Evitement d'un obstacle par une voiture autonome grâce au sonar.

38835
Jules Mignolet

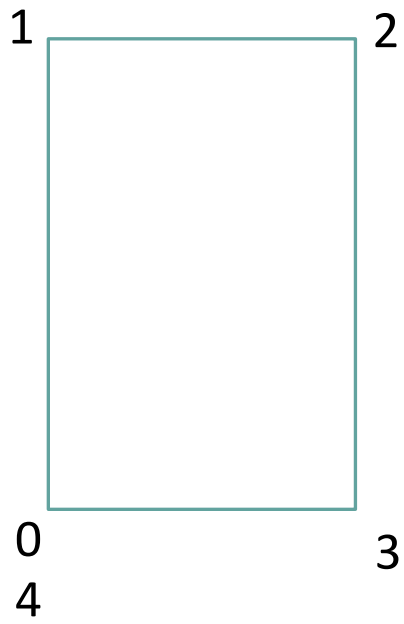
Objectifs du TIPE

- Modéliser une voiture autonome et un parcours d'obstacles.
- Modéliser la prise d'information du sonar.
- Permettre à la voiture de traverser le parcours sans percuter d’obstacles.
- Optimiser la trajectoire du véhicule.

Sommaire

Modélisation de l'environnement

La voiture et les obstacles :



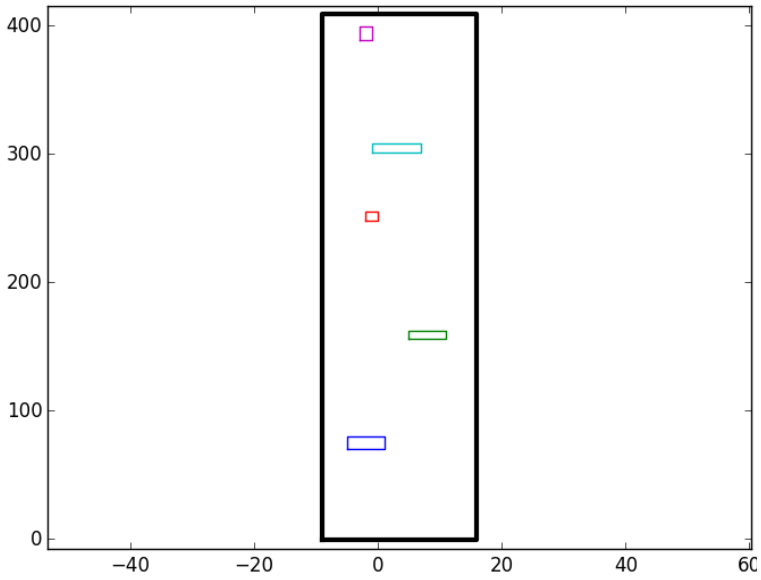
2 listes de 5 éléments représentant les 4 points :

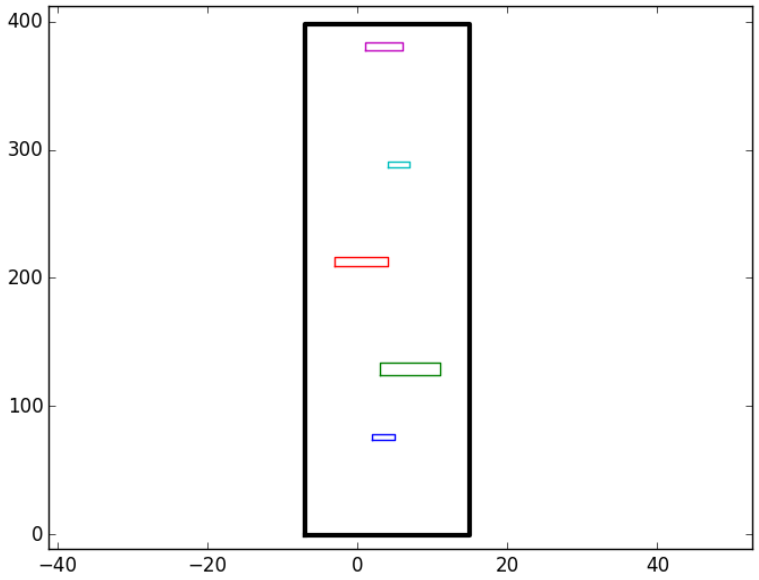
Abscisses = [-1 , -1 , 1 , 1 , -1]

Ordonnées = [-4 , 0 , 0 , -4 , -4]

