



PROJET SAMI



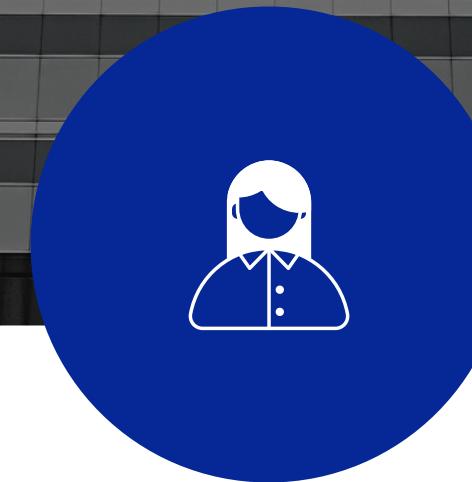
# Optimisation d'un chemin parcouru par un robot LEGO EV3

Groupe : Sami et ses outils

# SOMMAIRE

- PARTIE MATHEMATIQUES
- PARTIE AUTOMATIQUE
- PARTIE INFORMATIQUE

# PARTIE MATHÉMATIQUES



## Positionnement du problème

Présentation du problème, des objectifs à atteindre et choix de l'algorithme.

## algorithme de graham scan

Description du principe  
Méthode d'implémentation

## Les méthodes d'insertion

Comparaison des méthodes d'insertion à travers différents algorithmes

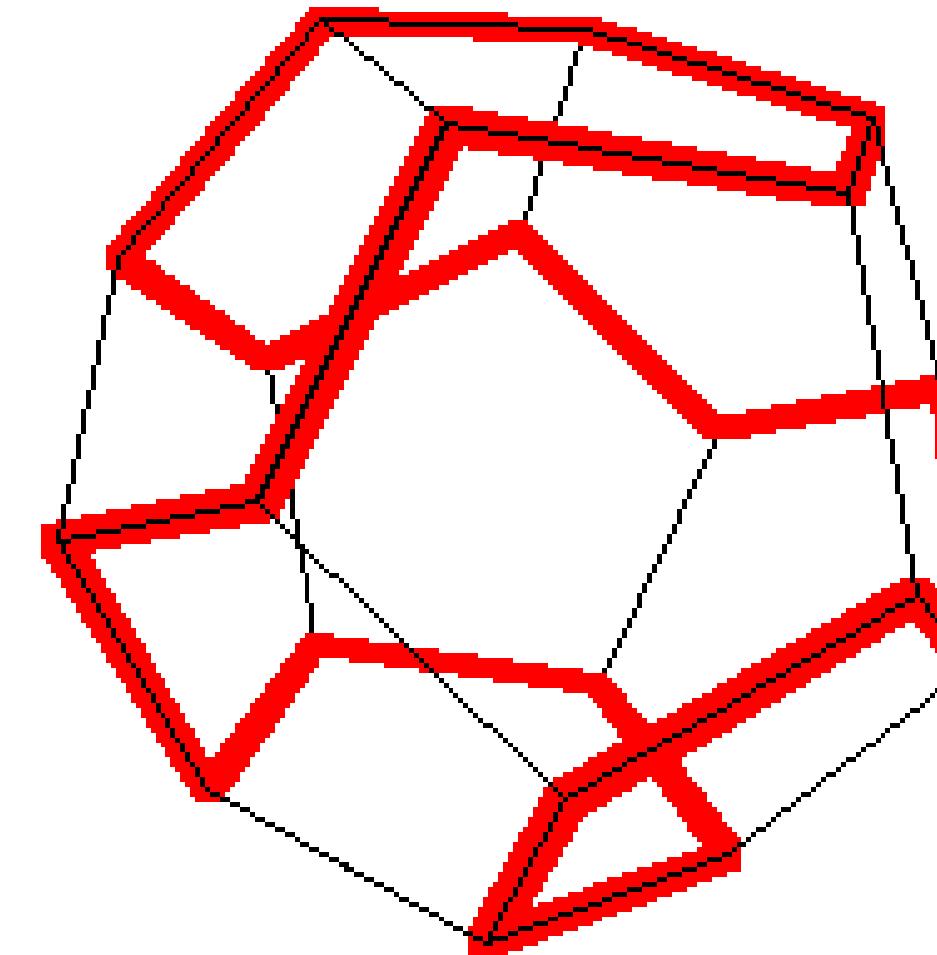
KHATTOU Adel

OEUDRAOGO Aicha

PUIBUSQUE Thomas

# Le voyageur du commerce

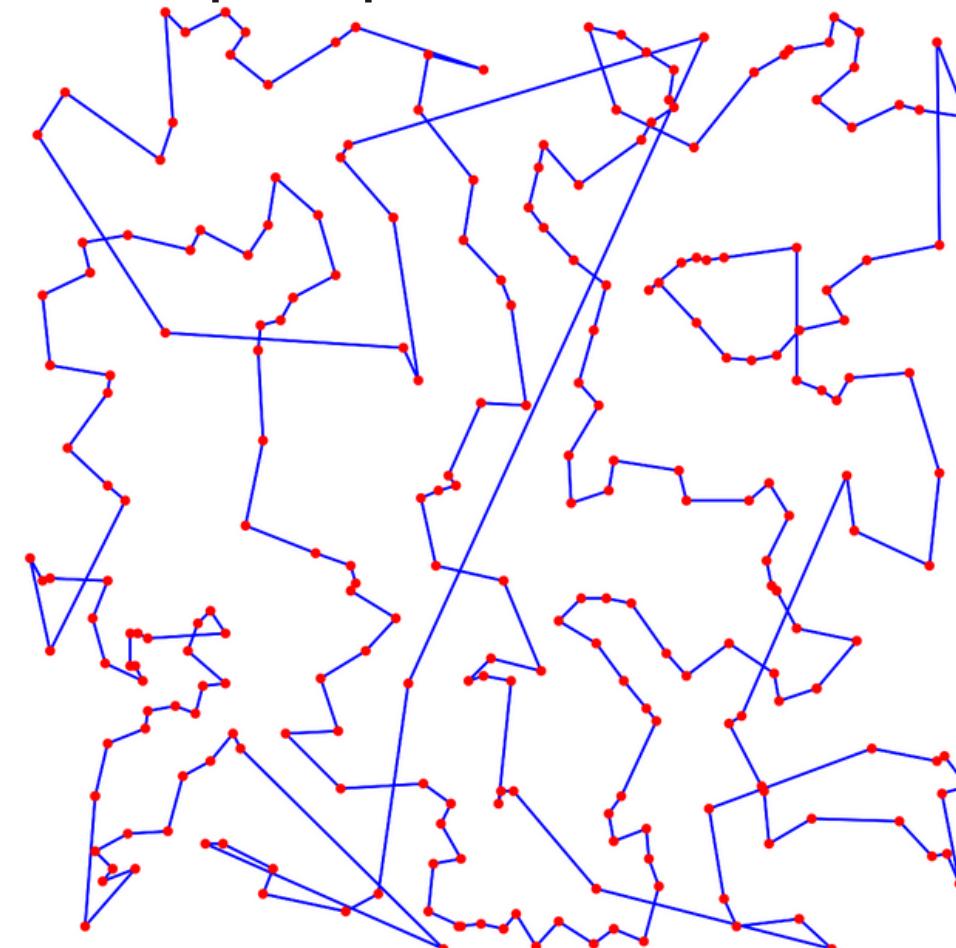
**Quel chemin faut-il choisir afin de minimiser la distance parcourue ?**



