

Diamètre :

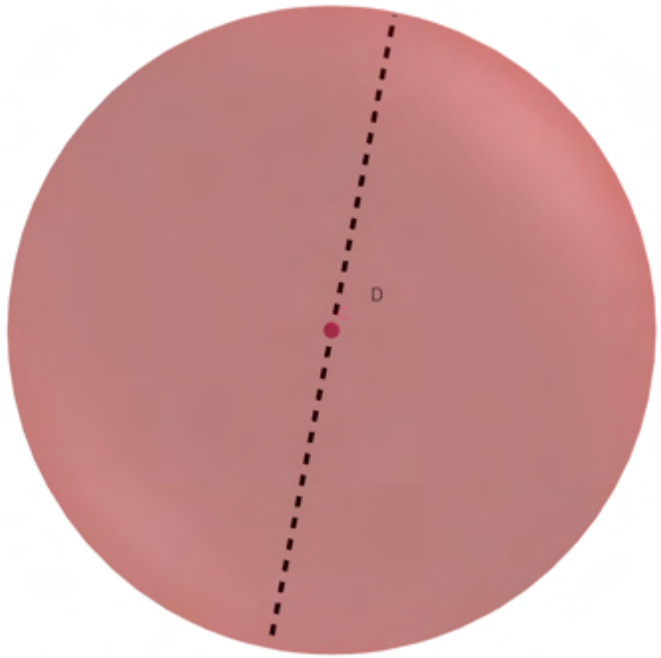
$$\boxed{D = \sup_{M,M' \in V} (\|M - M'\|)}$$

V borné :

$\Rightarrow \exists K \text{ t.q } \forall M \in V,\ \|M\| \leqslant K$

$\Rightarrow \|M - M'\| \leqslant 2K$

$\Rightarrow$ D existe



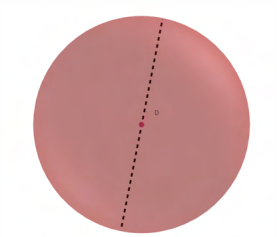
Point minimisant la distance aux autres points d'un volume

Existence

$$\begin{array}{rccc} \varphi : & \mathrm{V} & \longrightarrow & \mathbb{R} \\ & \mathrm{M} & \longmapsto & \iiint_{M' \in V} \|M - M'\| dxdydz \end{array}$$

$\{\varphi(\mathrm{M}),\ M \in V\}$ non vide et majoré par D

Point minimisant la distance à tous les points :
$\Rightarrow$ $\inf_{M \in V}(\varphi(M))$ existe



Point minimisant la distance aux autres points d'un volume

Qui est atteint

$$\begin{array}{lccc} \varphi : & V & \longrightarrow & \mathbb{R} \\ & M & \longmapsto & \iiint_{M' \in V} \|M - M'\| dxdydz \end{array}$$

$$\begin{aligned} |\varphi(M) - \varphi(N)| &= |\iiint_{M' \in V} \|M - M'\| \, dxdydz - \iiint_{M' \in V} \|N - M'\| \, dxdydz \\ &\leq |\iiint_{M' \in V} \|M - M'\| - \|N - M'\| \, dxdydz \, | \\ &\leq \iiint_{M' \in V} | \, \|M - M'\| - \|N - M'\| \, | dxdydz \\ &\leq Vol(V) \times \sup_{M' \in V} (|\|M - M'\| - \|N - M'\||) \end{aligned}$$

Point minimisant la distance aux autres points d'un volume

Qui est atteint

$$\begin{array}{rccc} \varphi : & V & \longrightarrow & \mathbb{R} \\ & M & \longmapsto & \iiint_{M' \in V} \|M - M'\| dxdydz \end{array}$$

$$|\varphi(M) - \varphi(N)| \leq Vol(V) \times \sup_{M' \in V} (|\|M - M'\| - \|N - M'\||)$$

$$\begin{aligned} \text{Or} : \|M - M'\| &= \|M - N + N - M'\| \\ &\leq \|M - N\| + \|N - M\| \\ \Rightarrow \|M - M'\| - \|N - M'\| &\leq \|M - N\| \end{aligned}$$

$$\Rightarrow |\varphi(M) - \varphi(N)| \leq Vol(V) \times \|M - N\|$$

$\Rightarrow \varphi$ est $Vol(V)$ - Lipschitzienne

Point minimisant la distance aux autres points d'un volume

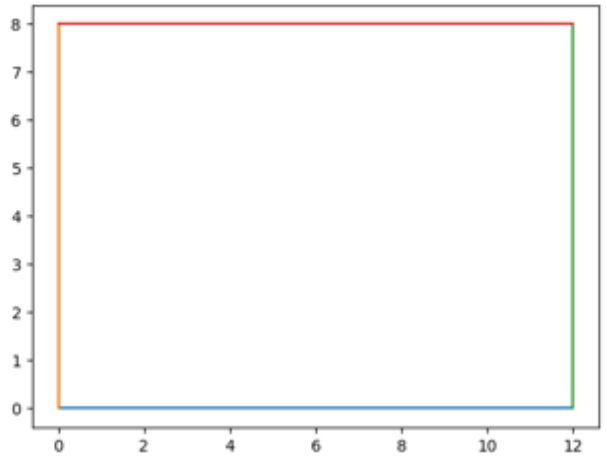
Qui est atteint

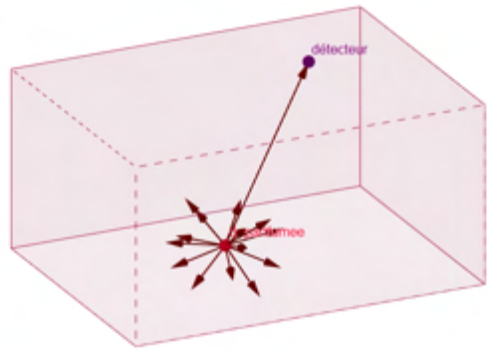
$$\left.\begin{array}{l} V \text{ borné} \\ V \text{ fermé} \\ \text{dim finie} \end{array}\right\} \Rightarrow V \text{compact} \left.\begin{array}{l} \\ \\ \\ \varphi \text{ continue} \end{array}\right\} \Rightarrow \text{donc cet inf est atteint}$$

Pièce rectangulaire à 1 détecteur

Hypothèses

- pièce rectangulaire
  - modélisée par ses coins
- déplacement de la fumée homogène
  - dans toutes les directions
  - à vitesse constante : 0.5m/s

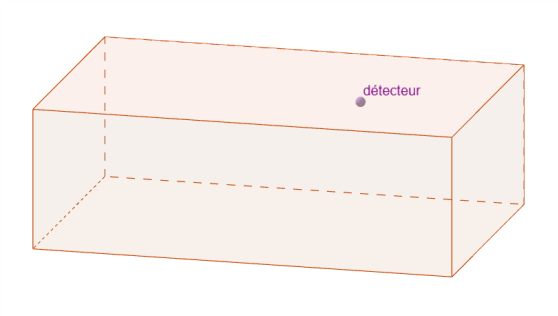


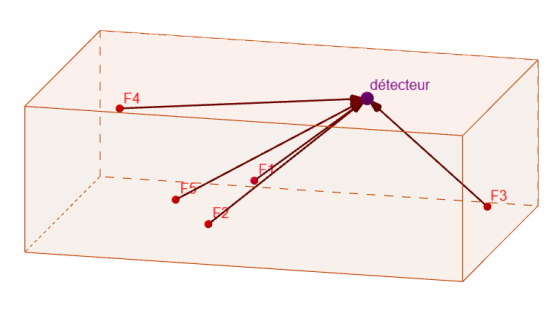


Pièce rectangulaire à 1 détecteur

Protocole

- Pour un détecteur donné : détermination du trajet de la fumée et du temps mis à le parcourir : moyenne pour 200 départs de fumées aléatoires

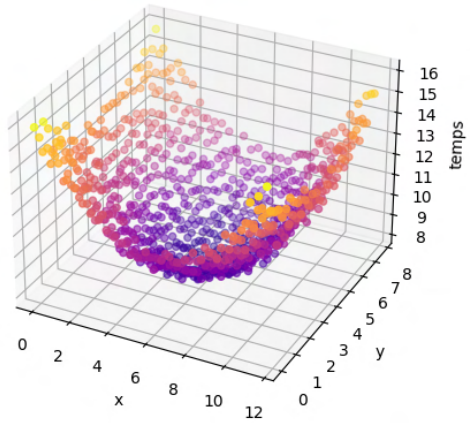


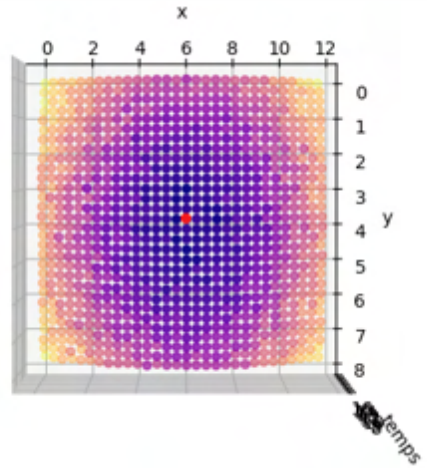


- Idem pour chaque couple de coordonnées de la pièce
- Etablissement des coordonnées pour lequel le temps est minimal
- Moyenne sur 30 de ces coordonnées au temps minimal