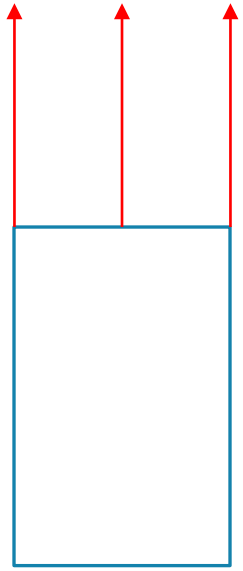
Modélisation de l'environnement

Le parcours





Modélisation du sonar



Principe de l’algorithme

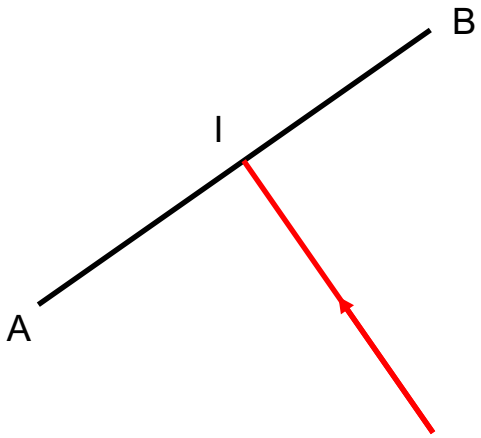
Détecter l’obstacle

Simuler l’évitement pour trouver le meilleur angle d’évitement

Passer à l’obstacle suivant

Principe de l'algorithme

Détecter l'obstacle

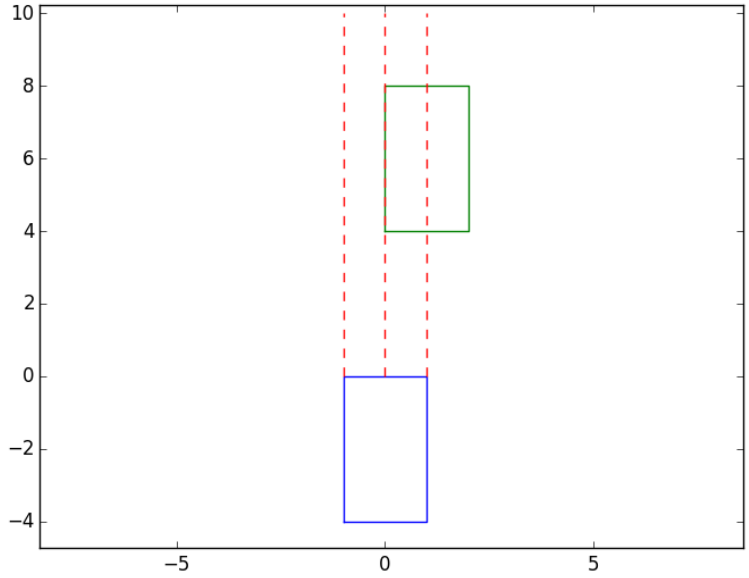


Si $\overrightarrow{IA}.\overrightarrow{IB}<=0$ alors I appartient à AB

Sinon, I n'appartient pas à AB

Principe de l'algorithme

Détecter l'obstacle



Détection Obstacle ⟶ True

Programmes nécessaires

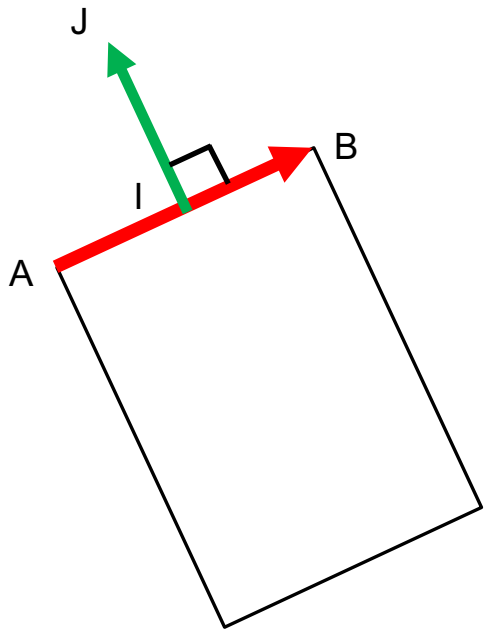
Avancer la voiture

Faire tourner la voiture

Simulation nécessaire à l'évitement de l'obstacle

Avancer la voiture

Direction de la voiture



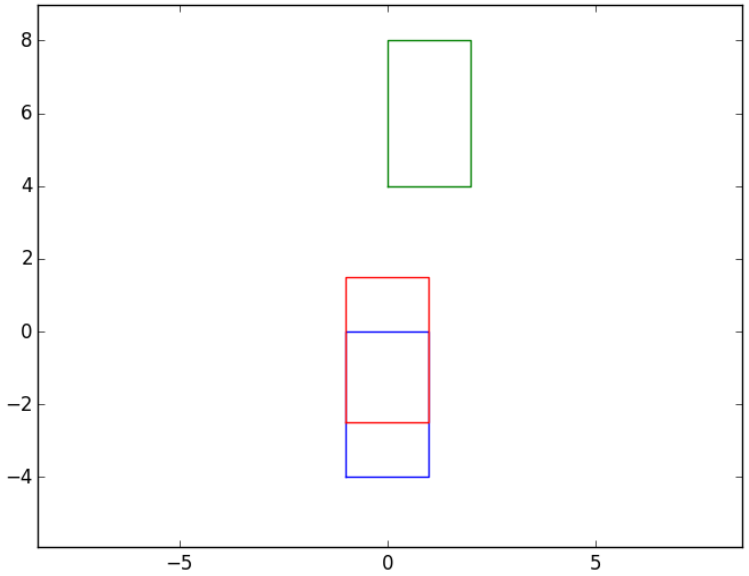
$\overrightarrow{AB} = (a, b)$

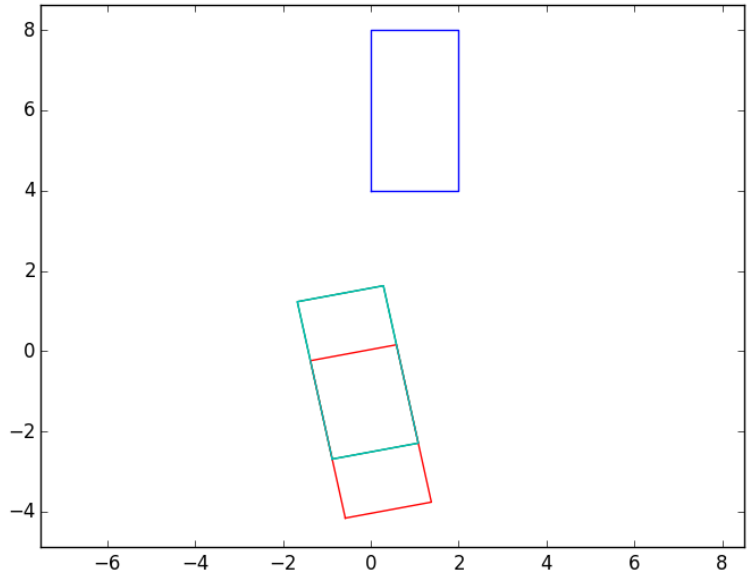
J = I + (-b, a)

$\overrightarrow{IJ}$ donne la direction de la voiture et des sonars

Avancer la voiture

Vitesse = 1.5 (sans unité car c'est le pas du déplacement)



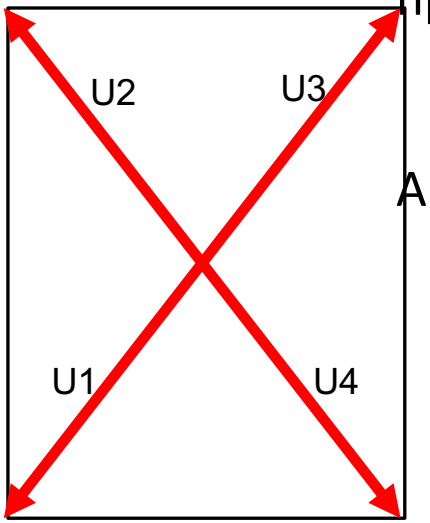


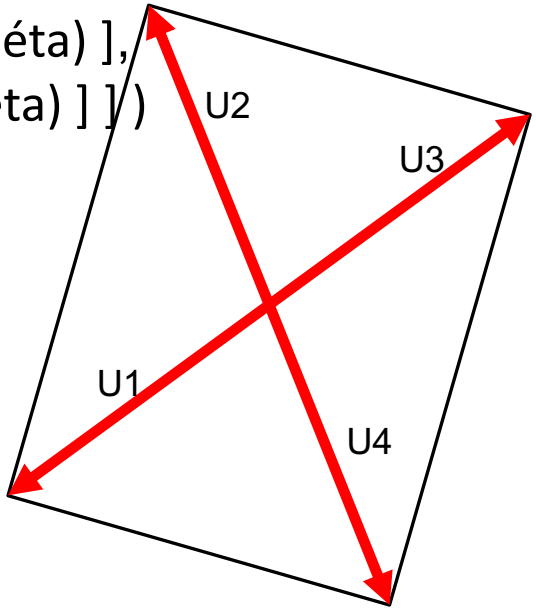
Tourner la voiture

Matrice de Rotation :

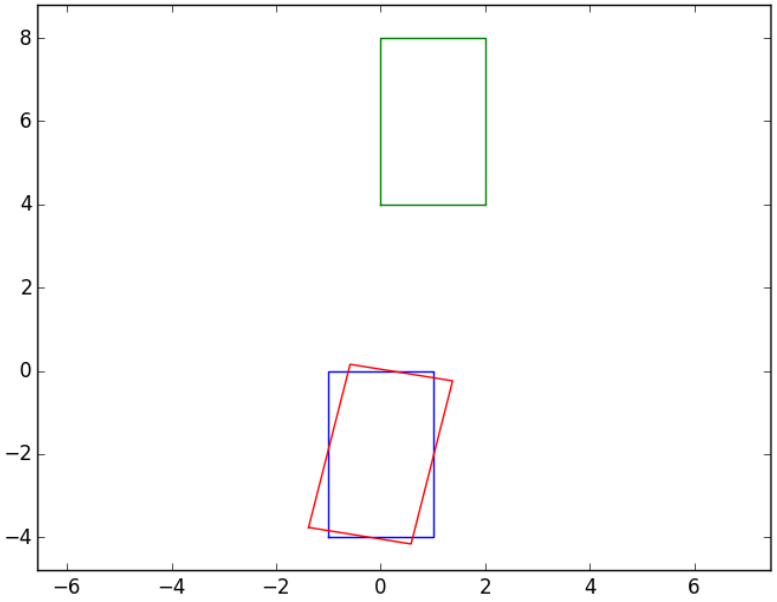
np.array ([[cos (théta) , - sin (théta)],
[sin (théta) , cos (théta)]])

Appliquée aux vecteurs U1 à U4





Tourner la voiture

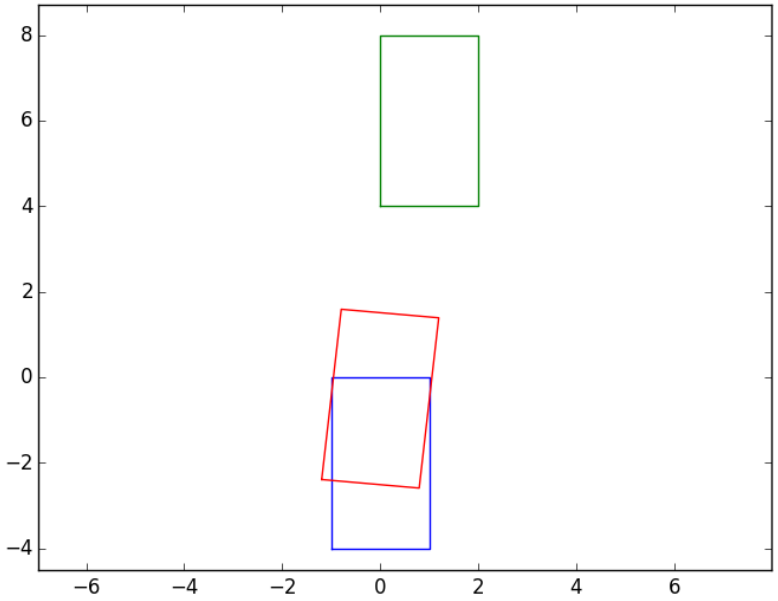


Variables nécessaires :

Position de la voiture

Théta

Avancer et tourner la voiture



Variables nécessaires :

Position de la voiture

Théta

Vitesse

Les simulations

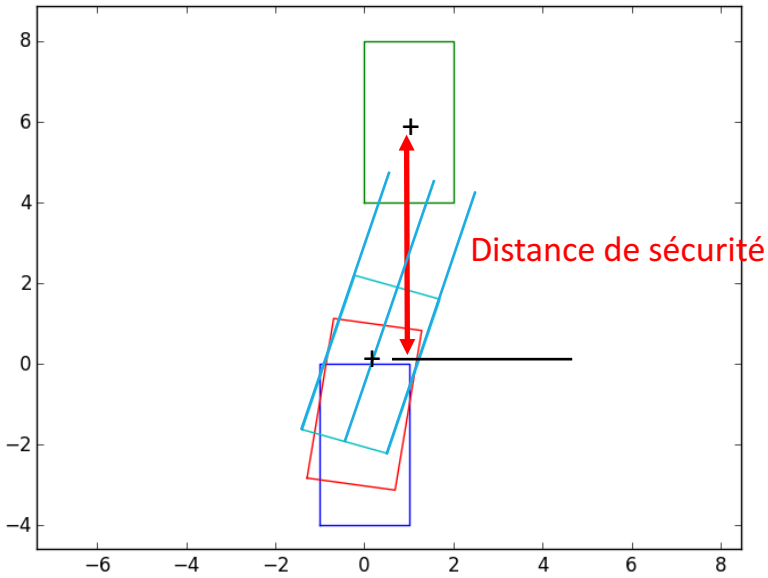
Objectif de la simulation :

- Déterminer l'angle de rotation nécessaire à l'évitement
- Déterminer si la voiture doit tourner à droite ou à gauche
- Indiquer combien de fois la voiture a avancé