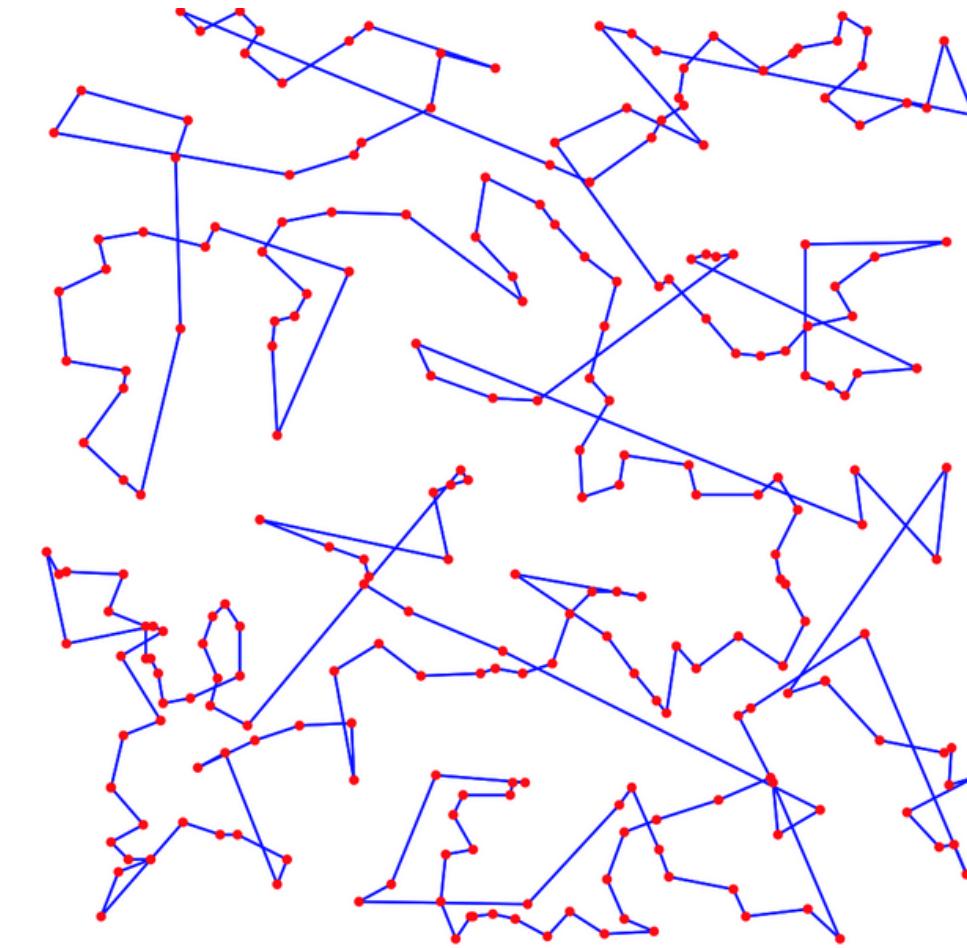
# Choix de l'algorithme

## Méthodes étudiées

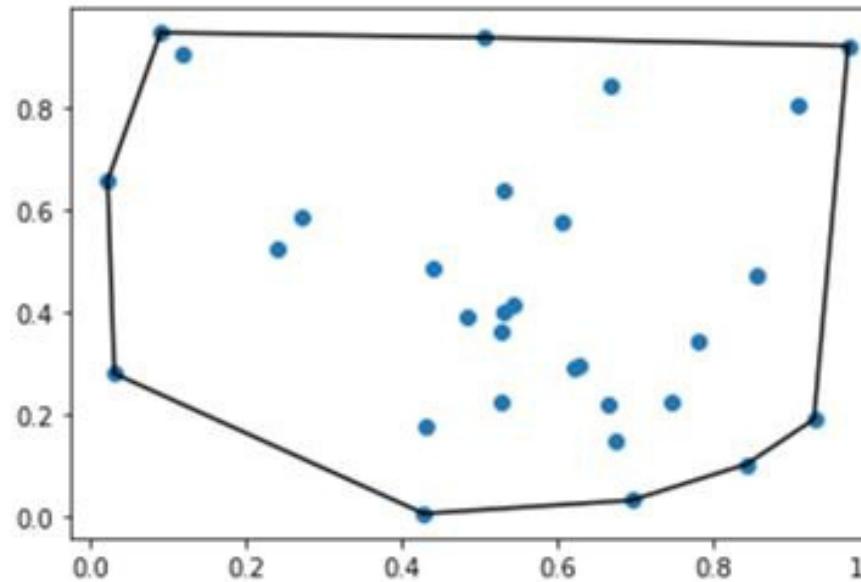
- Méthode par insertion
- Les plus proches voisins



- L'arbre couvrant de poids minimums



# Enveloppe convexe



Polygone

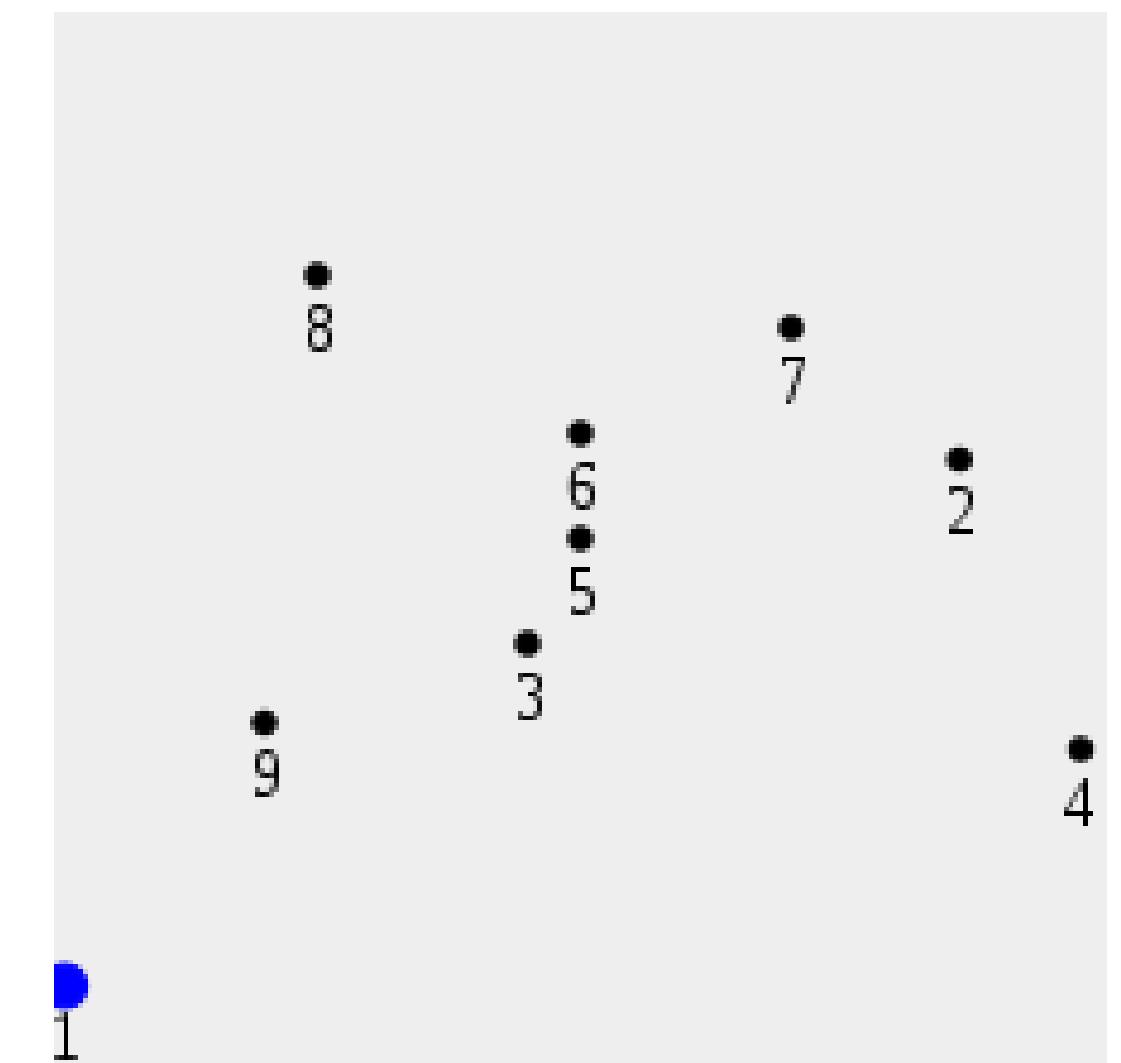
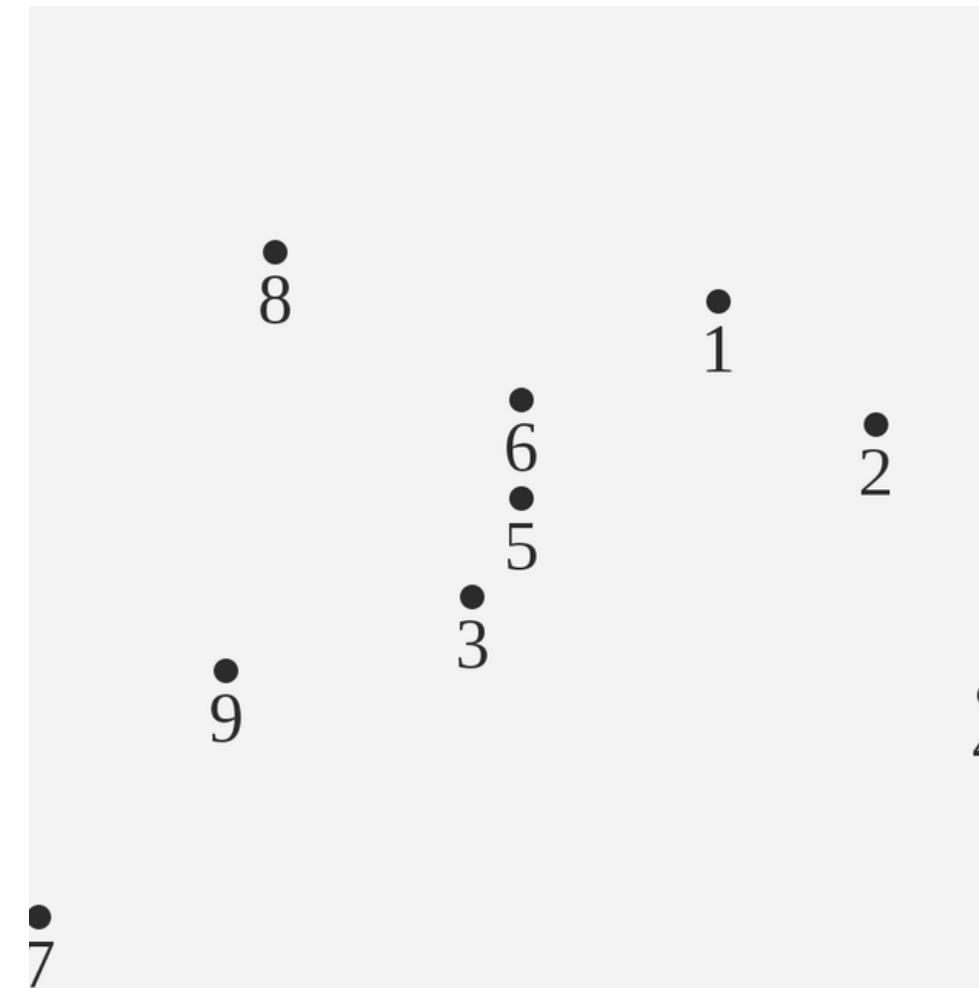
Convexe