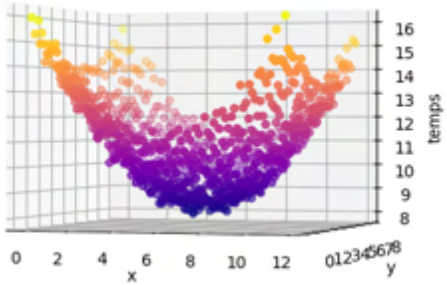
Pièce rectangulaire à 1 détecteur

Résultats





Temps mis pour que le détecteur se déclenche selon son emplacement

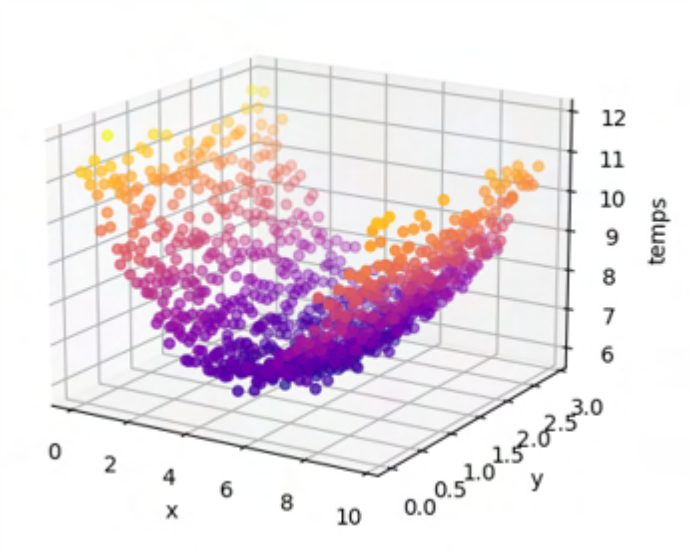


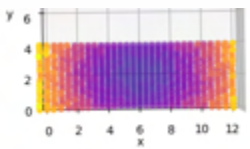
Temps minimal moyen	Coordonnées du détecteur
≈8.00s	(6.16, 4.24)
≈7.96s	(6.00, 4.05)
≈7.97s	(6.01, 3.98)
≈7.98s	(5.84, 3.87)
≈7.98s	(6.02, 3.96)
≈7.93s	(6.07, 4.17)

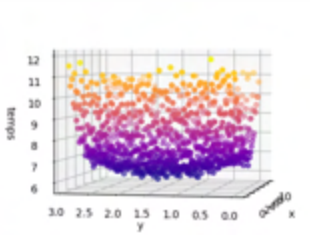
Pièce rectangulaire à 1 détecteur

Résultats pour d'autres pièces

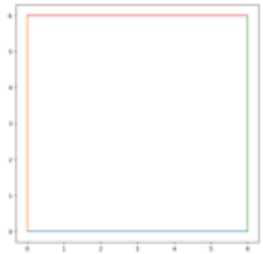
Pièce b

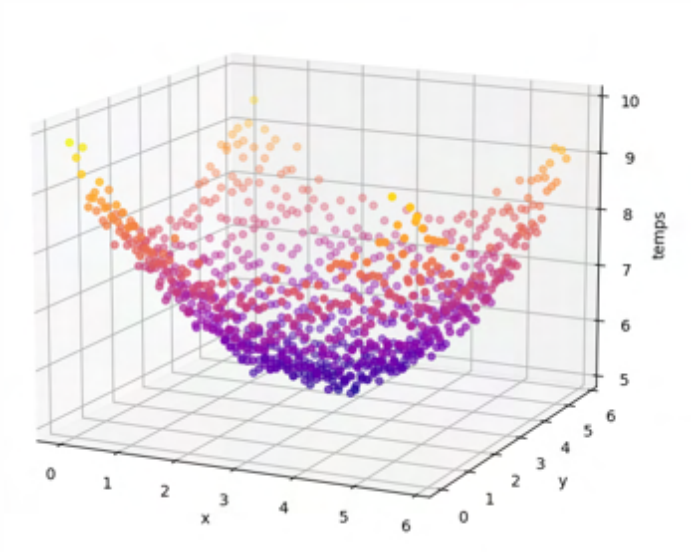


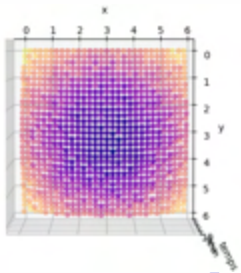


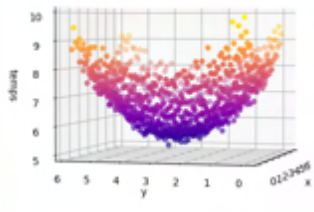


Pièce c





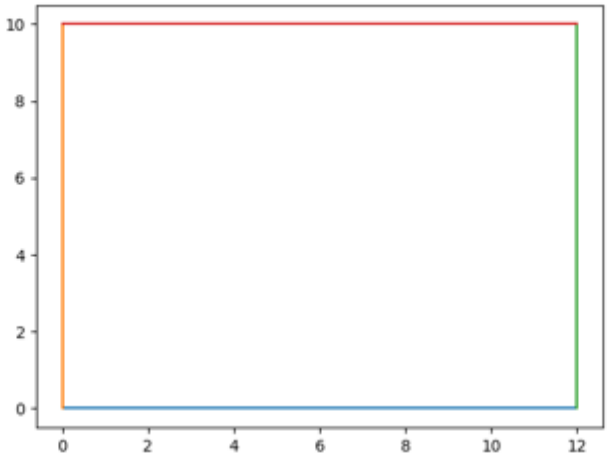


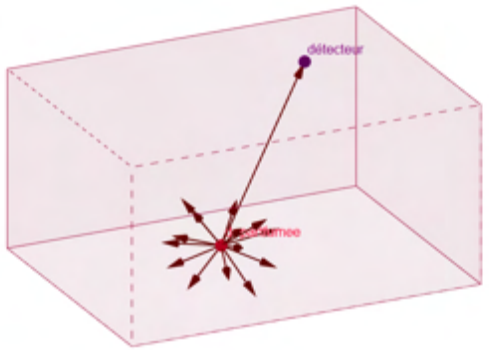


Pièce rectangulaire à plusieurs détecteurs

Hypothèses

- pièce rectangulaire
  - modélisée par ses coins
- déplacement de la fumée homogène
  - dans toutes les directions
  - à vitesse constante : 0.5m/s

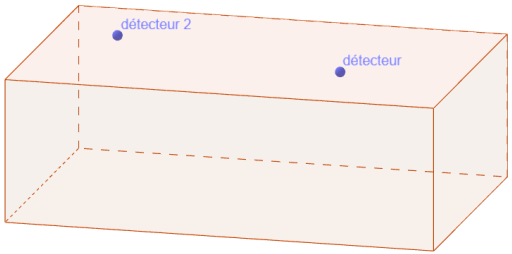


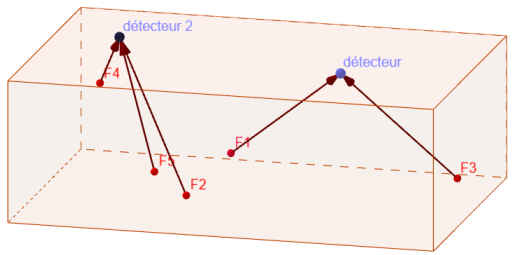


Pièce rectangulaire à plusieurs détecteurs

Protocole

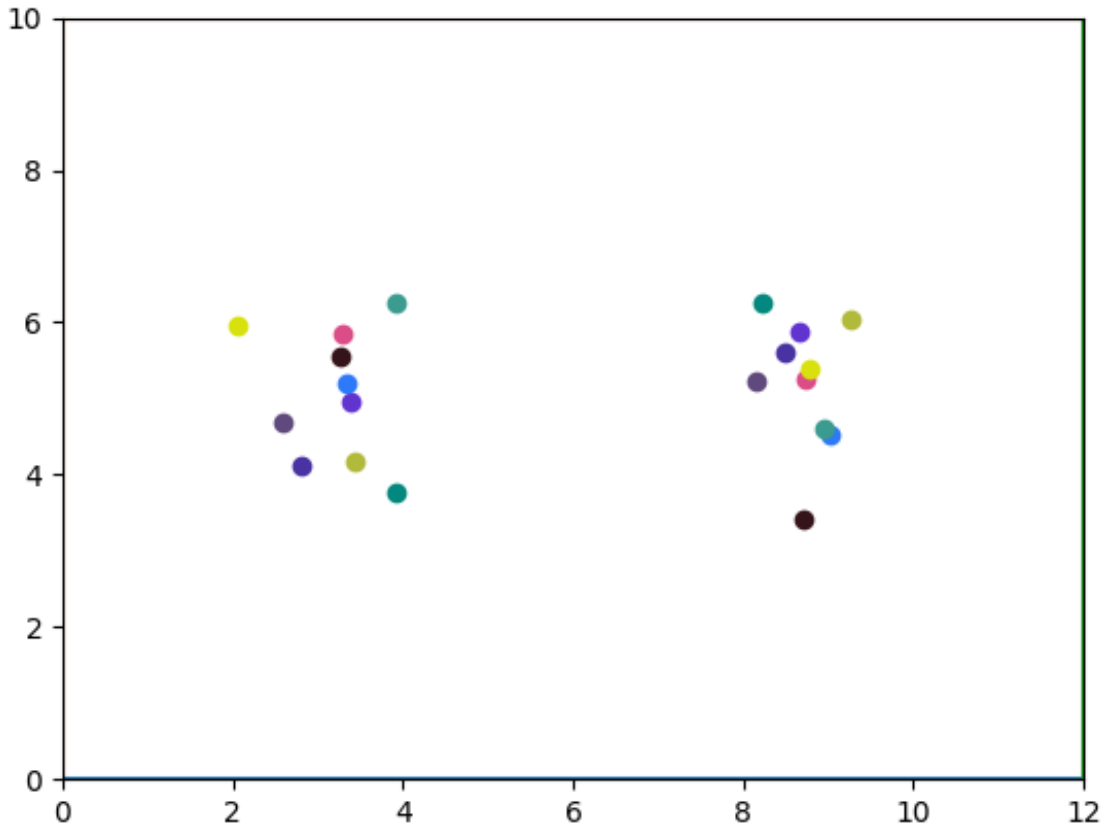
- Détermination du nombre de détecteurs de la pièce : 1/ 50m^2
- Pour un ensemble de détecteurs donné : détermination du trajet de la fumée et du temps mis à le parcourir : moyenne pour 1000 départs de fumées aléatoires





- Idem pour 5000 ensembles de détecteurs dans la pièce
- Etablissement de l'ensemble pour lequel le temps est minimal