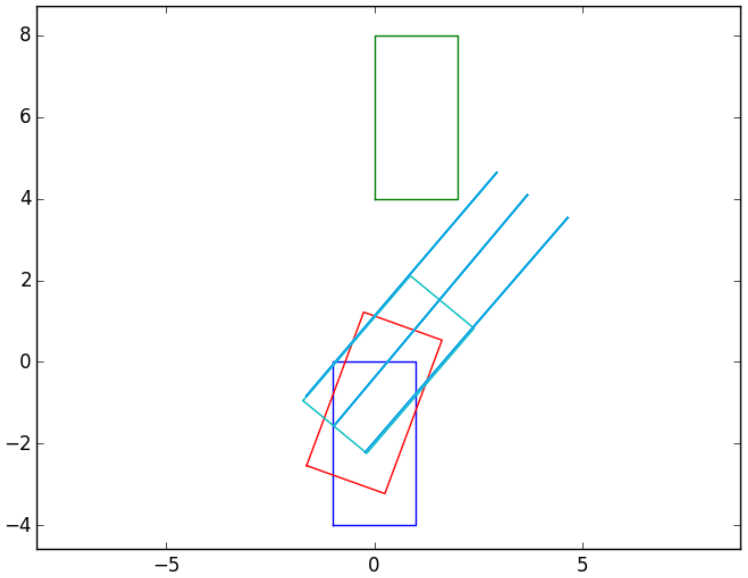
Les simulations

Déterminer l'angle de rotation nécessaire à l'évitement

Théta= - 0.15 rad



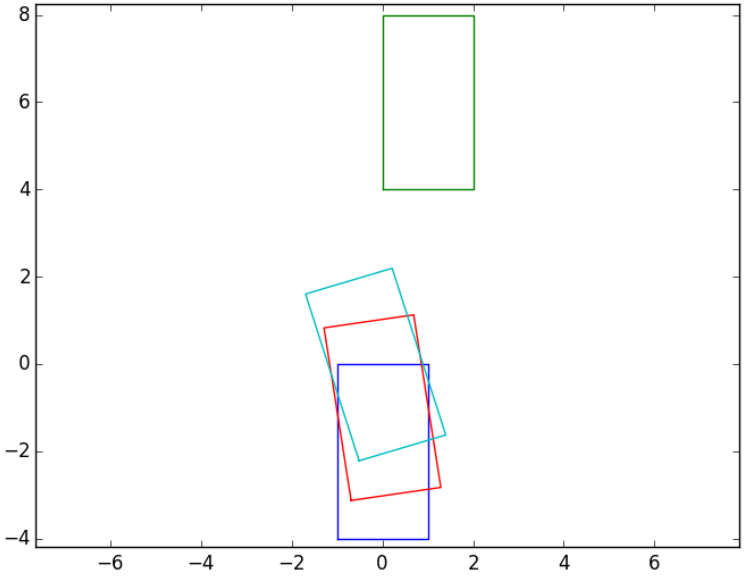
Théta= - 0.35 rad



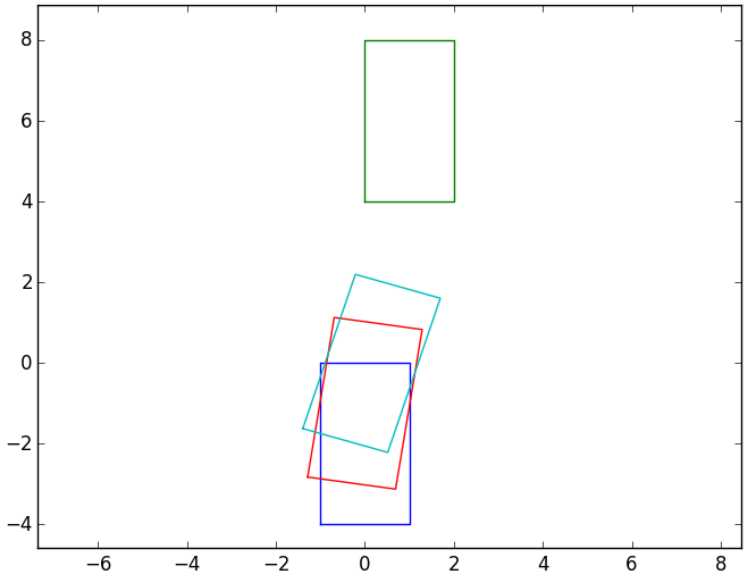
Les simulations

Déterminer si la voiture doit tourner à droite ou à gauche

Théta = 0.15 rad

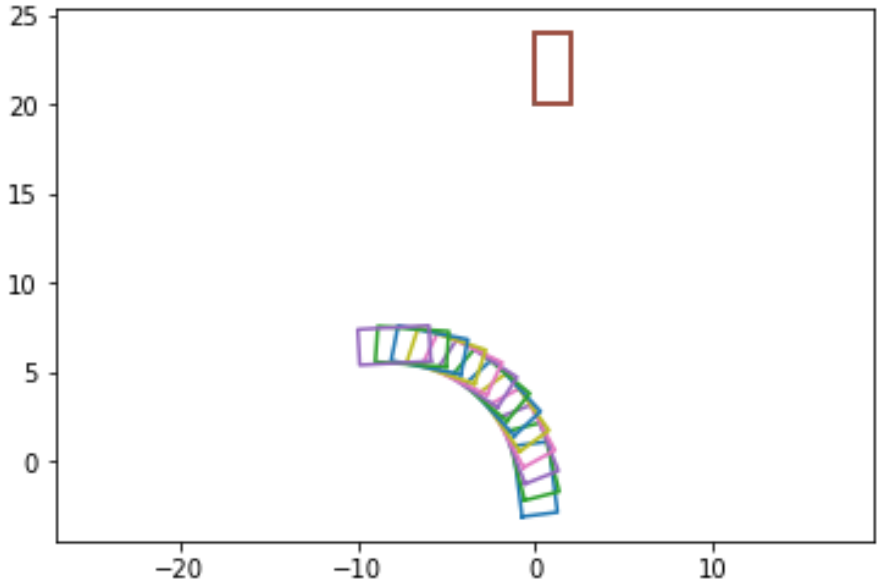


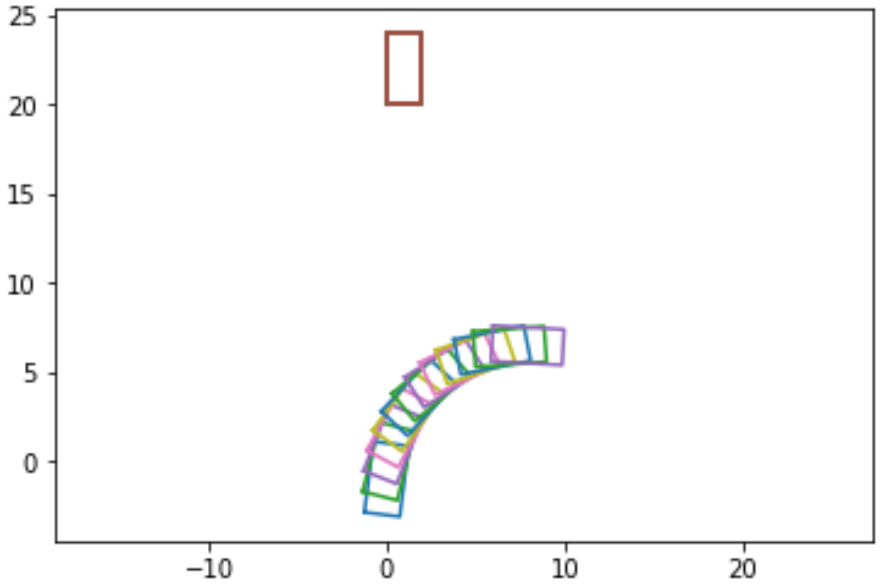
Théta = - 0.15 rad



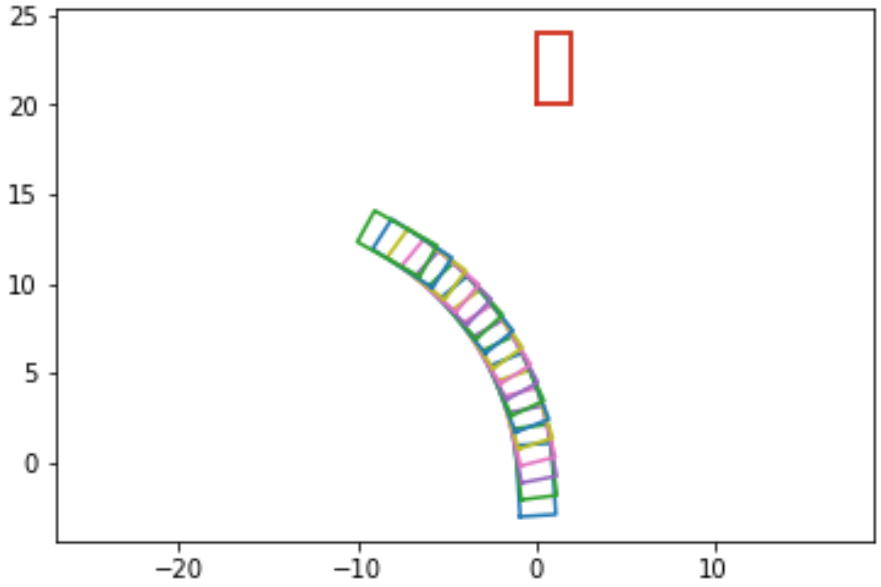
Exemple de simulation précise

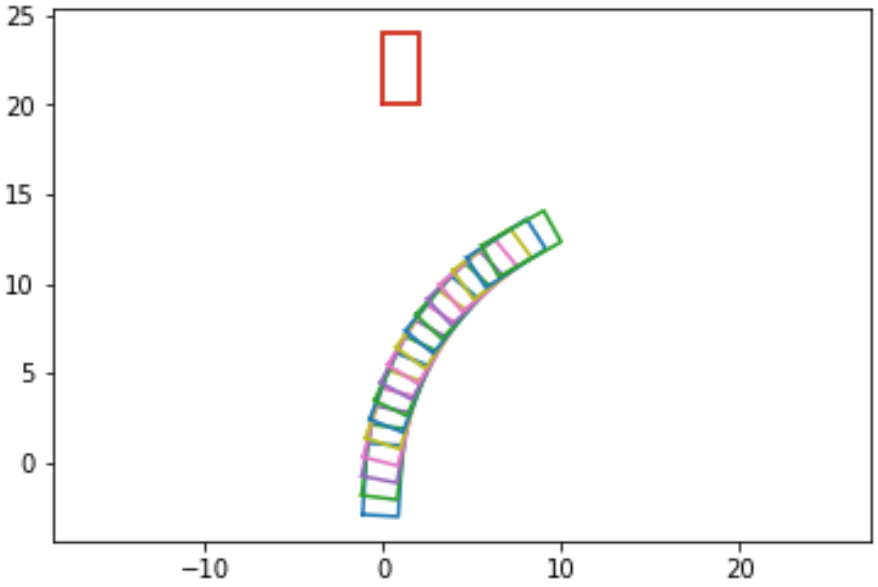
Simulation réalisée par dichotomie sur théta