Non convexe



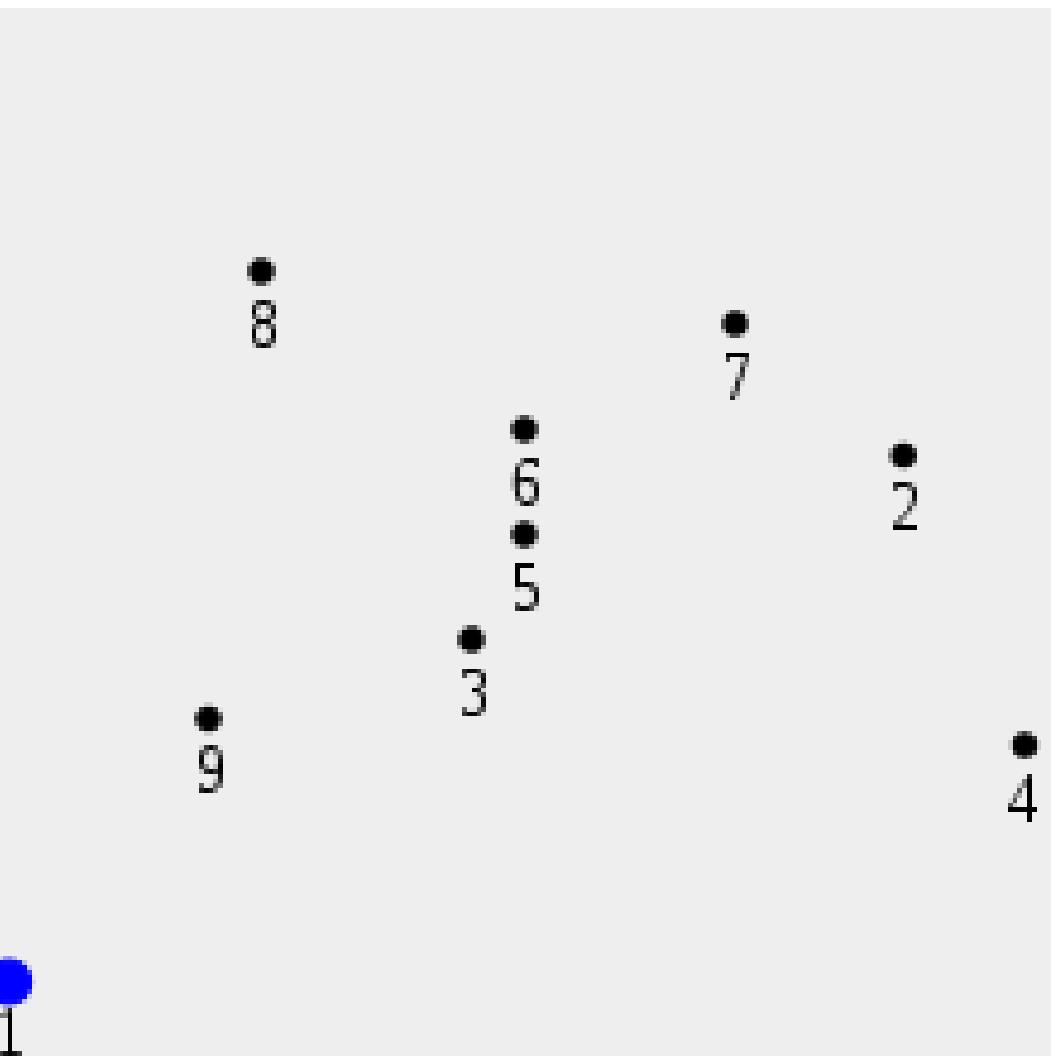
# Algorithme de graham



# Choix du premier point

On choisit le point avec la plus petite ordonnée

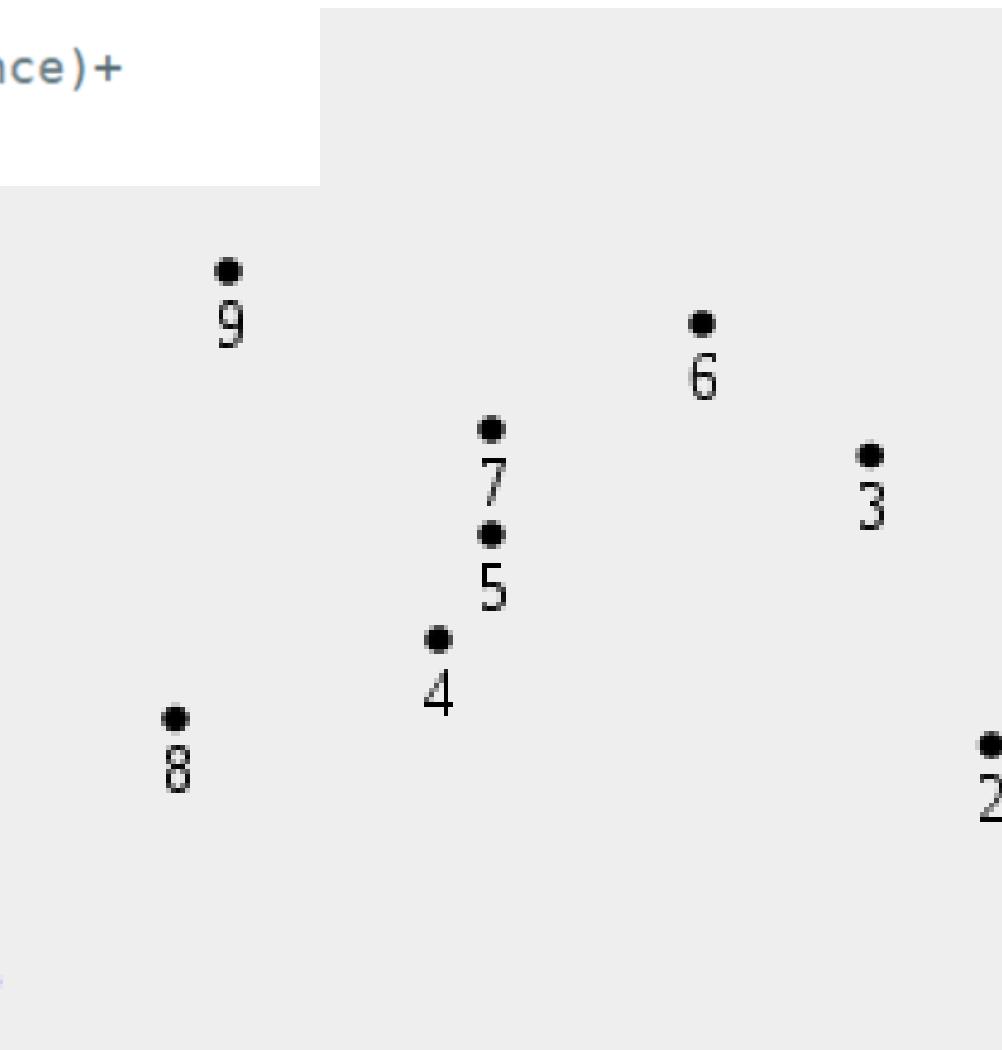
```
min_idx=None
for i,(x,y) in enumerate(points):
    if min_idx==None or y<points[min_idx][1]:
        min_idx=i
    if y==points[min_idx][1] and x<points[min_idx][0]:
        min_idx=i
ancrage=points[min_idx]
```