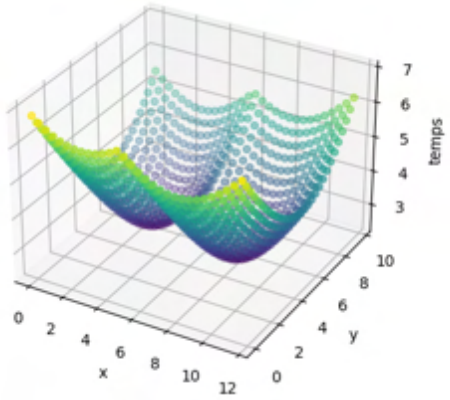
Emplacement des détecteurs

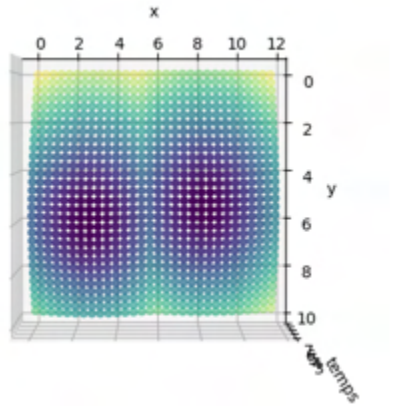


Pièce rectangulaire à plusieurs détecteurs

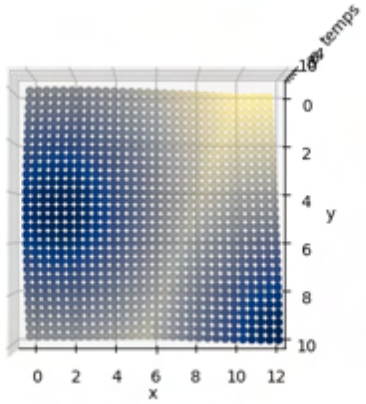
Résultats

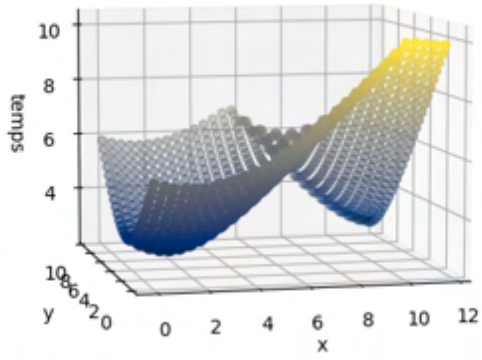
Pertinence des résultats





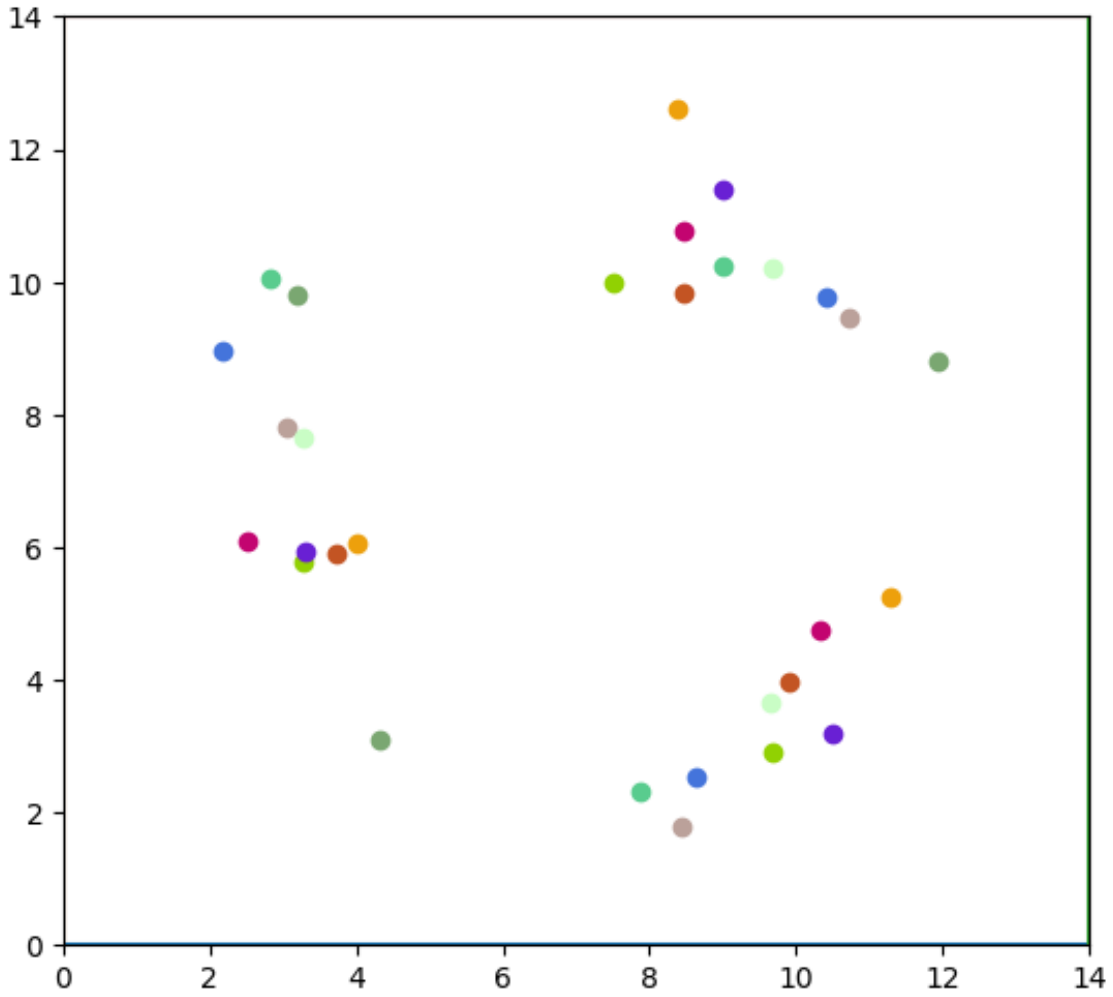
Temps mis par les détecteurs optimaux à se déclencher selon le départ de fumée





Temps mis par des détecteurs placés aléatoirement à se déclencher selon l'emplacement du départ de fumée

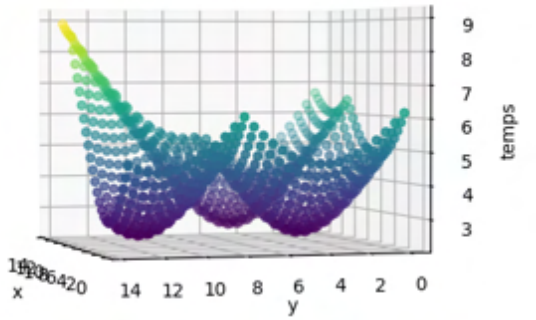
Autre exemple

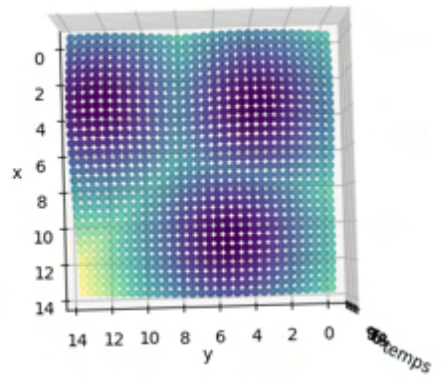


Pièce rectangulaire à plusieurs détecteurs

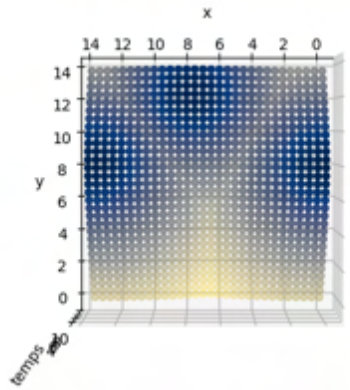
Résultats

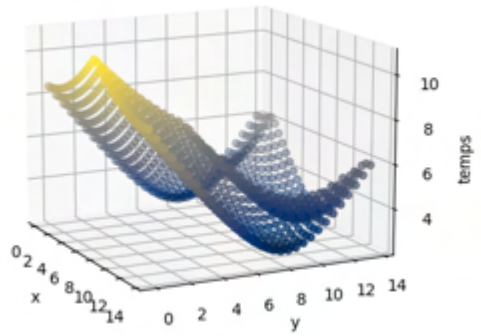
Pertinence des résultats





Temps mis par les détecteurs optimaux à se déclencher selon le départ de fumée





Temps mis par des détecteurs placés aléatoirement à se déclencher selon l'emplacement du départ de fumée

Pièce rectangulaire à plusieurs détecteur

Elements pouvant être améliorés

- Hypothèses de déplacement de la fumée :
  - Déplacement uniforme
- Forme de la pièce limitée

Pièce rectangulaire à plusieurs détecteur

Elements pouvant être améliorés

- Hypothèses de déplacement de la fumée :

  Déplacement uniforme

- Forme de la pièce limitée

Pièce rectangulaire à plusieurs détecteur

Elements pouvant être améliorés

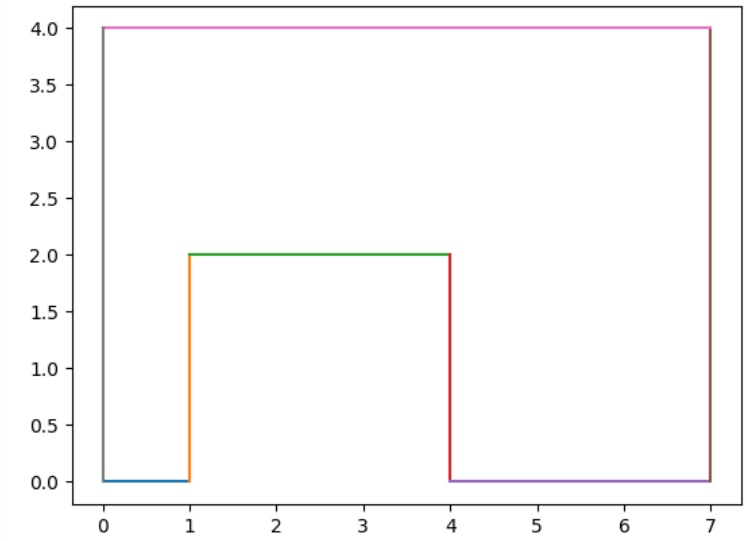
- Hypothèses de déplacement de la fumée :