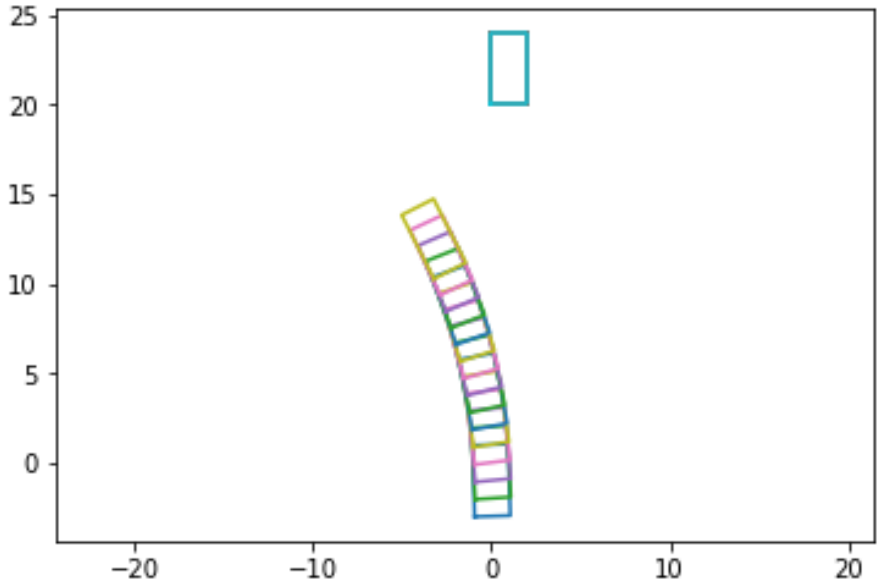


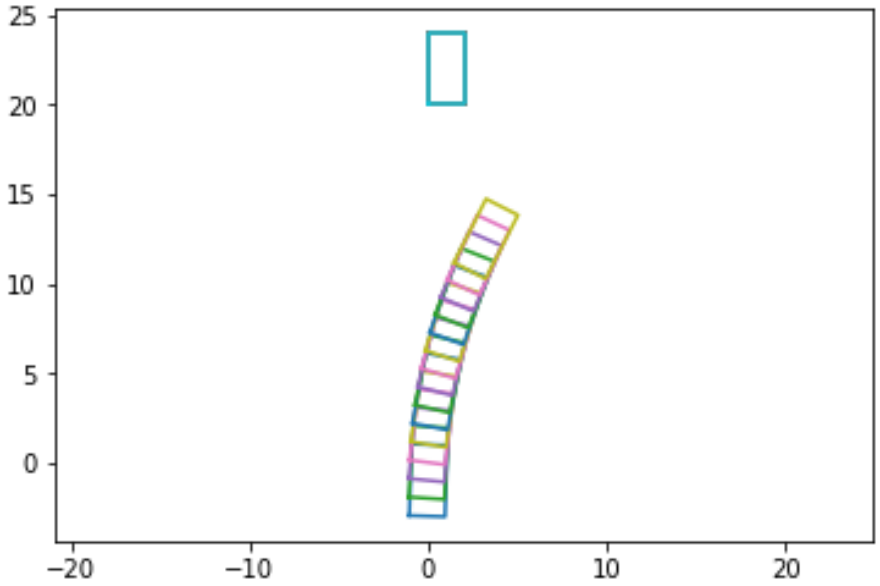
Exemple de simulation précise



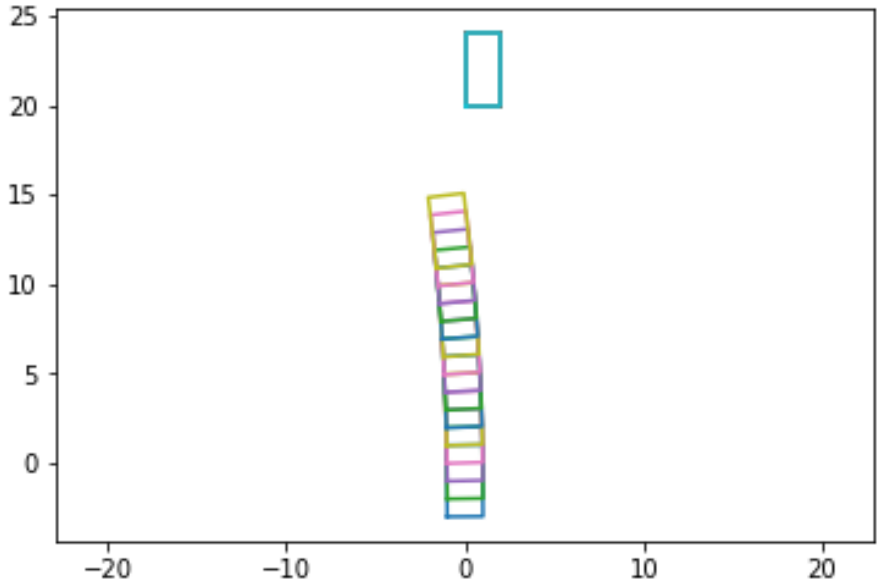


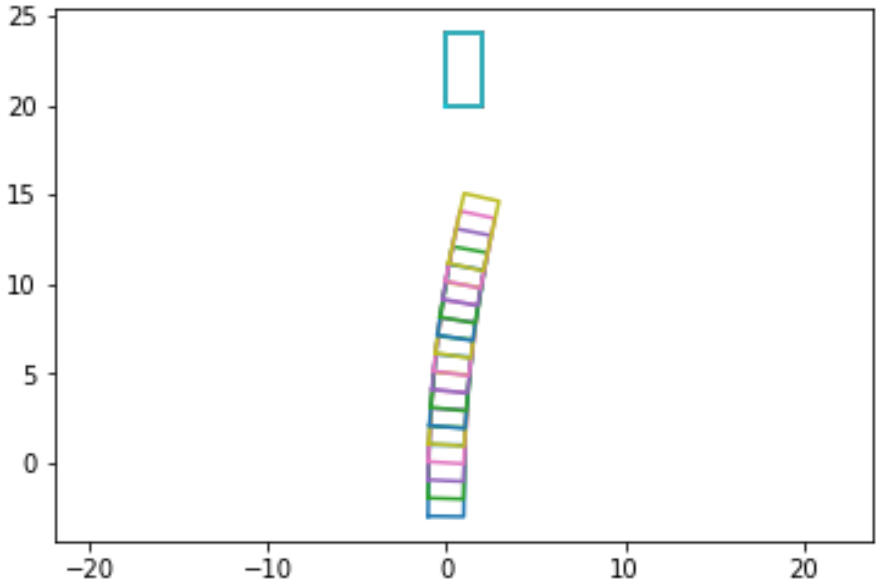
Exemple de simulation précise



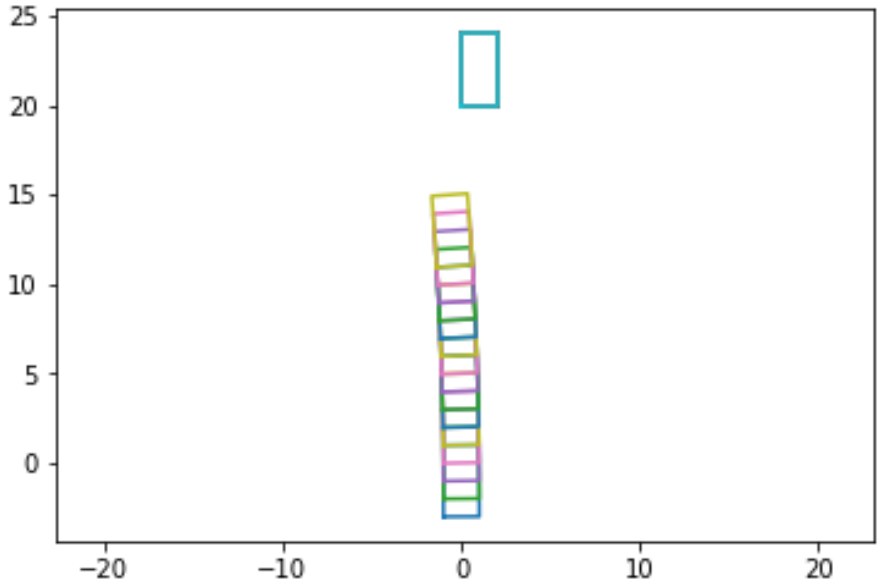


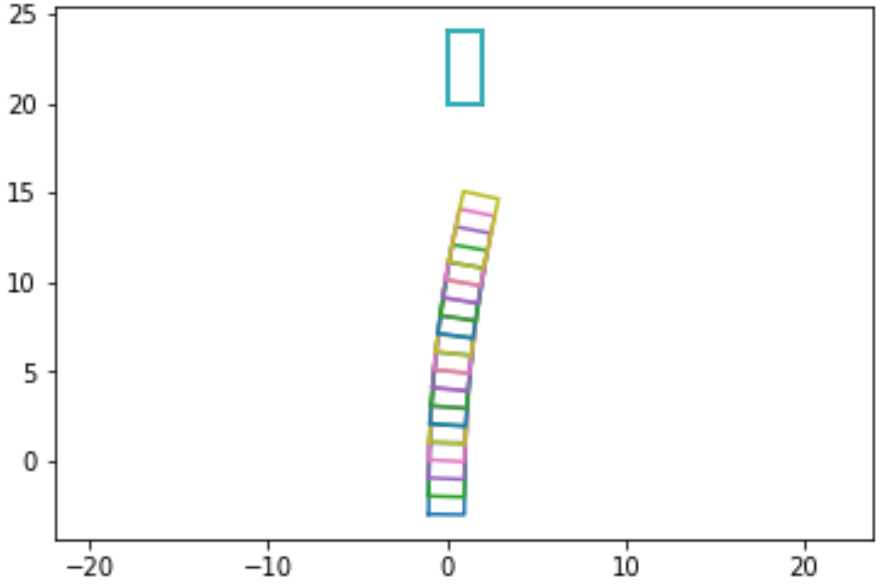
Exemple de simulation précise





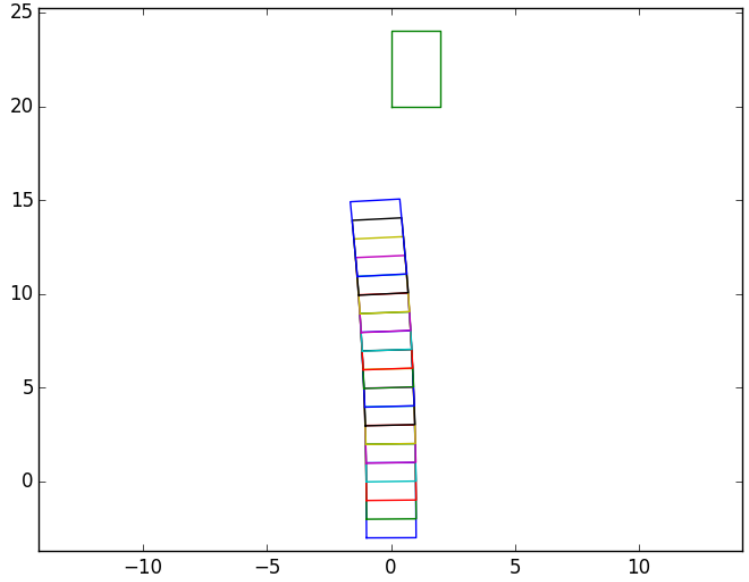
Exemple de simulation précise





Théta obtenu : 0.004791259765625 rad et compteur = 15