# Tri des points

Le tri se fait par rapport à l'angle que font les points avec le premier point

```
def quicksort(a):
    if len(a)<=1: return a
    smaller,equal,larger=[],[],[]
    piv_ang=polar_angle(a[randint(0,len(a)-1)]) # select ran
pivot
for i in range(0,len(a)):
    pt=a[i]
    pt_ang=polar_angle(pt)# calculate current point angl
    if  pt_ang<piv_ang: smaller.append(pt)
    elif pt_ang==piv_ang: equal.append(pt)
    else: larger.append(pt)
return quicksort(smaller) +sorted(equal,key=distance)+quicksort(larger)
```



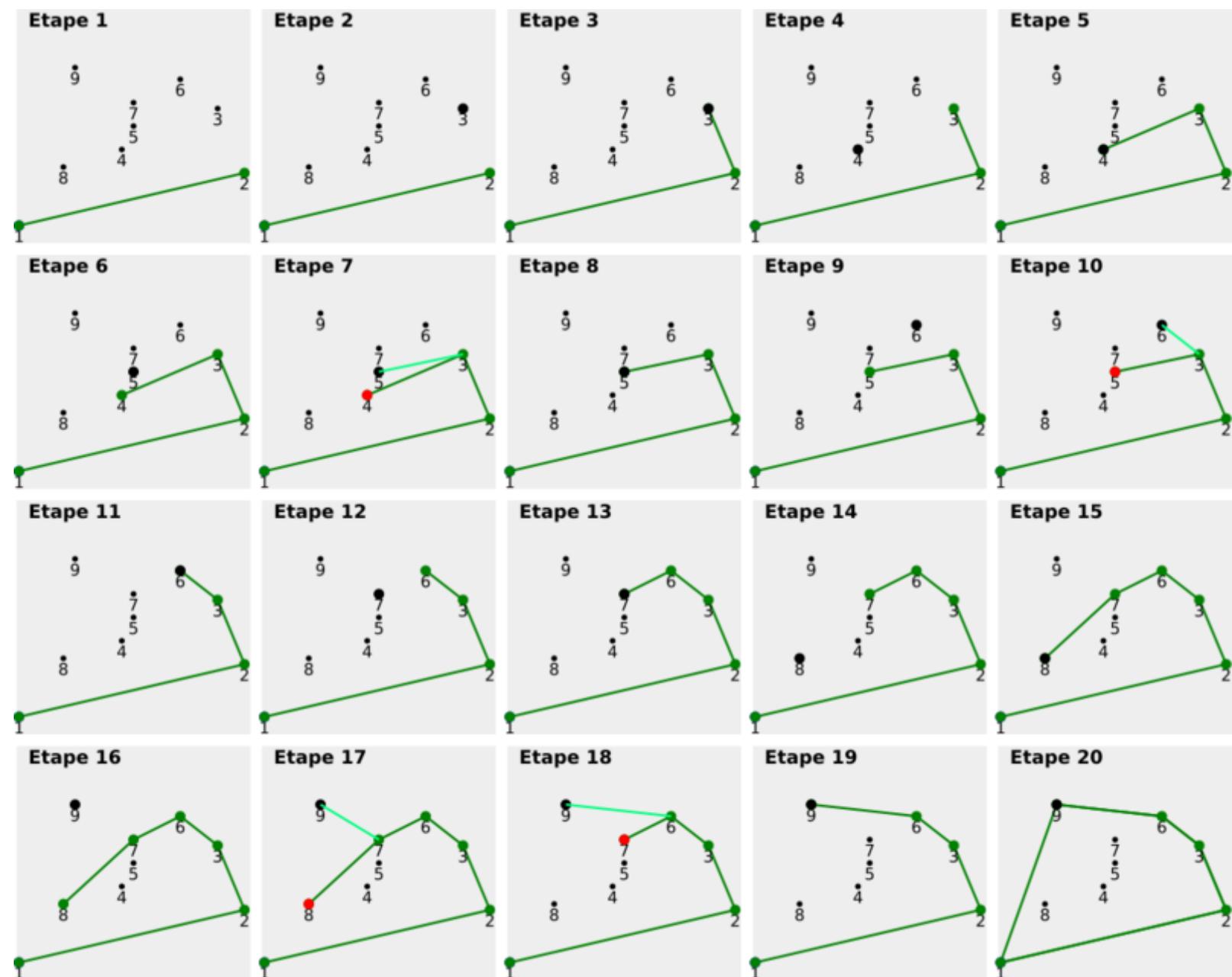
# Construction de l'enveloppe

On parcourt l'ensemble des points triés. Grâce à l'algorithme ci dessous on vérifie la présence de tournant à droite. Lorsqu'un tournant à droite est repéré, l'avant dernier point est retiré.

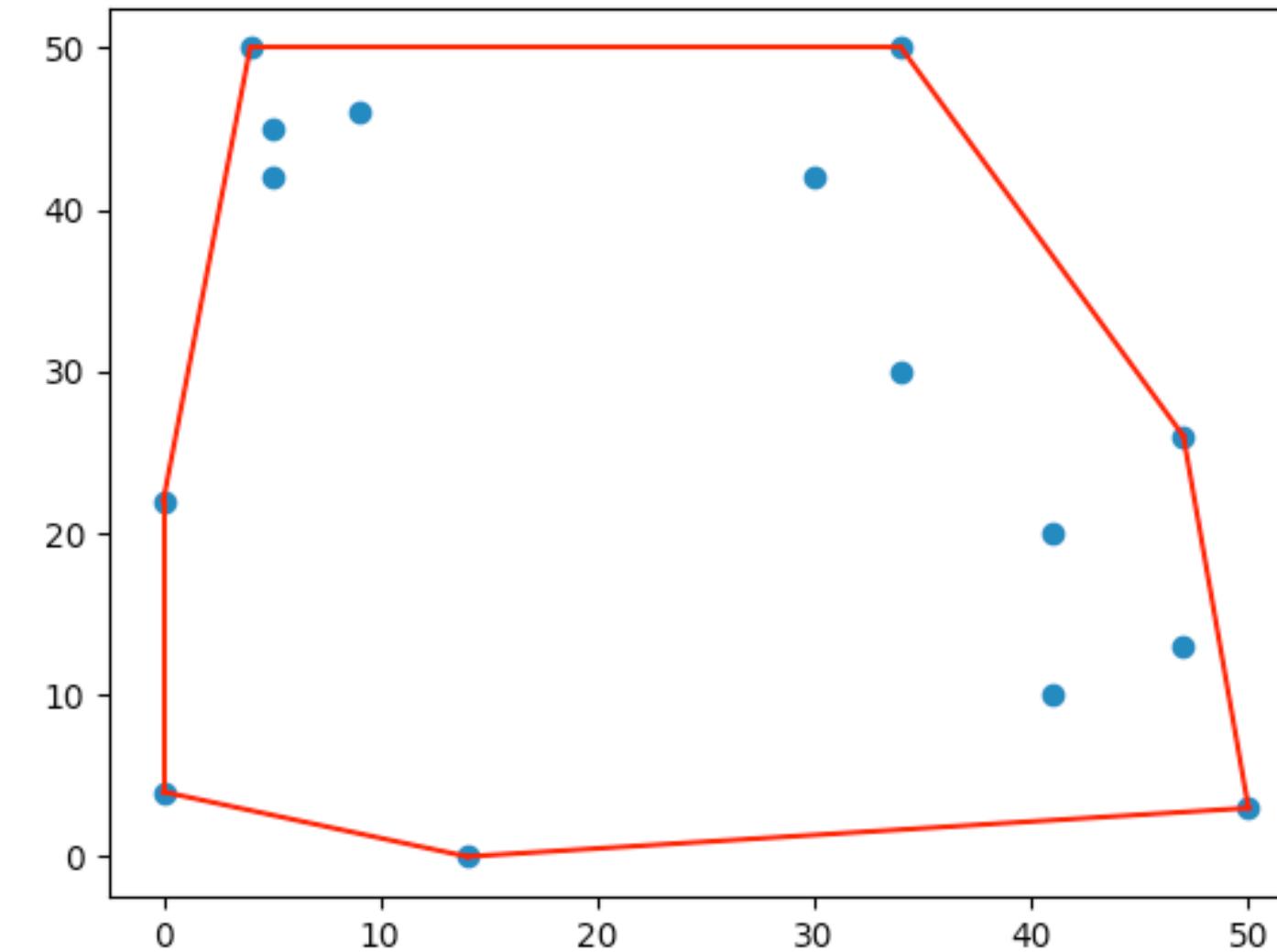
```
-----  
for s in pts_tries[1:]:  
    while det(envelop[-2],hull[-1],s)<=0: #tournant droite  
        del envelop[-1] # backtrack  
        if len(envelop)<2: break  
    envelop.append(s)  
    if show_progress: scatter_plot(points,envelop)  
return envelop
```