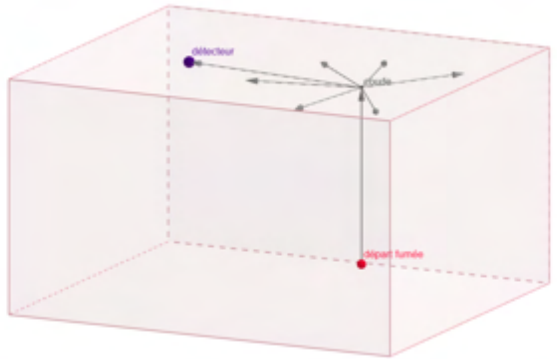
  Déplacement uniforme
- Forme de la pièce limitée

Pièce quelconque à un détecteur

Hypothèses

- pièce quelconque à angles droits
  - modélisée par ses coins
- déplacement de la fumée homogène
  - vers le plafond puis le long du plafond
  - à vitesse constante : 0.5m/s

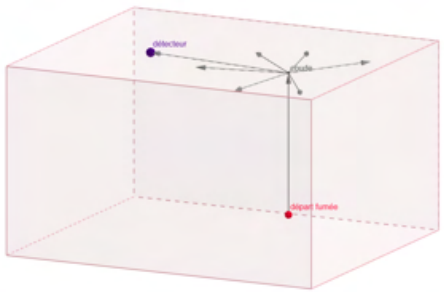


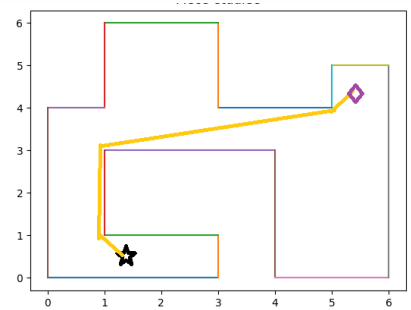


Pièce quelconque à un détecteur

Protocole

- Pour un détecteur donné dans la pièce :
  - Etablissement du trajet de la fumée, et des virages effectués s'il n'est pas direct
  - Détermination du temps mis pour effectuer ce trajet
  - Moyenne de ce temps pour 200 départs de fumée aléatoires

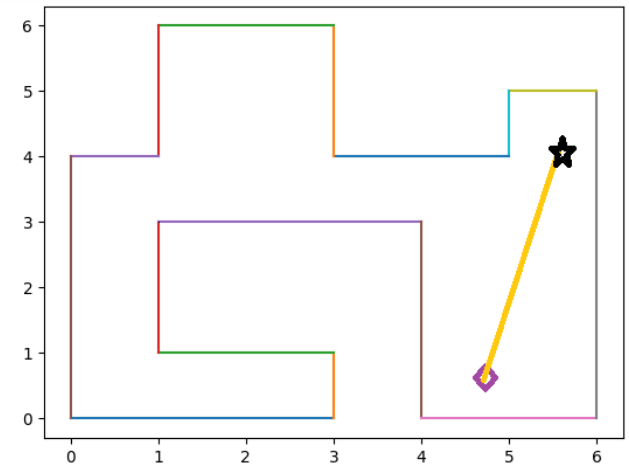


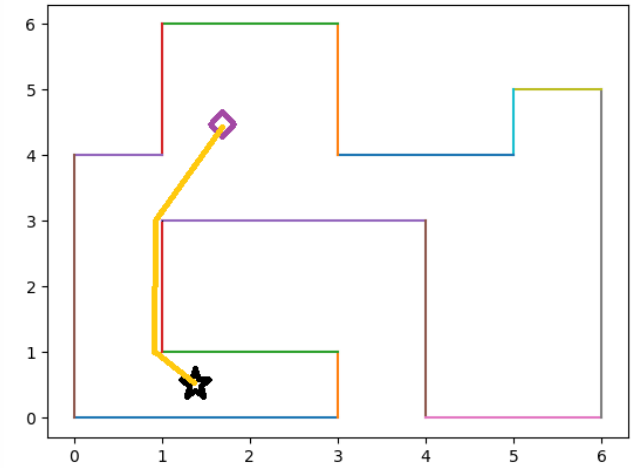


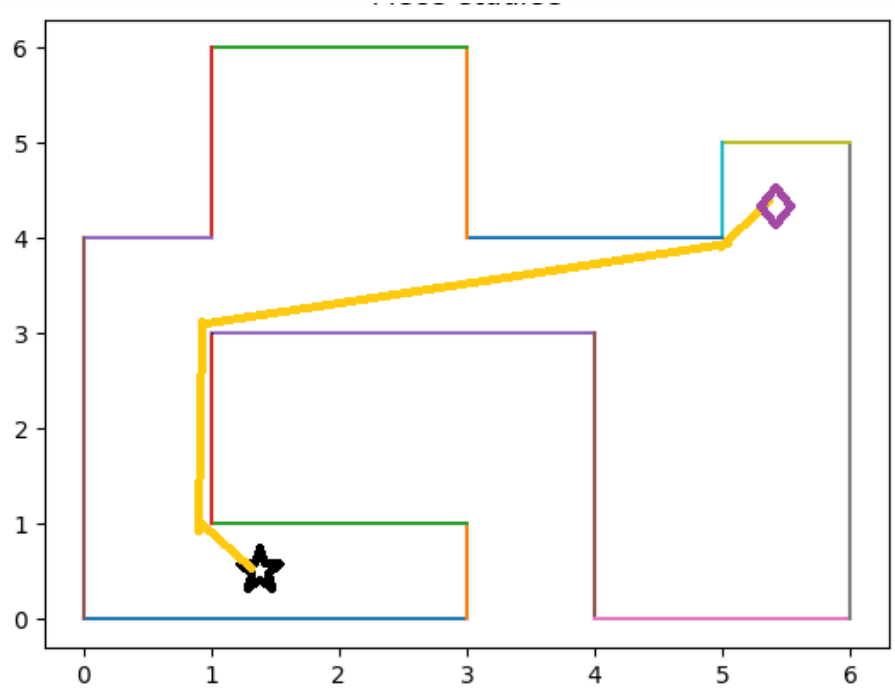
- Idem pour chaque détecteur à l'intérieur de la pièce
- Détermination du détecteur minimisant le temps de détection
- Moyenne des résultats en répétant 5 fois ce processus