Dépasser l’obstacle



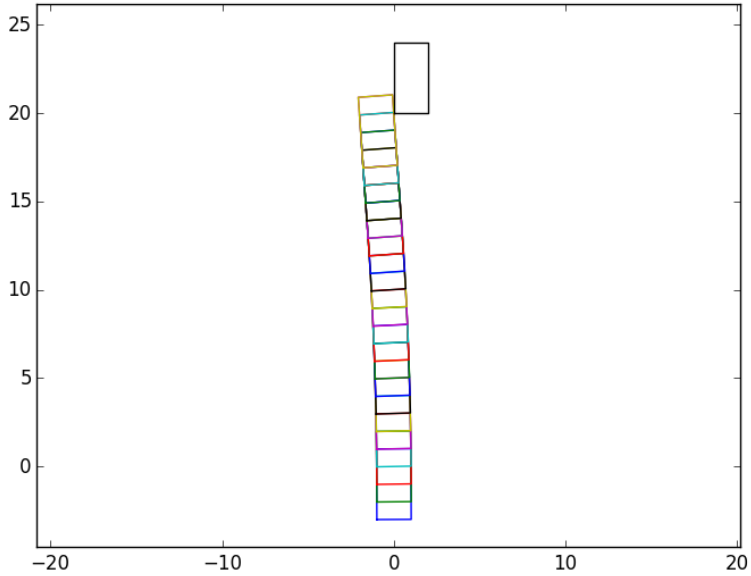
Variables nécessaires :

Position de la voiture et de l’obstacle

Distance de sécurité

Théta et compteur obtenu à la simulation

Dépasser l'obstacle



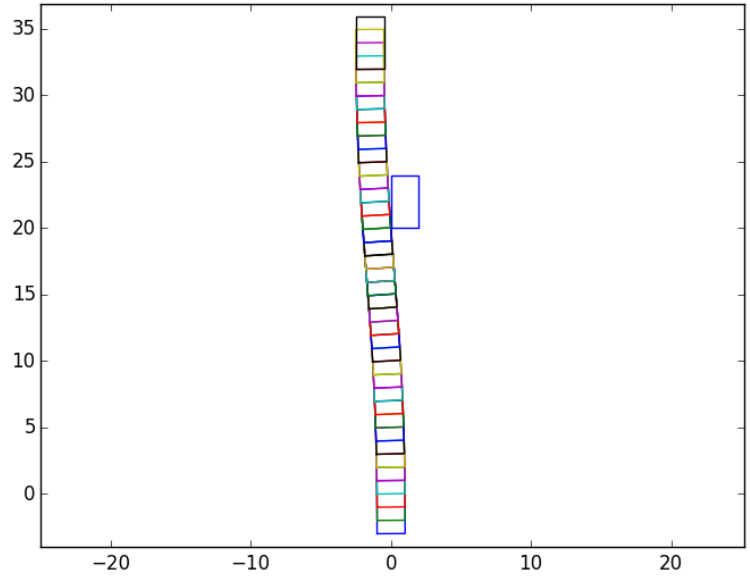
Variables nécessaires :

Position de la voiture et de l'obstacle

Distance de sécurité

Théta et compteur obtenu à la simulation

Dépasser l’obstacle



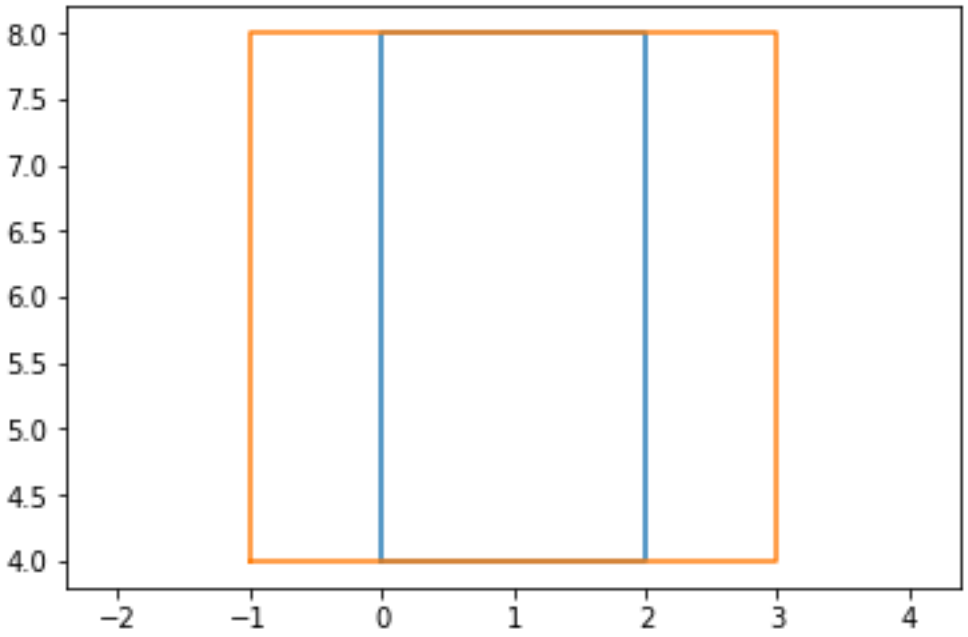
Variables nécessaires :