# Construction de l'enveloppe

A la fin du parcours on obtient donc notre enveloppe convexe



# Les méthodes d'insertion



# Avant tout...

Algorithme donnant les points qui ne sont pas dans l'enveloppe convexe :

```
def autre(pts, env_convex):
    reste = []
    for i in range(len(pts)):
        k=True
        for j in range(len(env_convex)):
            if pts[i]==env_convex[j]:
                k=False
        if k:
            reste.append(pts[i])
    return(reste)
```

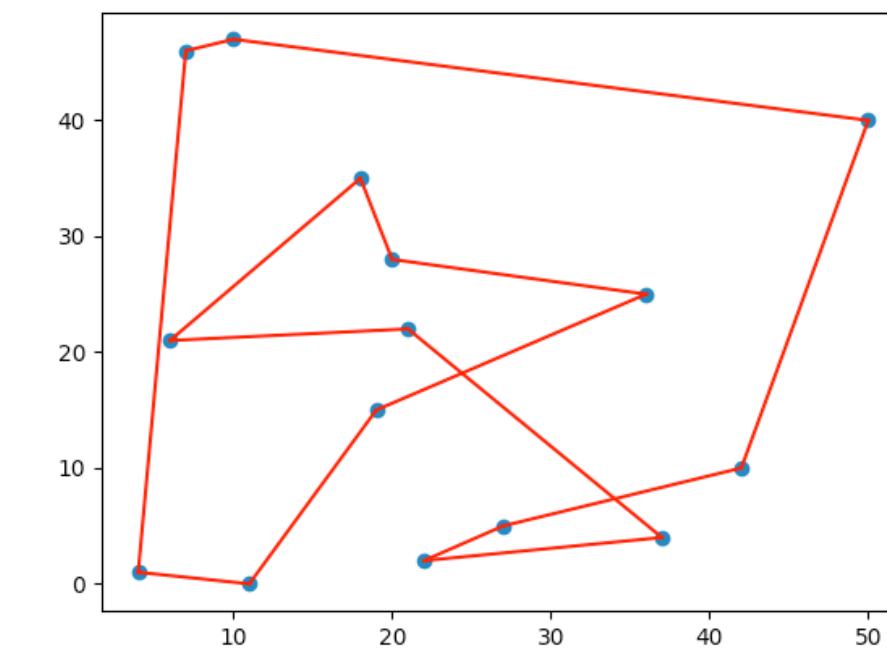
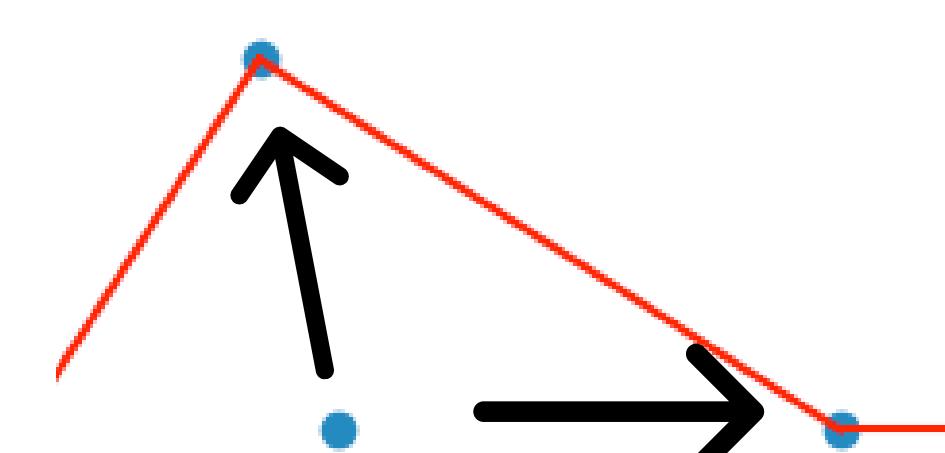


Chaque éléments de la liste pts sont ajoutés à reste lorsqu'ils ne sont pas dans env\_convex.

# Créer le meilleur chemin

**Idée générale :** prendre chaque points qu'il reste et le rajouter entre 2 points de l'enveloppe qui sont les plus proches.

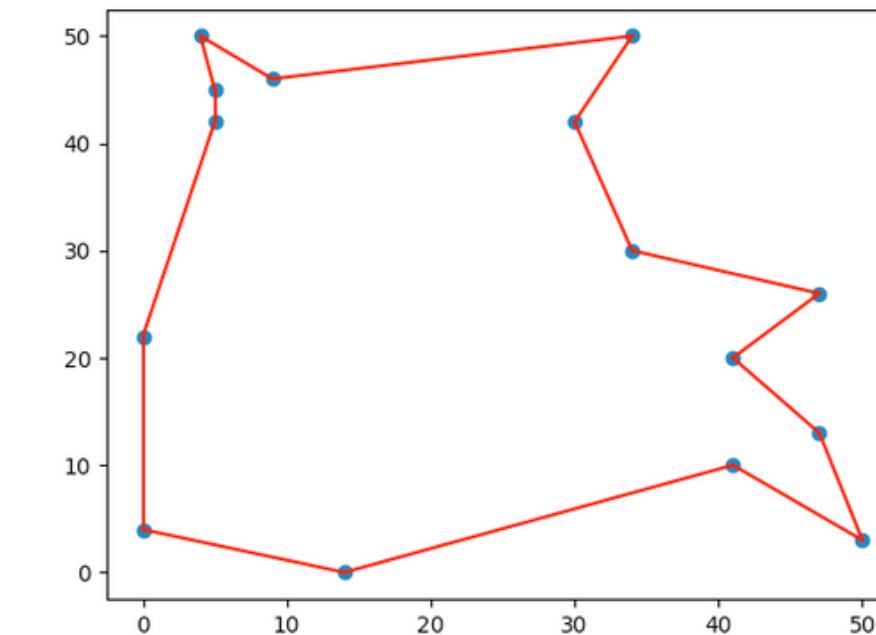
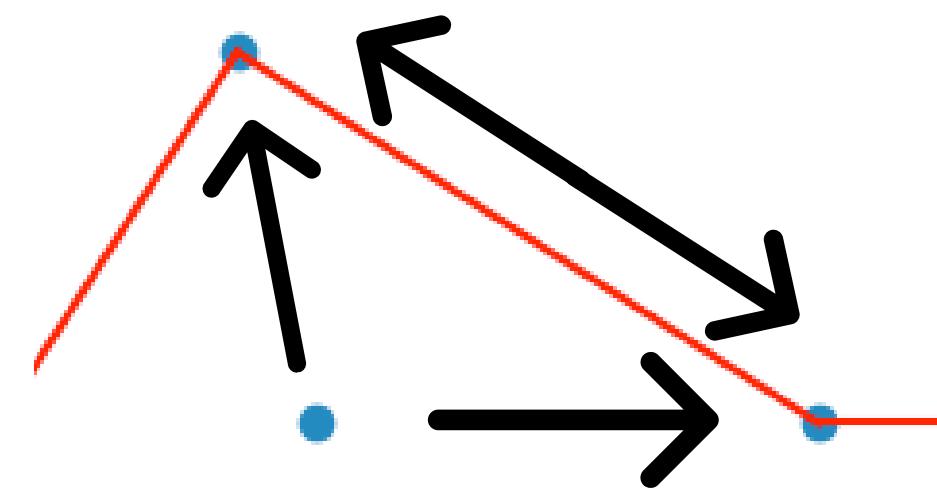
**1ère idée de condition d'insertion :** comparer la somme des distances entre 2 points de l'enveloppe et le point à insérer.



**Problème :** engendre des croisements dans quelques cas.

**Conclusion :** La 1ère idée semble être une condition nécessaire mais pas suffisante.

**2ème idée de condition d'insertion :** comparer la somme des distances entre 2 points de l'enveloppe et le point à insérer auquel on retire la distance entre ces 2 points de l'enveloppe.