Pièce quelconque à un détecteur

Protocole

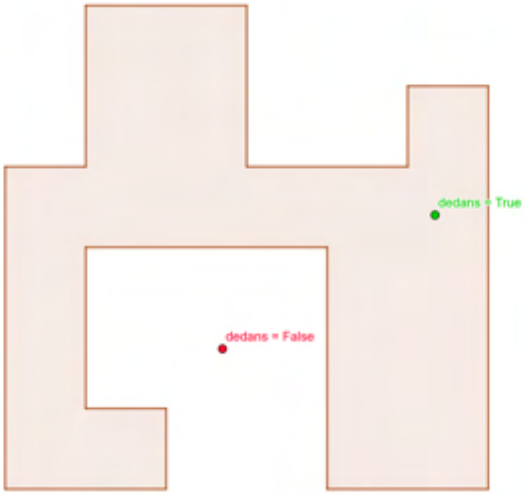




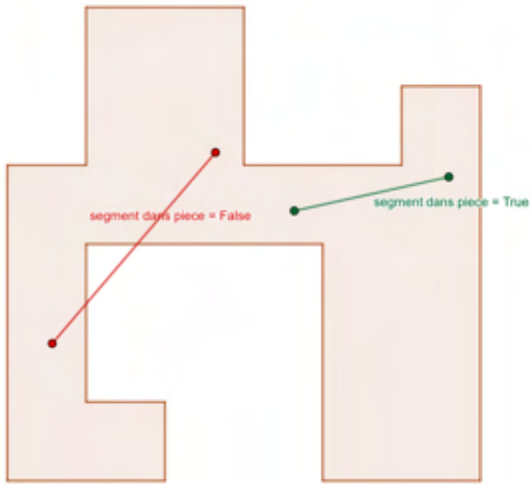


Pièce quelconque à un détecteur

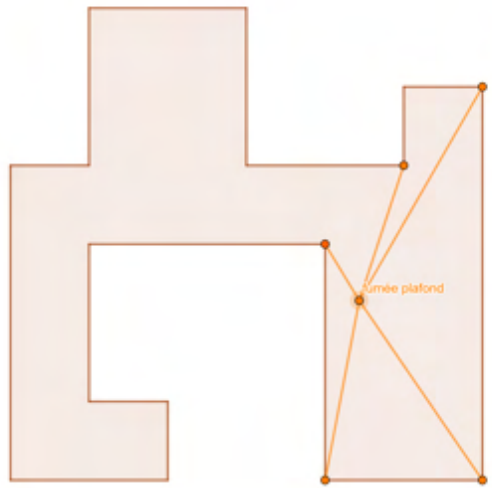
Outils pour déterminer la trajectoire



Point dans la pièce



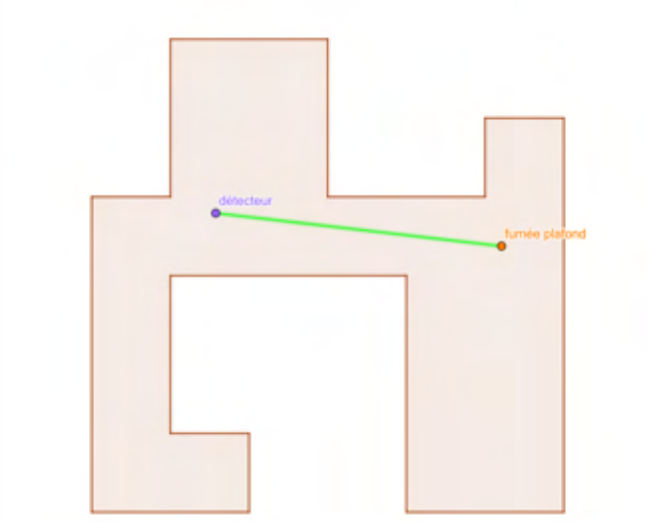
Segment dans la pièce



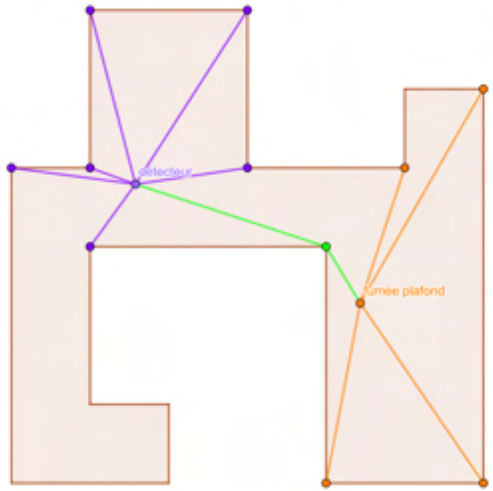
Coins atteignables

Pièce quelconque à un détecteur

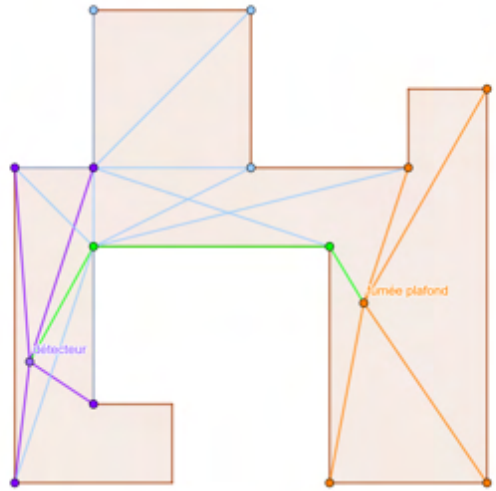
Détermination de la trajectoire



Cas 1 : trajet direct



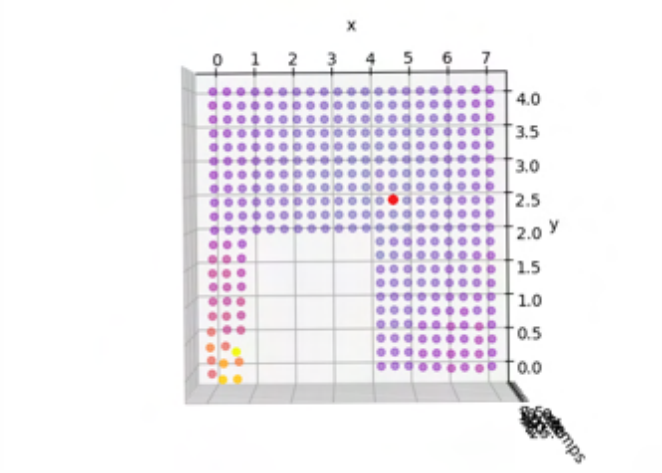
Cas 2 : coins communs



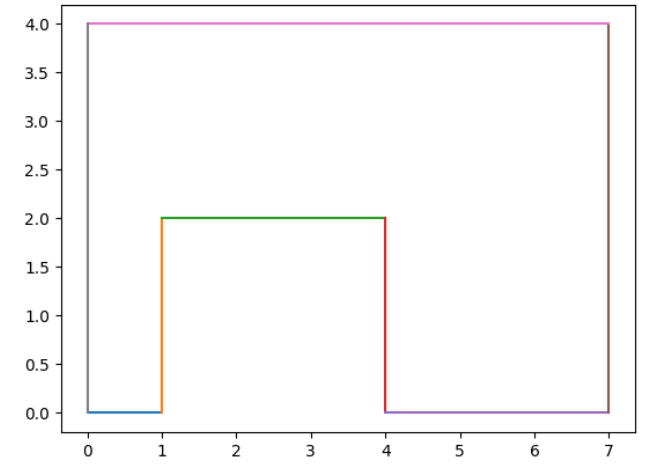
Cas 3 : pas de coins communs

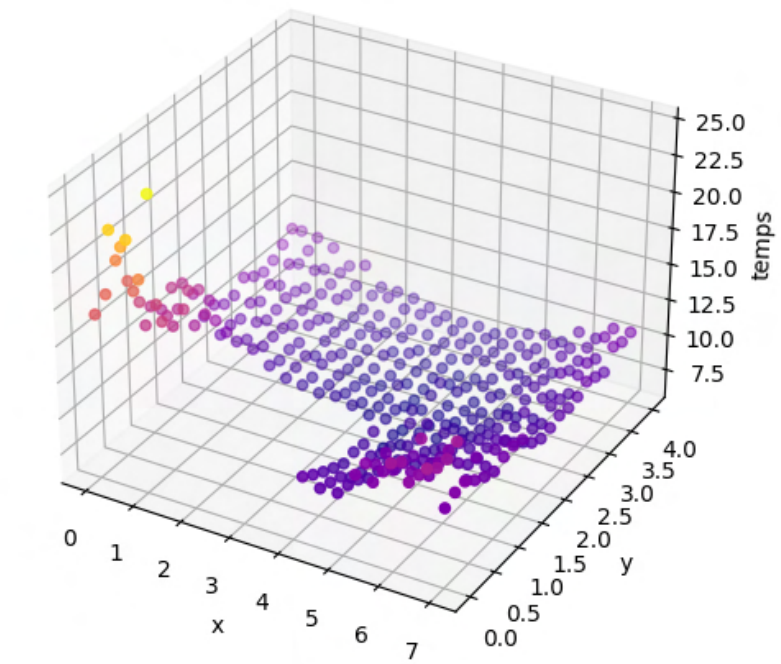
Pièce quelconque à un détecteur

Résultats



Pièce étudiée

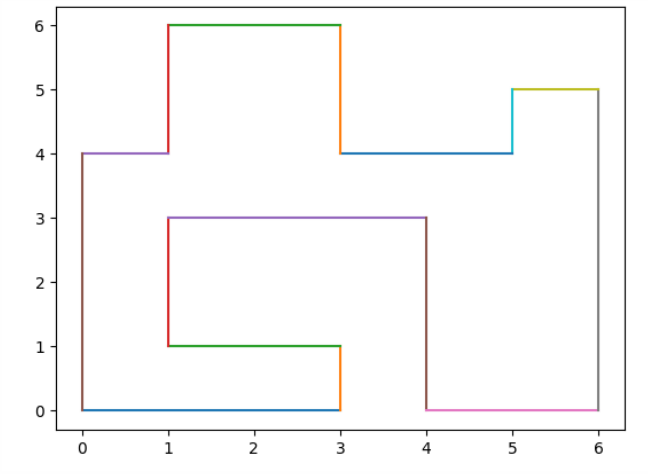




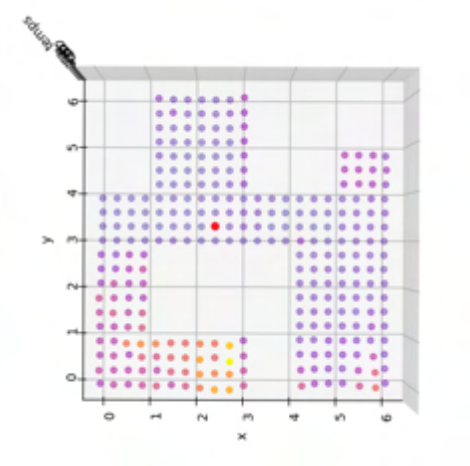
Temps mis par le détecteur à se déclencher selon son emplacement