Position de la voiture et de l’obstacle

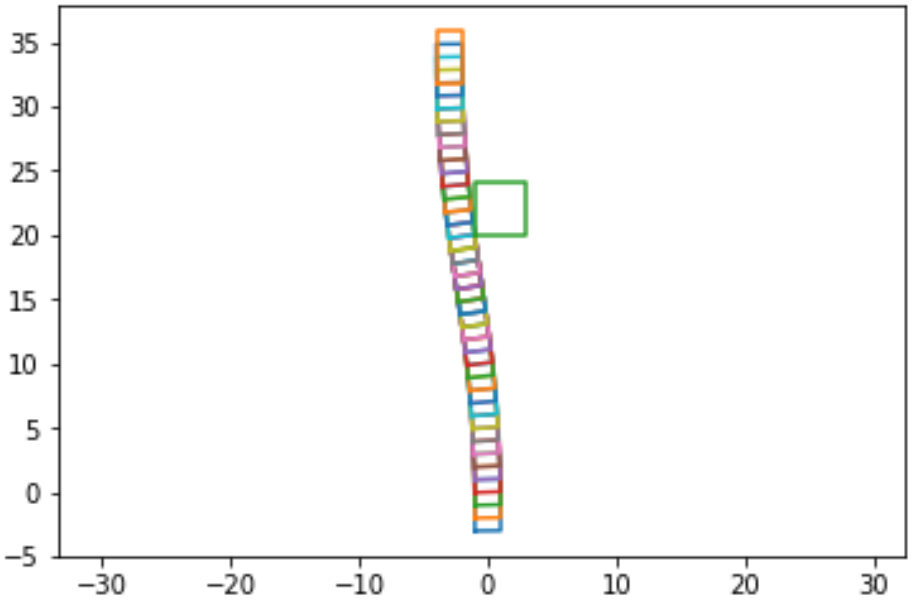
Distance de sécurité

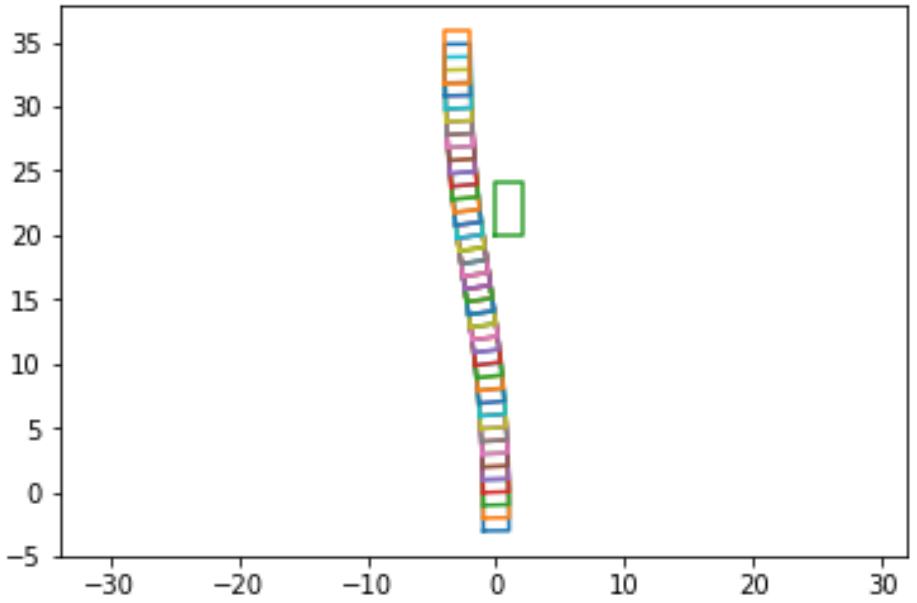
Théta et compteur obtenu à la simulation

Elargissement fictif des obstacles



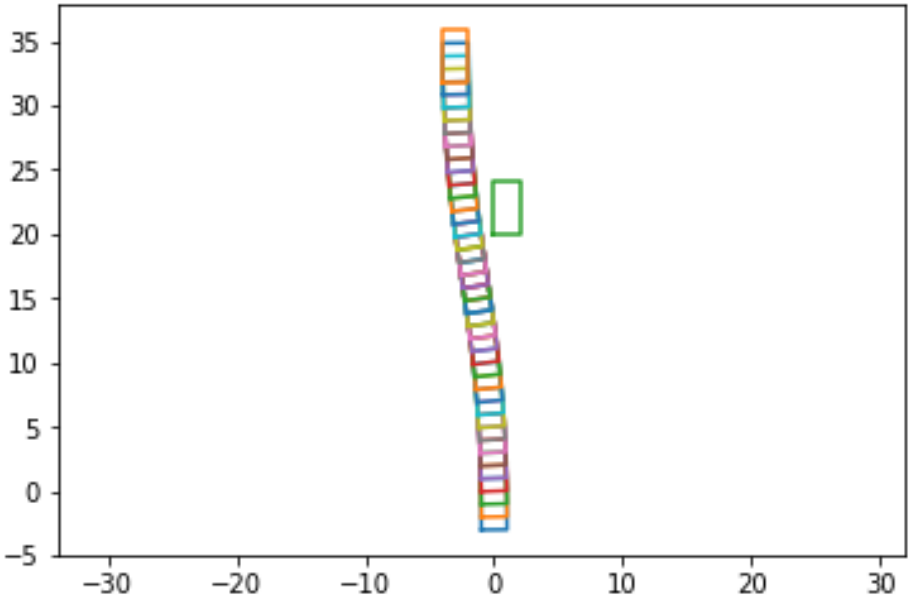
Dépassement de l’obstacle fictif



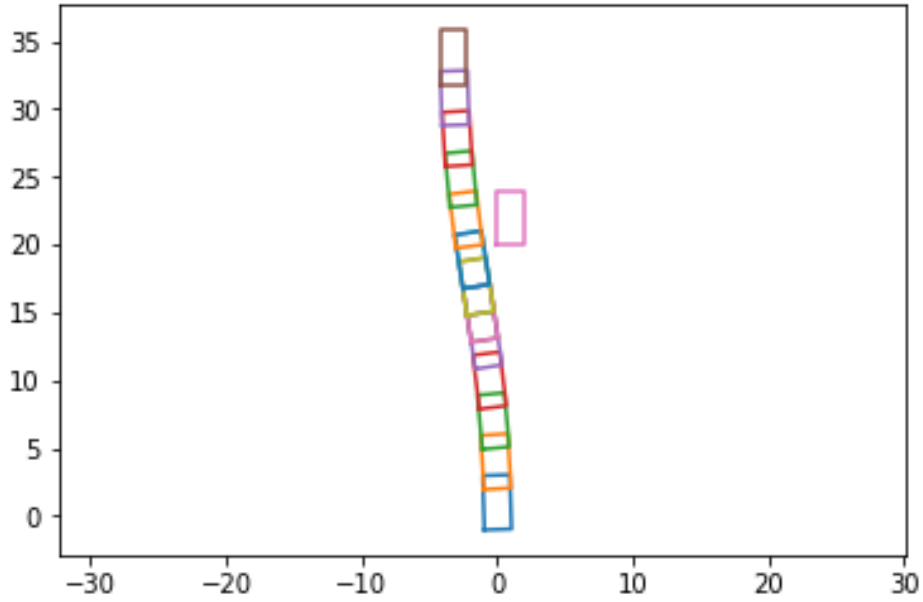


Influence de la vitesse

Vitesse = 1



Vitesse = 3

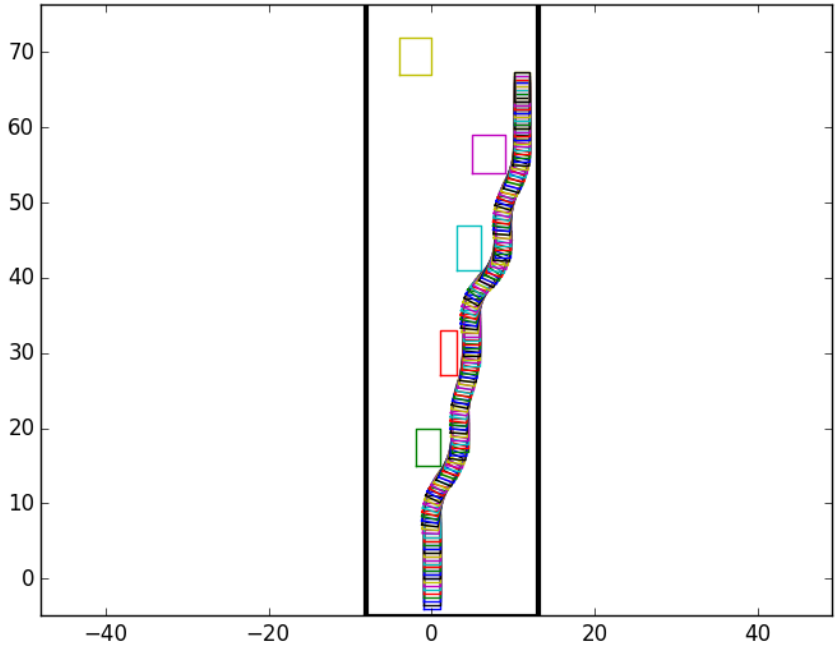


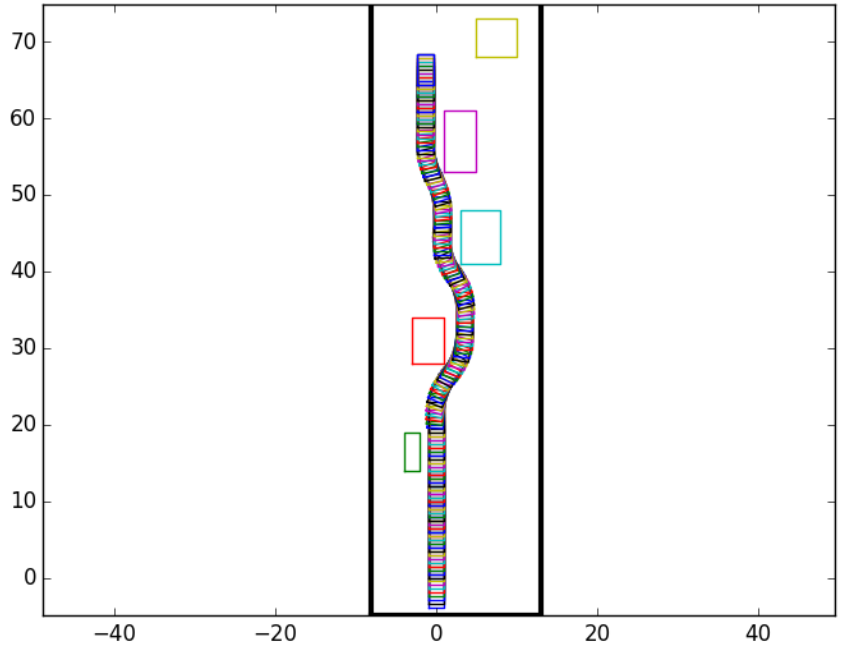
Trajectoire sur un parcours : programme final

Variables nécessaires :