**Remarque :** il y a parfois des croisements, sur 100 tests, il y a eu 3 croisements impliquants des croisements simples entre 4 points,

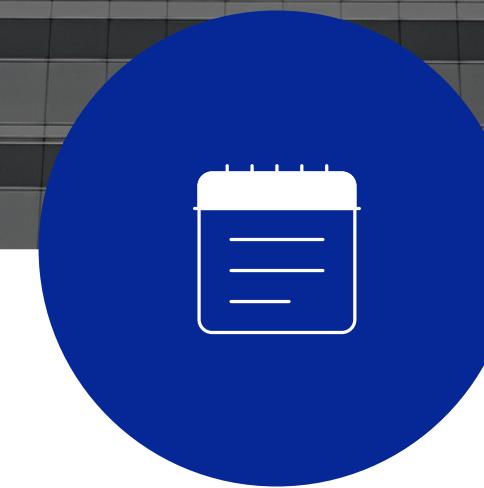
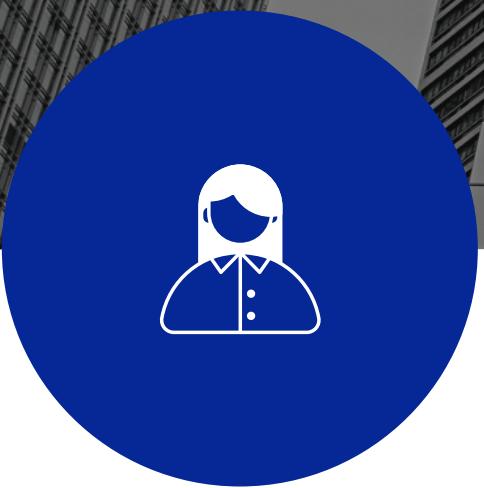
**Conclusion :** La 2ème idée semble être une condition suffisante pour avoir le meilleur chemin.

## Code de l'algorithme d'insertion :

```
def insertion2(pts):
    env_convex=graham_scan(pts)
    scatter_plot(pts,env_convex)
    reste=autre(pts,env_convex)
    for _ in range(len(reste)):
        index=len(env_convex)-1
        l1=distance(env_convex[len(env_convex)-1],reste[0])+distance(env_convex[0],reste[0])-distance(env_convex[len(env_convex)-1],env_convex[0])
        for j in range (len(env_convex)-1):
            l2=distance(env_convex[j],reste[0])+distance(env_convex[j+1],reste[0])-distance(env_convex[j],env_convex[j+1])
            if l1>l2 :
                index=j
                l1=l2
        env_convex.insert(index+1,reste[0])
        del reste[0]
    scatter_plot(pts,env_convex)
    return(env_convex)
```

**Aller plus loin :** faire un algorithme pour enlever les croisements ?

# PARTIE AUTOMATIQUE



## Etude du problème

Le robot EV3 doit être automatisé et doit parcourir le chemin calculé par les programmes de la partie mathématiques.

TRAORE AUDREY

## Objectifs

Après avoir analysé le problème nous avons pu voir plus précisément ce que nous devions faire pour arriver au bout du problème

TRAORE AUDREY

## Solution finale adoptée

On détermine  $k_1$ ,  $k_2$  et  $k_3$  grâce au schéma Simulink.

BOUQUET ELINA

# Étude du problème

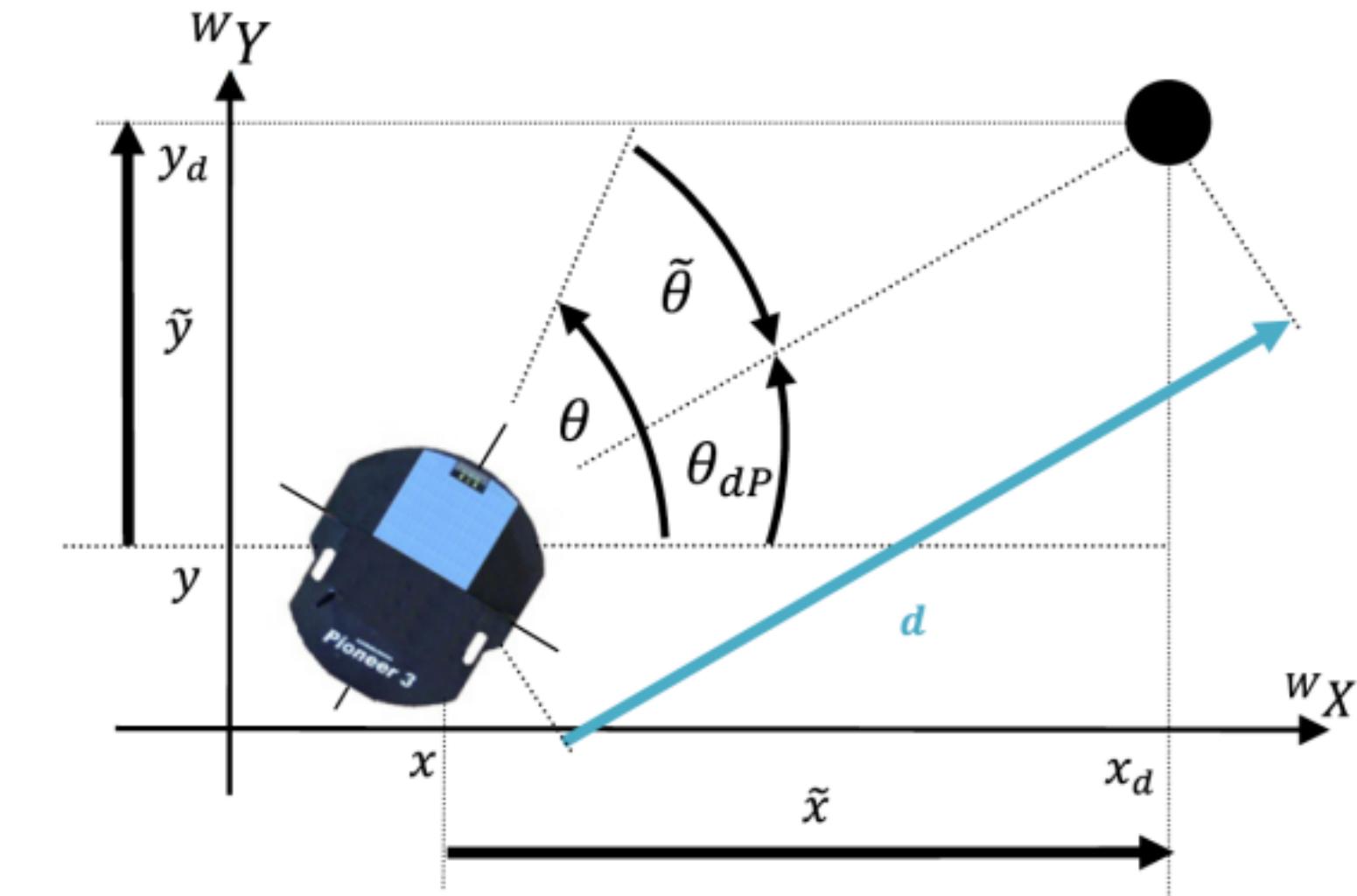


FIGURE 1 – Description du véhicule

La formule :

$$\begin{bmatrix} \omega \\ v \end{bmatrix} = \begin{pmatrix} -\frac{1}{2} & \frac{1}{2} \\ \frac{1}{2} & \frac{1}{2} \end{pmatrix} \begin{bmatrix} v_g \\ v_d \end{bmatrix}$$

- $\dot{x} = v \cos(\theta)$
- $\dot{y} = v \sin(\theta)$
- $\dot{\theta} = \omega$

Comment on l'obtient :

1. La vitesse angulaire de rotation

autour du CIR est  $\dot{\theta} = \frac{d\theta}{dt}$

2. La relation liant la rotation et le parcours est :  $r\Omega_d = (\rho + L)\theta$

parcours roue

arc de centre CIR

3. par dérivation, on obtient :  $r\omega_d = (\rho + L)\dot{\theta}$        $r\omega_g = (\rho - L)\dot{\theta}$

4. équations du déplacement :  $\dot{\theta} = r \frac{\omega_d - \omega_g}{2L}$

5. la vitesse linéaire du centre du robot dans la direction  $\theta$  est :  $v = r \left( \frac{\omega_g + \omega_d}{2} \right)$