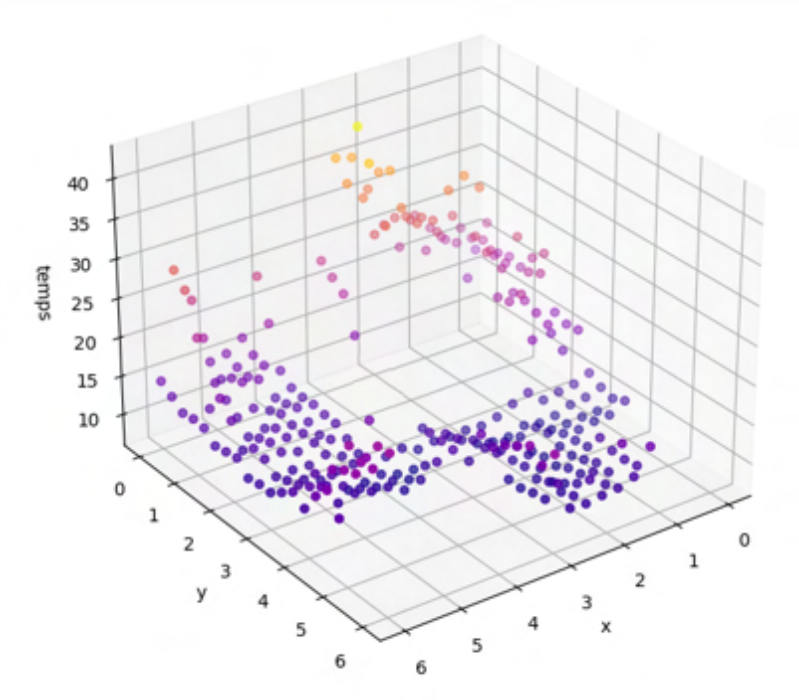
Pièce quelconque à un détecteur

Résultats



Pièce étudiée

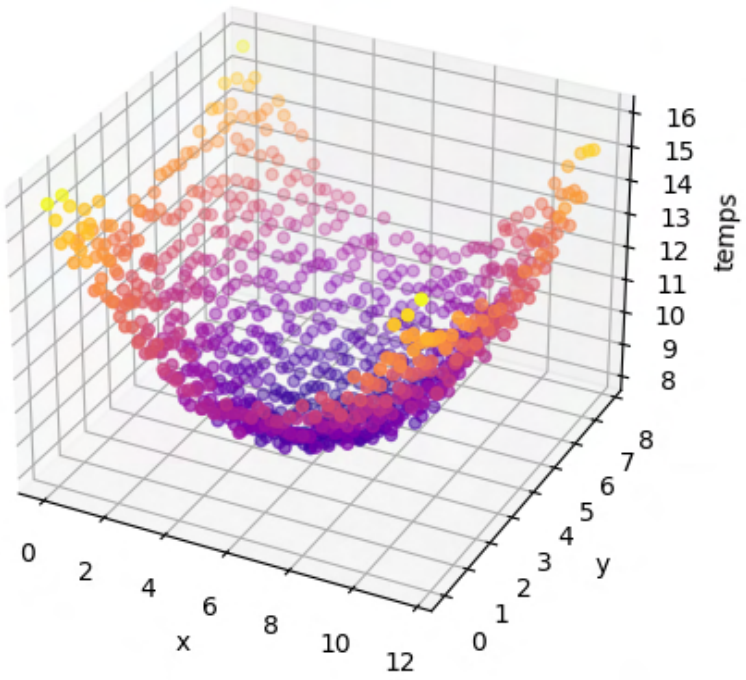




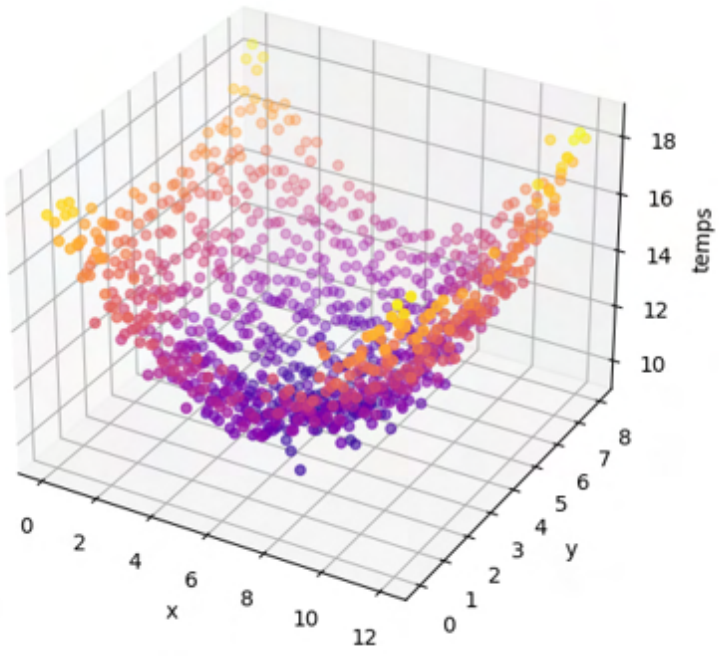
Temps mis par le détecteur à se déclencher selon son emplacement

Pièce quelconque à un détecteur

Cohérence des résultats



Graphique obtenu avec l'algorithme de la partie 2



Graphique obtenu avec cet algorithme

Conclusion

Confirme le bon sens pour des pièces à géométrie simple

Pourrait être amélioré avec un déplacement de la fumée plus réaliste

Contraintes physiques de la pièce + normes vagues

Bon indicateur pour des pièces à géométrie plus complexe

Conclusion

Confirme le bon sens pour des pièces à géométrie simple

Pourrait être amélioré avec un déplacement de la fumée plus réaliste

Contraintes physiques de la pièce + normes vagues

Bon indicateur pour des pièces à géométrie plus complexe

Conclusion

Confirme le bon sens pour des pièces à géométrie simple

Pourrait être amélioré avec un déplacement de la fumée plus réaliste

Contraintes physiques de la pièce + normes vagues

Bon indicateur pour des pièces à géométrie plus complexe

Conclusion

Confirme le bon sens pour des pièces à géométrie simple

Pourrait être amélioré avec un déplacement de la fumée plus réaliste

Contraintes physiques de la pièce + normes vagues

Bon indicateur pour des pièces à géométrie plus complexe

Annexe

Code : Pièce rectangulaire à 1 détecteur

Hypothèses

```python
import numpy as np
import random as rd
import matplotlib.pyplot as plt
import matplotlib as mp

# Pièce rectangulaire à 1 détecteur, déplacement de la fumée homogène

longueur1 = 12
largeur1 = 4
hauteur = 2.5


def piece_rect(long, larg, haut):
    surf = long * larg
    volume = surf * hauteur
    return [long, larg, haut, surf, volume]


def distance_points(pt1, pt2):  #points à 3 coordonnées
    return np.sqrt((abs(pt1[0] - pt2[0])) ** 2 + (abs(pt1[1] - pt2[1])) ** 2 + (abs(pt1[2] - pt2[2])) ** 2)


def depart_fumee(piece):
    """renvoie des coordonnées aléatoires dans la pièce"""
    x = rd.random() * piece[0]  # longueur
    y = rd.random() * piece[1]  # largeur
    z = rd.random() * hauteur  # z=/=hauteur, prend en compte n'importe quel endroit de la pièce
    return [x, y, z]


vitesse_fumee = 0.5  # m/s

```

Annexe

Code : Pièce rectangulaire à 1 détecteur

Calculs et graphique

```python
def tps_fumee_homogene_detect_donne(piece, detecteur):
    """renvoie le temps mis par un detecteur à se déclencher pour un départ aléatoire de fumée"""
    v = vitesse_fumee
    fumee = depart_fumee(piece)  # coordonnées
    temps = (distance_points(detecteur, fumee)) / v
    return temps


def moyenne_fumeehomogene(piece, detecteur):
    """moyenne pour un detecteur avec départs fumée aléatoires"""
    somme = 0
    for _ in range(200):
        somme += tps_fumee_homogene_detect_donne(piece, detecteur)
    return (somme / 200)


def resultats_fumee_homogene(piece):
    """graphique des moyennes du temps mis par les détecteurs dans chaque point de la pièce"""
    fig = plt.figure()  # création figure
    ax = plt.axes(projection='3d')
    ax.set_ylim3d([0, 12])
    ax.set_xlim3d([0, 12])
    x = []
    y = []
    z = []
    pas_x = piece[0] / 30
    pas_y = piece[1] / 30

    for i in range(30):
        for j in range(30):
            x.append(i * pas_x)
            y.append(j * pas_y)
            detecteur = [i * pas_x, j * pas_y, piece[2]]
            z.append(moyenne_fumeehomogene(piece, detecteur))

    ax.set_title("Temps mis par le détecteur à se déclencher")
    ax.set_zlabel("temps")
    ax.set_xlabel("x")
    ax.set_ylabel("y")

    points = ax.scatter(x, y, z, c=z, cmap='plasma', norm=mp.colors.Normalize())
    plt.show()

```

Annexe

Code : Pièce rectangulaire à 1 détecteur

Calculs

```
78 def temps_minimise(piece):
79     """ pr des détecteurs sur chaque point de la pièce,en quelles coordonnées le tps de réponse est minimal et
   quel est ce temps de réponse ?"""
80     x = []
81     y = []
82     z = []
83     pas_x = piece[0] / 30
84     pas_y = piece[1] / 30
85 
86     xmin, ymin = 0, 0
87     tmin = moyenne_fumeehomogene(piece, [0, 0, piece[2]])  # détecteur dans le coin
88 
89     for i in range(30):
90         for j in range(30):
91             x.append(i * pas_x)
92             y.append(j * pas_y)
93             detecteur = [i * pas_x, j * pas_y, piece[2]]
94             m = moyenne_fumeehomogene(piece, detecteur)
95             z.append(m)
96             if m < tmin:
97                 tmin = m
98                 xmin, ymin = detecteur[0], detecteur[1]
99 
100    return [xmin, ymin, tmin]
101 
102 
103 def moyenne_temps_minimise(piece):
104    """Moyenne des résultats de la fonction précédente"""
105    x, y = 0, 0
106    t = 0
107    for _ in range(30):
108        coord = temps_minimise(piece)
109        x += coord[0]
110        y += coord[1]
111        t += coord[2]
112    return [x / 30, y / 30, t / 30]
113 
```

Annexe

Code : Pièce rectangulaire à 1 détecteur

Résultats

```
115 #Affichage des résultats
116 piece_rectangulaire1 = piece_rect(longueur1, largeur1, hauteur)
117 print("piece =", piece_rectangulaire1)
118
119 # Schéma de la pièce
120 plt.plot([0, piece_rectangulaire1[0]], [0, 0])
121 plt.plot([0, 0], [0, piece_rectangulaire1[1]])
122 plt.plot([piece_rectangulaire1[0], piece_rectangulaire1[0]], [0, piece_rectangulaire1[1]])
123 plt.plot([0, piece_rectangulaire1[0]], [piece_rectangulaire1[1], piece_rectangulaire1[1]])
124 plt.show()
125
126 # Graphique
127 print(resultats_fumee_homogene(piece_rectangulaire1))
128
129 # Résultats
130 print("Pour 1 test, temps_minimise renvoie", temps_minimise(piece_rectangulaire1))
131 temps_min = moyenne_temps_minimise(piece_rectangulaire1)
132 print("En moyenne, le detecteur situe en", (temps_min[0], temps_min[1]), "est le plus rapide et se declenche
    en", temps_min[2], "s")
133
```

Annexe

Code : Pièce rectangulaire à plusieurs détecteurs

Modélisation

```
136 # Pièce rectangulaire à plusieurs détecteurs, déplacement de la fumée homogène
137
138 longueur2 = 14
139 largeur2 = 14
140 hauteur = 2.5
141
142
143 def piece_rect(longueur, largeur, hauteur):
144     surf = longueur * largeur
145     volume = surf * hauteur
146     return [longueur, largeur, hauteur, surf, volume]
147
148
149 def nbe_detecteurs(piece):  # surface en m2
150     if piece[3] < 50:
151         return 1
152     else:
153         return piece[3] // 50
154
155
156 distance_mini = 3  # distance minimale entre 2 détecteurs (en m)
157
158
159 def detecteurs(piece):
160     """place des détecteurs aléatoirement dans la pièce"""
161     n = nbe_detecteurs(piece)
162     detect = []
163     detect.append([rd.random() * piece[0], rd.random() * piece[1], piece[2]])
164     while len(detect) < n:
165         emplacement = [rd.random() * piece[0], rd.random() * piece[1], piece[2]]
166         distance_suffisante = True
167         for i in detect:
168             if distance_points(i, emplacement) < distance_mini:
169                 distance_suffisante = False
170                 break
171         if distance_suffisante:
172             detect.append(emplacement)
173     return detect
174
```