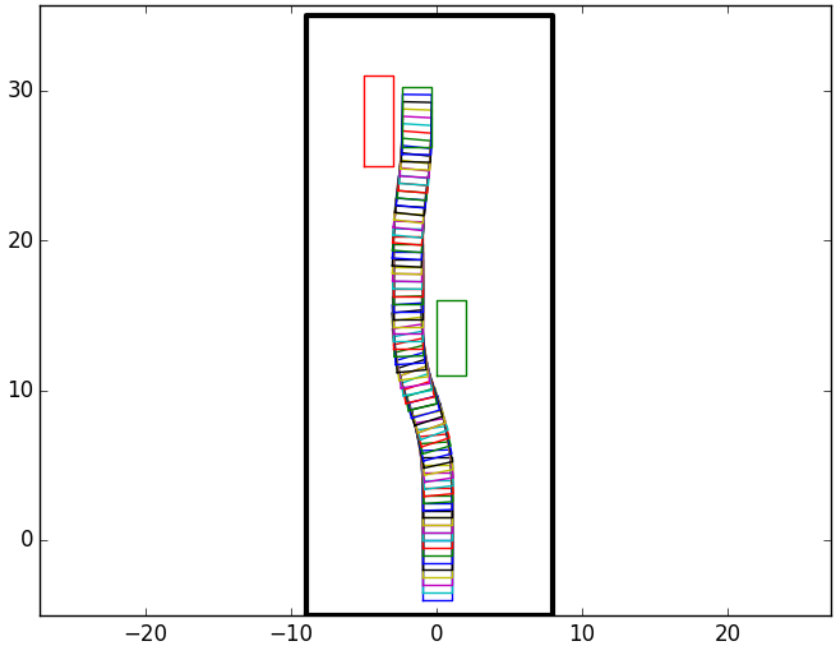
- Position initiale de la voiture
- Précision des simulations (dichotomies)
- Nombre d'obstacles
- Longueur et largeur maximale des obstacles
- Vitesse de la voiture
- Distance de détection et de sécurité

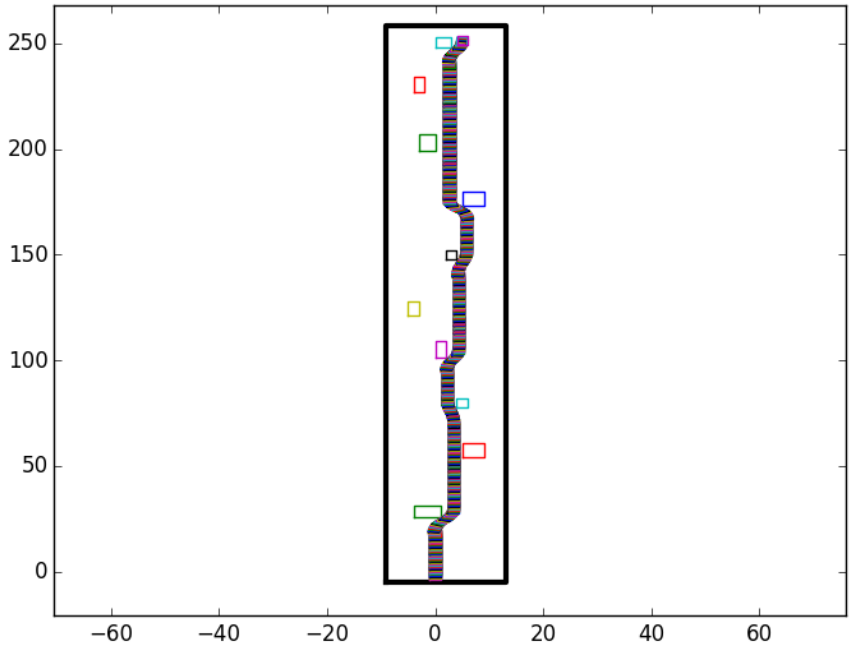
Résultats





Résultat

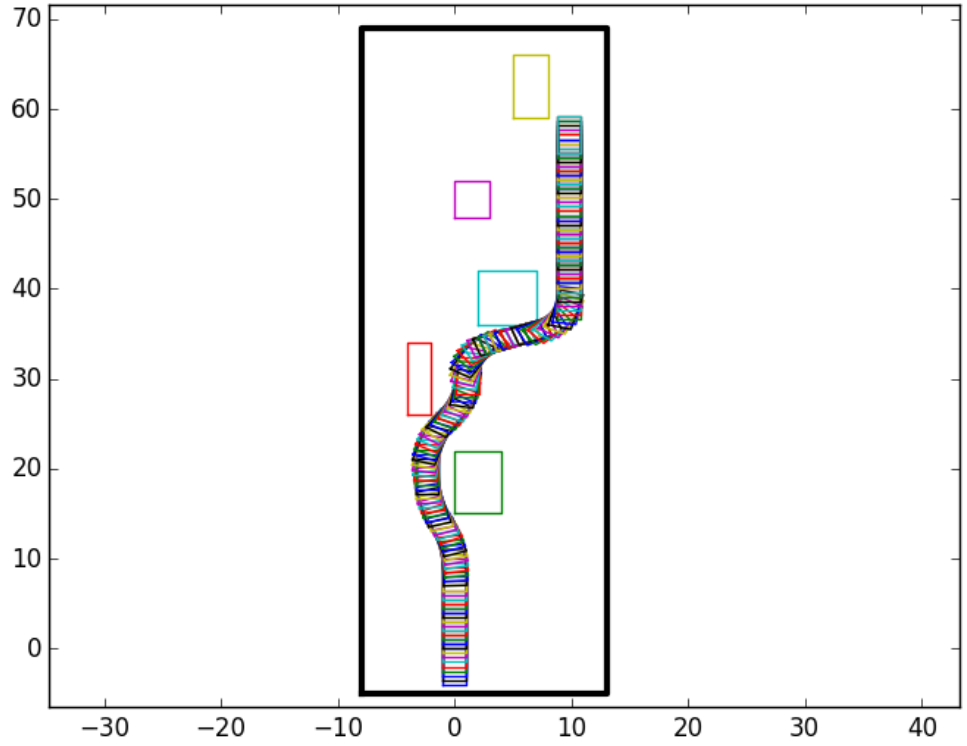




Critiques

- Vitesse de la voiture constante
- Trajectoire pas forcément optimale
- Manque parfois de précision