6. les composantes de la vitesse linéaire dans le repère OXY sont :

$$\dot{x} = r \left( \frac{\omega_d + \omega_g}{2} \right) \cos \theta \quad \dot{y} = r \left( \frac{\omega_d + \omega_g}{2} \right) \sin \theta$$



Erreur de position et d'orientation :

$$\bar{x} = \textcolor{brown}{x}_d - \textcolor{blue}{x}$$

$$\bar{y} = \textcolor{brown}{y}_d - \textcolor{blue}{y}$$

$$\bar{\theta} = \theta_{dP} - \textcolor{brown}{\theta}$$



Etude d'une loi de commande :

$$v = \frac{d}{k_1 + d} v_{max} \cos(\bar{\theta})$$

$$\omega = \frac{v}{d} \sin(\bar{\theta}) + k_2 \tanh(k_3 \bar{\theta})$$



## Identification des paramètres d'un premier ordre : Ks et tau

$$v_{sol} = K_s \text{ PWM}$$



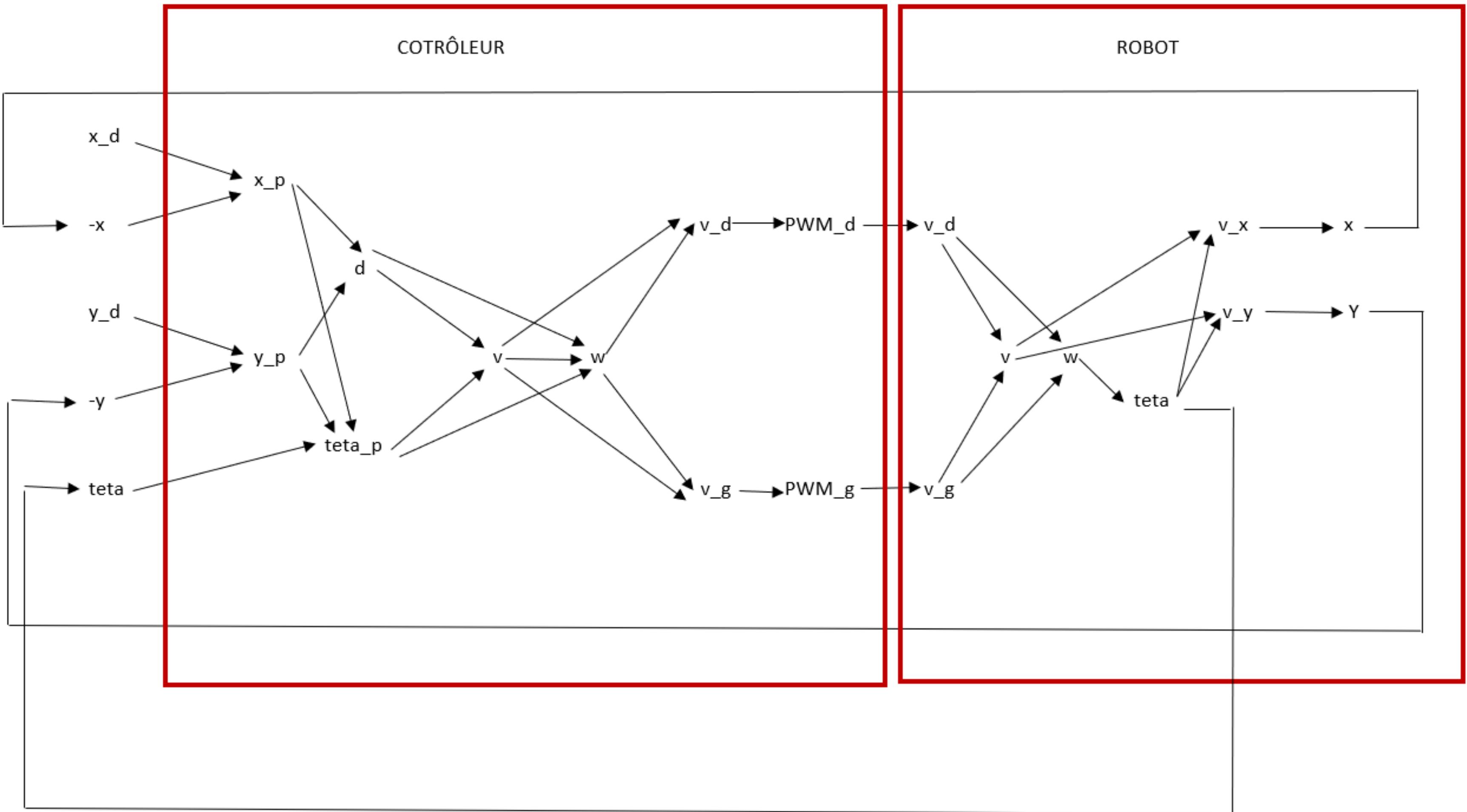
## Les commandes à appliquer sur chaque moteur

Par inversion matricielle et en utilisant toutes les données on obtient les consignes suivantes:

$$PWM_g = \frac{1}{Ks} \left[ \frac{d}{d + k1} v_{max} \times \cos(\tilde{\theta}) - \frac{\Delta}{2} \left( \frac{v}{d} \times \sin(\tilde{\theta}) + k2 \times \tanh(k3 \cdot \tilde{\theta}) \right) \right]$$

$$PWM_d = \frac{1}{Ks} \left[ \frac{d}{d + k1} v_{max} \times \cos(\tilde{\theta}) + \frac{\Delta}{2} \left( \frac{v}{d} \times \sin(\tilde{\theta}) + k2 \times \tanh(k3 \cdot \tilde{\theta}) \right) \right]$$

# Schéma





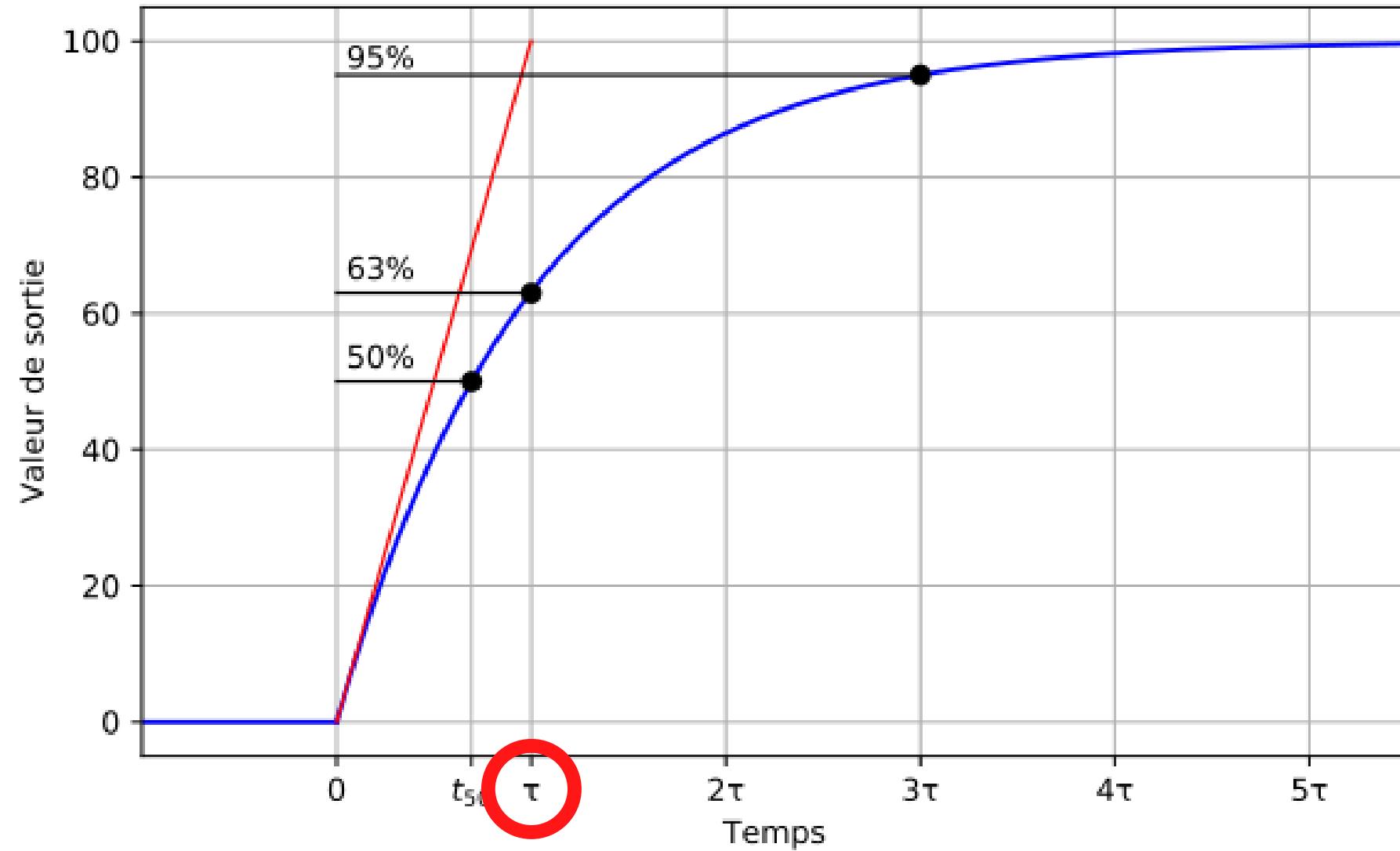
# Objectifs

- 1 - Mesurer le gain statique ( $K_s$ ) et la constante de temps ( $\tau$ ) du robot
- 2 - Construire le modèle robot et retour d'état sur Simulink
- 3 - Régler  $k_1$ ,  $k_2$ ,  $k_3$  pour optimiser le déplacement du robot