Annexe

Code : Pièce rectangulaire à plusieurs détecteurs

Position des détecteurs optimaux

```
176 def tps_detects_donnes(detecteurs, piece):
177     """renvoie le temps moyen mis par un ensemble de détecteurs pour se déclencher pour 1000 départs d
178     somme = 0
179     nbe_departs_fumee = 1000
180     for _ in range(nbe_departs_fumee):  # 1000 fumées aléatoires
181         fumee = depart_fumee(piece)
182         tmin = distance_points(detecteurs[0], fumee)
183         for det in detecteurs:  # temps mis par le détecteur de la pièce qui se déclenche le + vite
184             t = (distance_points(det, fumee))
185             if t < tmin:
186                 tmin = t
187         somme += tmin  # moyenne
188     return somme / nbe_departs_fumee
189
190
191 def detecteurs_opti(piece):
192     """tests pour 5000 ensembles aléatoires de détecteurs, lequel est le plus optimal"""
193     det_min = detecteurs(piece)
194     tmin = tps_detects_donnes(det_min, piece)
195     nbe_essais = 5000
196     for _ in range(nbe_essais):
197         det = detecteurs(piece)
198         t = tps_detects_donnes(det, piece)
199         if t < tmin:
200             tmin = t
201             det_min = det
202     return det_min
203
```

Annexe

Code : Pièce rectangulaire à plusieurs détecteurs

Position des détecteurs optimaux

```
205 def graphique_emplacement_detecteurs(piece):
206     """affiche 1 emplacement optimal des détecteurs dans la pièce"""
207     L = detecteurs_opti(piece)
208     for x in L:
209         plt.plot(x[0], x[1], marker="o")
210     plt.xlim(0, piece[0])
211     plt.ylim(0, piece[1])
212     plt.show()
213     return L
214 
215 
216 def graphique_emplacements_detecteurs_superposes(piece):
217     """affiche 10 emplacements optimaux des détecteurs pour vérifier la concordance des résultats"""
218     for _ in range(10):
219         L = detecteurs_opti(piece)
220         couleur = (rd.random(), rd.random(), rd.random())
221         for det in L:
222             plt.plot(det[0], det[1], marker="o", color=couleur)
223     plt.xlim(0, piece[0])
224     plt.ylim(0, piece[1])
225     plt.show()
226 
```

Annexe

Code : Pièce rectangulaire à plusieurs détecteurs

Temps de détection pour un départ de fumée donné

```
228 def temps_det_et_fumee_donnes(piece, detecteurs, fumee):
229     """renvoie le temps mis par les détecteurs pour se déclencher pour un départ de fumée donné"""
230     tmin = distance_points(detecteurs[0], fumee)
231     for det in detecteurs:  # temps mis par le détecteur de la pièce qui se déclenche le + vite
232         t = (distance_points(det, fumee))
233         if t < tmin:
234             tmin = t
235     return tmin
236
237
238 def graphique_temps_detection(piece):
239     """affiche la moyenne du temps mis par des détecteurs optimaux selon l'emplacement du départ de fumée dans la
    pièce"""
240     detecteurs = detecteurs_opti(piece)
241     fig = plt.figure()  # création figure
242     ax = fig.add_subplot(projection='3d')  # création de la 3d
243
244     x = []
245     y = []
246     z = []
247     pas_x = piece[0] / 30
248     pas_y = piece[1] / 30
249
250     for i in range(30):
251         for j in range(30):
252             x.append(i * pas_x)
253             y.append(j * pas_y)
254             fumee = [i * pas_x, j * pas_y, 0]
255             z.append(temps_det_et_fumee_donnes(piece, detecteurs, fumee))
256
257     ax.set_title("Temps mis par les détecteurs optimaux à se déclencher selon l'emplacement du départ de fumée")
258     ax.set_zlabel("temps")
259     ax.set_xlabel("x")
260     ax.set_ylabel("y")
261
262     points = ax.scatter(x, y, z, c=z, cmap='viridis', norm=mp.colors.Normalize())
263     plt.show()
264
```

Annexe

Code : Pièce rectangulaire à plusieurs détecteurs

Comparaison et résultats

```python
def graphique_temps_detection_pas_opti(piece):
    """affiche la moyenne du temps mis par des détecteurs aléatoires selon l'emplacement du départ de fumée dans la pièce"""
    detects = detecteurs(piece)
    fig = plt.figure()  # création figure
    ax = fig.add_subplot(projection='3d')  # création de la 3d

    x = []
    y = []
    z = []
    pas_x = piece[0] / 30
    pas_y = piece[1] / 30

    for i in range(30):
        for j in range(30):
            x.append(i * pas_x)
            y.append(j * pas_y)
            fumee = [i * pas_x, j * pas_y, 0]
            z.append(temps_det_et_fumee_donnes(piece, detects, fumee))

    ax.set_title(
        "Temps mis par des détecteurs placés aléatoirement à se déclencher selon l'emplacement du départ de fumée")
    ax.set_zlabel("temps")
    ax.set_xlabel("x")
    ax.set_ylabel("y")

    points = ax.scatter(x, y, z, c=z, cmap='cividis', norm=mp.colors.Normalize())
    plt.show()


# Affichage des résultats
piece_rectangulaire2 = piece_rect(longueur2, largeur2, hauteur)
print("Piece =", piece_rectangulaire2)
print("nombre de detecteurs =", nbe_detecteurs(piece_rectangulaire2))

# schéma de la pièce
plt.plot([0, piece_rectangulaire2[0]], [0, 0])
plt.plot([0, 0], [0, piece_rectangulaire2[1]])
plt.plot([piece_rectangulaire2[0], piece_rectangulaire2[0]], [0, piece_rectangulaire2[1]])
plt.plot([0, piece_rectangulaire2[0]], [piece_rectangulaire2[1], piece_rectangulaire2[1]])

# Résultats
print("Un emplacement optimal des détecteurs serait", detecteurs_opti(piece_rectangulaire2))
print(graphique_emplacements_detecteurs_superposes(piece_rectangulaire2))
print(graphique_temps_detection(piece_rectangulaire2))
print(graphique_temps_detection_pas_opti(piece_rectangulaire2))
```

Annexe

Code : Pièce à géométrie quelconque

Modélisation

```
312 # Pièce à géométrie quelconque
313
314 # coin = (abscisse, ordonnée)
315 coins1 = [(0, 0), (1, 0), (1, 2), (4, 2), (4, 0), (7, 0), (7, 4), (0, 4)]  # dans l'ordre
316 coins2 = [(0, 1), (2, 1), (2, 0), (7, 0), (7, 5), (1, 5), (1, 4), (0, 4)]
317 coins3 = [(0, 0), (2, 0), (2, 1), (1, 1), (1, 3), (4, 3), (4, 0), (6, 0), (6, 5), (5, 5), (5, 4), (3, 4), (3,
    6),
318           (1, 6), (1, 4), (0, 4)]
319 hauteur = 2.5
320
321
322 def distance_coins(pt1, pt2):
323     """distance entre 2 points à 2 coordonnées"""
324     return np.sqrt((abs(pt1[0] - pt2[0])) ** 2 + (abs(pt1[1] - pt2[1])) ** 2)
325
326
327 def murs_piece(coins):
328     """prend en entrée les coins de la pièce, renvoie des triplets (coin de départ, coin d'arrivée,
    longueur)"""
329     n = len(coins)
330     murs = []
331     for i in range(n - 1):
332         depart = coins[i]
333         arrivee = coins[i + 1]
334         longueur = distance_coins(depart, arrivee)
335         murs.append((depart, arrivee, longueur))
336     murs.append((coins[n - 1], coins[0], distance_coins(coins[n - 1], coins[0])))
337     return murs
338
339
340 murs1 = murs_piece(coins1)
341 murs2 = murs_piece(coins2)
342 murs3 = murs_piece(coins3)
343
344 surface1 = 22
345 surface2 = 32
346 surface3 = 22
347
348 piece_quelconque1 = [coins1, murs1, surface1, hauteur, surface1 * hauteur]
349 piece_quelconque2 = [coins2, murs2, surface2, hauteur, surface2 * hauteur]
350 piece_quelconque3 = [coins3, murs3, surface3, hauteur, surface3 * hauteur]
351
352
353 # Schéma de la pièce
354 def schema_piece(coins):
355     n = len(coins)
356     for i in range(n - 1):
357         plt.plot([coins[i][0], coins[i + 1][0]], [coins[i][1], coins[i + 1][1]])
358     plt.plot([coins[n - 1][0], coins[0][0]], [coins[n - 1][1], coins[0][1]])
359     plt.title("Pièce étudiée")
360     plt.show()
361
```

Annexe

Code : Pièce à géométrie quelconque

Départ de la fumée dans la pièce

```
# Fumée
def dimensions_max(piece):
    """renvoie les coordonnées du rectangle contenant la pièce en entier"""
    xmax = piece[0][0][0]
    ymax = piece[0][0][1]
    for e in piece[0]:
        if e[0] > xmax:
            xmax = e[0]
        if e[1] > ymax:
            ymax = e[1]
    return [xmax, ymax]


def dedans(pt, piece):
    """vérifie si le point se situe dans la pièce ou non"""
    abs_murs = []
    ord_murs = []
    for e in piece[0]:
        if e[0] not in abs_murs:
            abs_murs.append(e[0])
        if e[1] not in ord_murs:
            ord_murs.append(e[1])
    dessous = []
    dessus = []
    gauche = []
    droite = []
    for mur in piece[1]:
        if mur[0][0] == mur[1][0] and (mur[0][1] <= pt[1] <= mur[1][1] or mur[0][1] >= pt[1] >= mur[1][1]):
            if pt[0] <= mur[0][0]:
                droite.append(mur)
            if pt[0] >= mur[0][0]:
                gauche.append(mur)
        if mur[0][1] == mur[1][1] and (mur[0][0] <= pt[0] <= mur[1][0] or mur[0][0] >= pt[0] >= mur[1][0]):
            if pt[1] <= mur[0][1]:
                dessus.append(mur)
            if pt[1] >= mur[0][1]:
                dessous.append(mur)
    cotes = [dessous, dessus, gauche, droite]
    for c in cotes:
        if c == [] or (len(c) % 2 == 0 and (pt[0] not in abs_murs) and (pt[1] not in ord_murs)):
            return False
    return True


def depart_fumee(piece):
    """renvoie des coordonnées aléatoires dans la pièce"""
    dim = dimensions_max(piece)
    alea = [rd.random() * dim[0], rd.random() * dim[1]]
    while not dedans(alea, piece):
        alea = [rd.random() * dim[0], rd.random() * dim[1]]
    alea.append(rd.random() * hauteur)
    return alea