Critiques



Conclusion

- Modélisation satisfaisante

- Amélioration possible

Fin

Annexes

```
import matplotlib.pyplot as plt
from math import *
import numpy as np
from copy import deepcopy
import random

abs_Voiture=np.array([-1,-1,1,1,-1])
ord_Voiture=np.array([-4,0,0,-4,-4])


def detection_Obstacle(X1,Y1,X2,Y2): # test d'intersection de d'un segment avec une droite

    B=((Y2[1]-Y2[0])/(X2[1]-X2[0])) #Coeff dir de la droite représentant la face avant de l'
    b=Y2[1]-B*X2[1]   #Ordonnee a l'origine de cette meme droite

    if X1[1]!=X1[0]:

       A=((Y1[1]-Y1[0])/(X1[1]-X1[0])) #Coeff dir de la droite représentant l'emission du so
       a=Y1[1]-A*X1[1] #Ordonnee a l'origine de cette meme droite
       Xi=(b-a)/(A-B) #Abscisse du point d'intersection des 2 droites
       Yi=(A*Xi)+a #Ordonnée du point d'intersection des 2 droites
       VectIA=[X2[0]-Xi,Y2[0]-Yi]
       VectIB=[X2[1]-Xi,Y2[1]-Yi]
       Ps=(VectIA[0]*VectIB[0])+(VectIA[1]*VectIB[1])
       if Ps<=0: #Test d'appartenance du point d'intersection au segment AB definisant la fa
           return True
       else :
           return False
    else: #Si l'emission est verticale
```

```
        Xi=X1[0]
        Yi=(B*X1[0])+b
        VectIA=[X2[0]-Xi,Y2[0]-Yi]
        VectIB=[X2[1]-Xi,Y2[1]-Yi]
        Ps=(VectIA[0]*VectIB[0])+(VectIA[1]*VectIB[1])

        if Ps<=0:
            return True
        else :
            return False


def milieu_segment(X,Y):
    Xm=((X[1]+X[0])/2)
    Ym=((Y[1]+Y[0])/2)
    M=np.array([Xm,Ym])
    return M


def point_emission(M,parchoc):
    U=[parchoc[0][1]-parchoc[0][0],parchoc[1][1]-parchoc[1][0]]
    Pt_emission=np.array([M[0]-U[1],M[1]+U[0]])
    return Pt_emission

def detection_Obstacle_sonar_triple(X1,Y1,X2,Y2):
    L1=deepcopy(X1)
    L2=deepcopy(Y1)

    abs_parchoc=np.array([L1[1],L1[2]])
    ord_parchoc=np.array([L2[1],L2[2]])
```

```
    parchoc=np.array([abs_parchoc,ord_parchoc])

    sonar_central = milieu_segment(abs_parchoc,ord_parchoc)
    E=point_emission(sonar_central,parchoc)

    abs_emission_sonar_central=np.array([sonar_central[0],E[0]])
    ord_emission_sonar_central=np.array([sonar_central[1],E[1]])

    abs_sonar_gauche=np.array([L1[0],L1[1]])
    ord_sonar_gauche=np.array([L2[0],L2[1]])

    abs_sonar_droite=np.array([L1[3],L1[2]])
    ord_sonar_droite=np.array([L2[3],L2[2]])

    abs_face_avant_obstacle=np.array([X2[0],X2[3]])
    ord_face_avant_obstacle=np.array([Y2[0],Y2[3]])

    if detection_Obstacle(abs_emission_sonar_central,ord_emission_sonar_central,abs_face_avan

        return(True)
    else:
        return(False)


def rotation(X1,Y1,teta): # permet de faire tourner la voiture
    Gv=np.array([(X1[2]+X1[0])/2,(Y1[2]+Y1[0])/2]) # centre de gravité de la voiture
    U1=np.array([X1[0]-Gv[0],Y1[0]-Gv[1]])
    U2=np.array([X1[1]-Gv[0],Y1[1]-Gv[1]])
    U3=np.array([X1[2]-Gv[0],Y1[2]-Gv[1]])
    U4=np.array([X1[3]-Gv[0],Y1[3]-Gv[1]])
```

```
    M_rot=np.array([[cos(teta),-sin(teta)],
                    [sin(teta),cos(teta)]])

    U1=np.dot(M_rot,U1)
    U2=np.dot(M_rot,U2)
    U3=np.dot(M_rot,U3)
    U4=np.dot(M_rot,U4)

    X_obtenues=np.array([Gv[0]+U1[0],Gv[0]+U2[0],Gv[0]+U3[0],Gv[0]+U4[0],Gv[0]+U1[0]])
    Y_obtenues=np.array([Gv[1]+U1[1],Gv[1]+U2[1],Gv[1]+U3[1],Gv[1]+U4[1],Gv[1]+U1[1]])
    rotation=np.array([X_obtenues,Y_obtenues])

    return rotation


def distance_entre_2_points(A,B): # A et B des listes avec l'abscisse et l'ordonnée des 2 poi
    a,b,c,d=A[0],A[1],B[0],B[1]
    D=sqrt((c-a)**2+(d-b)**2)
    return D


def vrai_distance_voiture_obstacle(X1,Y1,X2,Y2):

    Gv=[(X1[2]+X1[0])/2,(Y1[2]+Y1[0])/2] # centre de gravité de la voiture
    Go=[(X2[2]+X2[0])/2,(Y2[2]+Y2[0])/2] # centre de gravité de l'obstacle

    I=[Go[0],Gv[1]]

    d=distance_entre_2_points(I,Go)
```

```
    return d

def obstacle_fictif_pour_evitement(X1):
    L1=[]
    L1.append(X1[0]-0.5)
    L1.append(X1[1]-0.5)
    L1.append(X1[2]+0.5)
    L1.append(X1[3]+0.5)
    L1.append(X1[4]-0.5)
    return(L1)

def direction_voiture(X1,Y1):
    norme=sqrt((X1[2]-X1[1])**2+(Y1[2]-Y1[1])**2)
    U=[(1/norme)*(-(Y1[2]-Y1[1])),(1/norme)*(X1[2]-X1[1])]
    return U


def deplacer_voiture(X1,Y1,vitesse,teta):

    U=direction_voiture(X1,Y1)

    L1=deepcopy(X1)
    L2=deepcopy(Y1)

    L1=[(x+vitesse*U[0]) for x in L1]
    L2=[(x+vitesse*U[1]) for x in L2]

    rotation1=rotation(L1,L2,teta)
```

```
    L1=rotation1[0]
    L2=rotation1[1]

    return([L1,L2])


def avancer(X1,Y1,vitesse):

    L1=deepcopy(X1)
    L2=deepcopy(Y1)
    U=direction_voiture(L1,L2)
    L1=[(x+vitesse*U[0]) for x in L1]
    L2=[(x+vitesse*U[1]) for x in L2]
    plt.plot(L1,L2)

    return([L1,L2])

def depasser_obstacle(X1,Y1,X2,Y2,vitesse):
    Y_ref=Y2[0]
    L1=deepcopy(X1)
    L2=deepcopy(Y1)
    M=milieu_segment([L1[1],L1[2]],[L2[1],L2[2]])
    while M[1]<Y_ref:
        L=avancer(L1,L2,vitesse)
        L1=L[0]
        L2=L[1]
        M=milieu_segment([L1[1],L1[2]],[L2[1],L2[2]])
        plt.plot(L1,L2)
    return([L1,L2])
```

```
def simulation_pour_trouver_teta(X1,Y1,X2,Y2,vitesse,d,precision):

    teta1,teta2,teta3,teta4=0,1,0,(-1)

    compteur1,compteur2=0,0

    D=vrai_distance_voiture_obstacle(X1,Y1,X2,Y2)

    L1=deepcopy(X1)
    L2=deepcopy(Y1)

    if not detection_Obstacle_sonar_triple(L1,L2,X2,Y2): # Test pour savoir si les sonars int
        return ([0,0])

    while (abs(teta1-teta2))>precision: #test avec teta positif

        teta = (teta1+teta2)/2

        while D>d and L2[0]<=L2[1] : #Je verifie que la voiture ne tourne pas trop pour attei

            A=deplacer_voiture(L1,L2,vitesse,teta)
            L1=A[0]
            L2=A[1]
            D=vrai_distance_voiture_obstacle(L1,L2,X2,Y2)
            compteur1+=1

        if detection_Obstacle_sonar_triple(L1,L2,X2,Y2):

            teta1,teta2=teta,teta2
```

```
    else :

        teta1,teta2=teta1,teta

    D=vrai_distance_voiture_obstacle(X1,Y1,X2,Y2)
    L1=deepcopy(X1)
    L2=deepcopy(Y1)
    vrai_compteur1=compteur1
    compteur1=0

D=vrai_distance_voiture_obstacle(X1,Y1,X2,Y2)
L1=deepcopy(X1)
L2=deepcopy(Y1)

while (abs(teta4-teta3))>precision: #test avec teta negatif

    teta0 = (teta4+teta3)/2

    while D>d and L2[0]<=L2[1] : #Je verifie que la voiture ne tourne pas trop pour attei

        A=deplacer_voiture(L1,L2,vitesse,teta0)
        L1=A[0]
        L2=A[1]
        D=vrai_distance_voiture_obstacle(L1,L2,X2,Y2)
        compteur2+=1

    if detection_Obstacle_sonar_triple(L1,L2,X2,Y2):

        teta3,teta4=teta0,teta4
```

```
        else :

            teta3,teta4=teta3,teta0

        D=vrai_distance_voiture_obstacle(X1,Y1,X2,Y2)
        L1=deepcopy(X1)
        L2=deepcopy(Y1)
        vrai_compteur2=compteur2
        compteur2=0

    teta=(teta1+teta2)/2
    teta0=(teta3+teta4)/2

    if abs(teta)>abs(teta0):
        return([teta0,vrai_compteur2])
    else:
        return([teta,vrai_compteur1])


def declenchement_de_l_evitement(X1,Y1,compteur,vitesse,d,teta):

    L1=deepcopy(X1)
    L2=deepcopy(Y1)

    for _ in range (compteur):
        A=deplacer_voiture(L1,L2,vitesse,teta)
        L1=A[0]
        L2=A[1]
        plt.plot(L1,L2)
```

```
    return([L1,L2])


def redressement_voiture(X1,Y1,compteur,vitesse,teta):

    L1=deepcopy(X1)
    L2=deepcopy(Y1)

    for _ in range (compteur):
        A=deplacer_voiture(L1,L2,vitesse,-teta)
        L1=A[0]
        L2=A[1]
        plt.plot(L1,L2)

    return([L1,L2])


def max(L):
    n=len(L)
    M=0
    for x in L:
        if x>=M:
            M=x
    return M

def min(L):
    n=len(L)
    m=L[0]
    for x in L:
        if x<=m:
```

```
            m=x
    return m


def parcours(n,l,L):
    ord_0=0
    L_abs=[]
    L_ord=[]
    L_abs_min=[]

    for i in range(n):
        l1=random.randint(2,l)
        L1=random.randint(4,L)
        abs_0=random.randint(-5,5)
        L_abs_min.append(abs_0)
        d=random.randint(50,100)
        ord_0=ord_0+d
        abs_obs=[abs_0,abs_0,abs_0+l1,abs_0+l1,abs_0]
        ord_obs=[ord_0,ord_0+L1,ord_0+L1,ord_0,ord_0]
        L_abs.append(abs_obs)
        L_ord.append(ord_obs)

    for i in range (n):
        plt.plot(L_abs[i],L_ord[i])
    m=min(L_abs_min)
    M=max(L_abs_min)
    abs_contour=[m-4,m-4,M+l+3,M+l+3,m-4]
    ord_contour=[-1,ord_0+L+10,ord_0+L+10,-1,-1]
    plt.plot(abs_contour,ord_contour,color='black',linewidth=3)
    return([L_abs,L_ord])
```

```
def trajectoire_sur_le_parcours(X1,X2,vitesse,d_securite,d_detection,n,l,L,precision):
    L1=deepcopy(X1)
    L2=deepcopy(X2)
    obstacles=parcours(n,l,L)

    for K in range (n):

        obstacles_fictif=obstacle_fictif_pour_evitement(obstacles[0][K])
        D=vrai_distance_voiture_obstacle(L1,L2,obstacles[0][K],obstacles[1][K])
        while D>d_detection:
            A=avancer(L1,L2,vitesse)
            L1=A[0]
            L2=A[1]
            D=vrai_distance_voiture_obstacle(L1,L2,obstacles[0][K],obstacles[1][K])

        Z=simulation_pour_trouver_teta(L1,L2,obstacles_fictif,obstacles[1][K],vitesse,d_secur
        teta=Z[0]

        compt=Z[1]
        B=declenchement_de_l_evitement(L1,L2,compt,vitesse,d_securite,teta)
        L1=B[0]
        L2=B[1]

        C=depasser_obstacle(L1,L2,obstacles[0][K],obstacles[1][K],vitesse)
        L1=C[0]
        L2=C[1]

        D=redressement_voiture(L1,L2,compt,vitesse,teta)
        L1=D[0]
```

```
        L2=D[1]

plt.plot(abs_Voiture,ord_Voiture)
trajectoire_sur_le_parcours(abs_Voiture,ord_Voiture,1,10,20,5,8,10,0.0000000001)
plt.axis("equal")
plt.show()
```

Annexes