```

Annexe

Code : Pièce à géométrie quelconque

Détermination du trajet

```python
# Détermination du trajet
def segment_dans_piece(point1, point2, piece):  # points à 2 coordonnées, pas 3
    """vérifie si un segment est situé intégralement dans la pièce"""
    n = 300
    pt1 = np.array(point1)
    pt2 = np.array(point2)
    for i in range(n + 1):
        t = i / n
        pt = t * pt1 + (1 - t) * pt2
        if not dedans(pt, piece):
            return False
    return True


def atteindre_coins(pt, piece):
    """renvoie les coins de la pièce atteignables en un segment par un point"""
    coins_atteignables = []
    for c in piece[0]:
        if segment_dans_piece(pt, c, piece) and pt != c:
            coins_atteignables.append(c)
    return coins_atteignables


def coins_et_depart(pt, piece):
    """renvoie une liste de (coin atteignable, point de départ) pour un point donné"""
    coins = atteindre_coins(pt, piece)
    return [(c, pt) for c in coins]


def au_moins_coin_commun_listes(l1, l2):
    """vérifie si deux points ont un coin atteignable en commun en partant des listes de leurs coins atteignables"""
    # avec l1 = coins_et_depart(pt1, piece)
    # et   l2 = coins_et_depart(pt2, piece)
    coins_passage = []
    n2 = len(l2)
    for couple in l1:
        for i in range(n2):
            if l2[i][0] == couple[0]:
                coins_passage.append(couple[0])
    np = len(coins_passage)
    if np >= 1:
        return True
    else:
        return False

```

Annexe

Code : Pièce à géométrie quelconque

Détermination du trajet

```
462 def traj1(pt1, pt2, piece):
463     """prend en entrée la pièce et 2 points du plafond ayant au moins 1 coin en commun et renvoie une liste [pt
    plafond1, coin de passage le plus rapide, pt plafond2]"""
464     coins_atteignables1 = atteindre_coins(pt1, piece)
465     coins_atteignables2 = atteindre_coins(pt2, piece)
466     coins_passage = []
467     for c in coins_atteignables1:
468         if c in coins_atteignables2:
469             coins_passage.append(c)
470     n = len(coins_passage)
471     if n == 1:
472         return [pt1, coins_passage[0], pt2]
473     elif n >= 2:
474         dmin = distance_coins(pt1, coins_passage[0]) + distance_coins(coins_passage[0], pt2)
475         coin = 0
476         for i in range(1, n):
477             if distance_coins(pt1, coins_passage[i]) + distance_coins(coins_passage[i], pt2) < dmin:
478                 dmin = distance_coins(pt1, coins_passage[i]) + distance_coins(coins_passage[i], pt2)
479                 coin = i
480         return [pt1, coins_passage[coin], pt2]
481
482
483 def traj_plus_rapide(l1, l2, piece):
484     """part de listes de coins atteignables et renvoie le coin commun et celui d'avant"""
485     depart_coinpassage_arrivee = []
486     n2 = len(l2)
487     for couple in l1:
488         for i in range(n2):
489             if l2[i][0] == couple[0]:
490                 depart_coinpassage_arrivee.append((couple[1], couple[0], l2[i][1]))
491     dmin = 10 ^ 6
492     traj = depart_coinpassage_arrivee[0]
493     for dca in depart_coinpassage_arrivee:
494         if segment_dans_piece(dca[0], dca[1], piece) and segment_dans_piece(dca[1], dca[2], piece):
495             d = distance_coins(dca[0], dca[1]) + distance_coins(dca[1], dca[2])
496             if d < dmin:
497                 dmin = d
498                 traj = dca
499     # print(traj)
500     return [traj[0], traj[1]]
```

Annexe

Code : Pièce à géométrie quelconque

Détermination du trajet

```
503 def traj3(pt1, pt2, piece):
504     """prend 2 points du plafond et renvoie le trajet entre les deux"""
505     if segment_dans_piece(pt1, pt2, piece):
506         return [pt1, pt2]
507     l1 = coins_et_depart(pt1, piece)
508     l2 = coins_et_depart(pt2, piece)
509     if au_moins_coin_commun_listes(l1, l2):
510         return traj1(pt1, pt2, piece)
511     else:
512         while not au_moins_coin_commun_listes(l1, l2):
513             voisins_des_voisins = []
514             for c in l1:  # on complète l1 avec les voisins des coins d'arrivée
515                 voisins_des_voisins = coins_et_depart(c[0], piece)
516                 for v in voisins_des_voisins:  # on supprime les doublons
517                     if ((v[1], v[0]) not in l1) and ((v[0], v[1]) not in l1):
518                         l1.append(v)
519         trajet = [pt2]
520         while trajet[0] != pt1:
521             trajet = traj_plus_rapide(l1, l2, piece) + trajet
522             for e in l1:
523                 if e[1] == trajet[0]:
524                     l1.remove(e)
525                     l2.append((trajet[0], trajet[1]))
526     n = len(trajet)
527     for i in range(n):
528         for j in range(n):
529             if i >= n - j - 1:
530                 pass
531             else:
532                 if segment_dans_piece(trajet[i], trajet[n - j - 1], piece):
533                     return trajet[:i + 1] + trajet[n - j - 1:]
534     return trajet
```

Annexe

Code : Pièce à géométrie quelconque

Calcul du temps de détection et graphique

```
537 # Temps de détection
538 def temps_detection_fumee_coude(piece, detec):
539     """calcule la distance du trajet parcouru et le temps pour la parcourir"""
540     depart = depart_fumee(piece)
541     distance = distance_points(depart, (depart[0], depart[1], hauteur))  # jusqu'au plafond
542     traj_plafond = traj3((depart[0], depart[1]), (detec[0], detec[1]), piece)
543     n = len(traj_plafond)
544     for i in range(n - 1):
545         distance += distance_coins(traj_plafond[i], traj_plafond[i + 1])
546     return distance / vitesse_fumee
547
548
549 def moyenne_temps_detection_detecteur(piece, detecteur):
550     """ calcule la moyenne du temps de détection pour un détecteur donné avec départs de fumée aléatoires"""
551     somme = 0
552     n = 200
553     for _ in range(n):
554         somme += temps_detection_fumee_coude(piece, detecteur)
555     return (somme / n)
556
557
558 def moyenne_temps_detection_piece(piece):
559     """affiche le graphique du temps moyen mis par les détecteurs dans chaque point de la pièce"""
560     fig = plt.figure()
561     ax = plt.axes(projection='3d')
562     x = []
563     y = []
564     z = []
565     subdiv = 50
566     xmax, ymax = dimensions_max(piece)
567     pas_x = xmax / subdiv
568     pas_y = ymax / subdiv
569
570     for i in range(subdiv + 1):
571         for j in range(subdiv + 1):
572             if dedans((i * pas_x, j * pas_y), piece):
573                 x.append(i * pas_x)
574                 y.append(j * pas_y)
575                 detecteur = [i * pas_x, j * pas_y, piece[3]]
576                 z.append(moyenne_temps_detection_detecteur(piece, detecteur))
577
578     ax.set_title("Temps mis par le détecteur à se déclencher")
579     ax.set_zlabel("temps")
580     ax.set_xlabel("x")
581     ax.set_ylabel("y")
582
583     points = ax.scatter(x, y, z, c=z, cmap='plasma', norm=mp.colors.Normalize())
584     plt.show()
585
```

Annexe

Code : Pièce à géométrie quelconque

Calculs et résultats

```
587 def temps_minimise(piece):
588     """pr des détecteurs en chaque point de la pièce, pour quelles coordonnées on a le tps de réponse minimal
    et quel est ce temps de réponse ?"""
589     x = []
590     y = []
591     z = []
592     subdiv = 50
593     xmax, ymax = dimensions_max(piece)
594     pas_x = xmax / subdiv
595     pas_y = ymax / subdiv
596 
597     xmin, ymin = 0, 0
598     tmin = moyenne_temps_detection_detecteur(piece, piece[0][0])  # détecteur dans le 1er coin
599 
600     for i in range(subdiv + 1):
601         for j in range(subdiv + 1):
602             x.append(i * pas_x)
603             y.append(j * pas_y)
604             if dedans((i * pas_x, j * pas_y), piece):
605                 detecteur = [i * pas_x, j * pas_y, piece[3]]
606                 m = moyenne_temps_detection_detecteur(piece, detecteur)
607                 z.append(m)
608                 if m < tmin:
609                     tmin = m
610                     xmin, ymin = detecteur[0], detecteur[1]
611 
612     return [xmin, ymin, tmin]
613 
614 
615 def moyenne_temps_minimise(piece):
616     """Moyenne des résultats de la fonction précédente"""
617     x, y = 0, 0
618     t = 0
619     repet = 5
620     for _ in range(repet):
621         coord = temps_minimise(piece)
622         x += coord[0]
623         y += coord[1]
624         t += coord[2]
625     return [x / repet, y / repet, t / repet]
626 
627 
628 # Résultats de l'étude :
629 # Pièce :
630 print('Les coins de la piece 3 sont', piece_quelconque3[0], ", ses murs sont", piece_quelconque3[1], ", sa
    surface",
631       piece_quelconque3[2], ", et son volume", piece_quelconque3[3])
632 print(schema_piece(coins3))
633 #Temps de détection et graphique :
634 print(moyenne_temps_detection_piece(piece_quelconque3))
635 print("Pour 1 test, temps_minimise renvoie", temps_minimise(piece_quelconque3))
636 print(moyenne_temps_minimise(piece_quelconque3))
637 
```