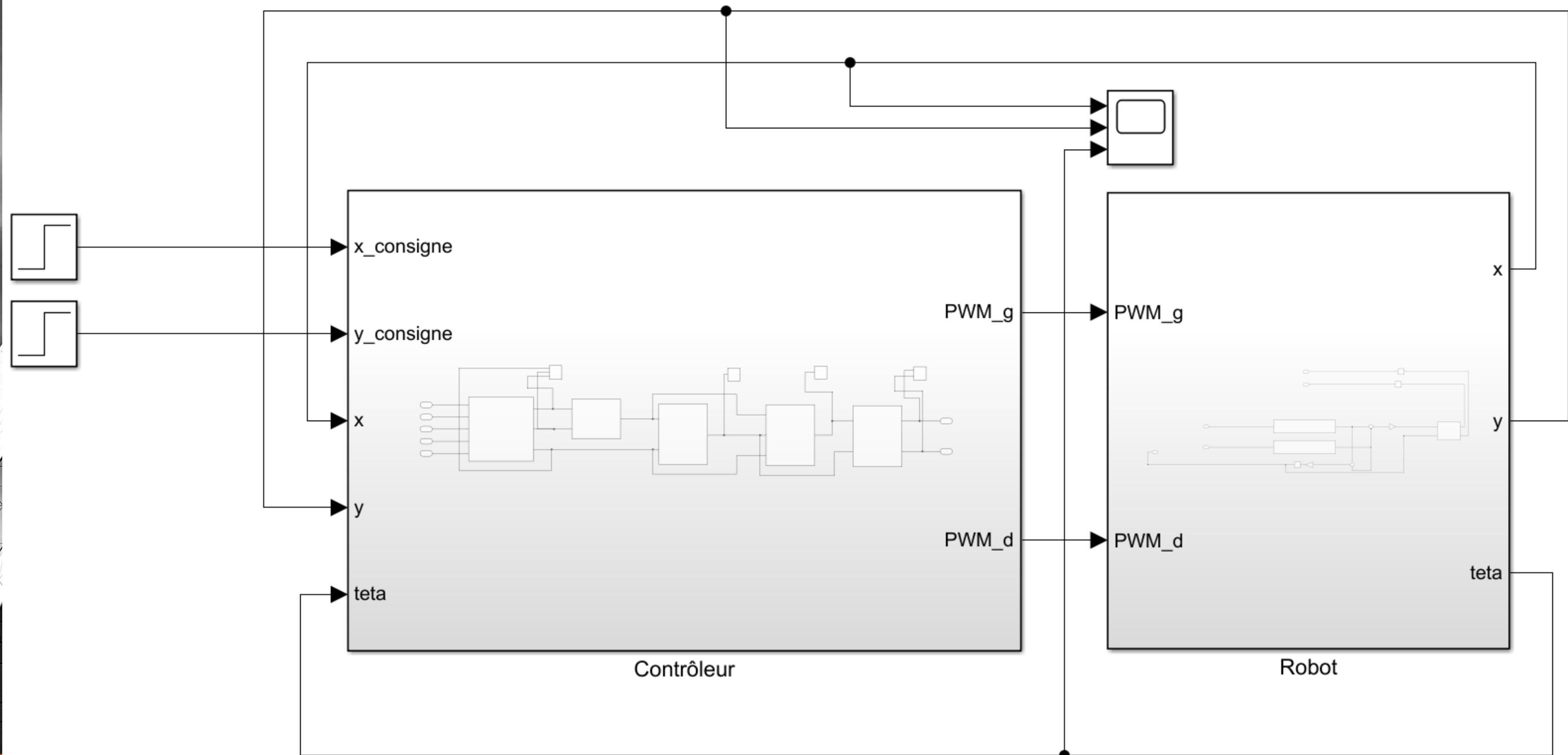
# Mesures

On fait avancer le robot en ligne droite et on trace  
 $v=f(t)$  :

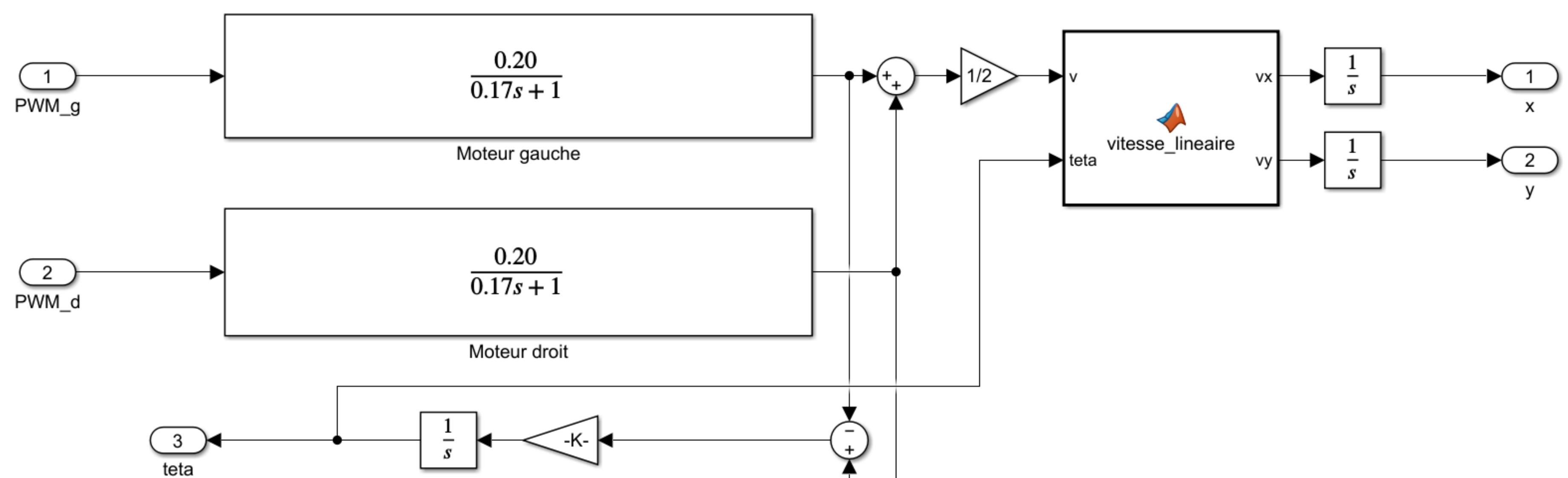
- La pente de la courbe correspond à  $K_s$
- Et tau :



# Solution finale adoptée



# Partie robot

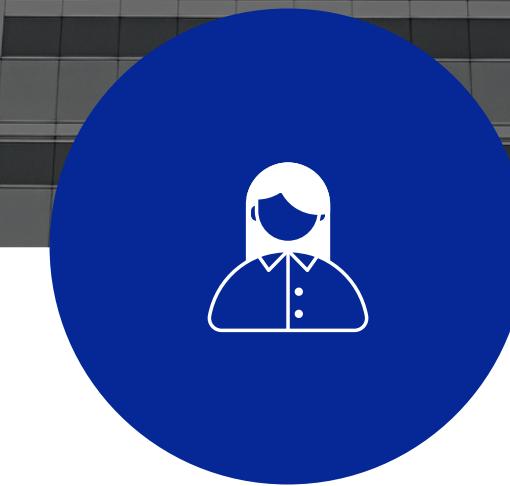


# Résultats

Après plusieurs avoir tester pour chaque valeur  $k_1$ ,  $k_2$  et  $k_3$  on obtient un résultat obtimal pour:

$k_1=0.3$ ,  $k_2= 1$  et  $k_3=1$

# PARTIE INFORMATIQUE



## Partie d'Idrissa

### Objectif

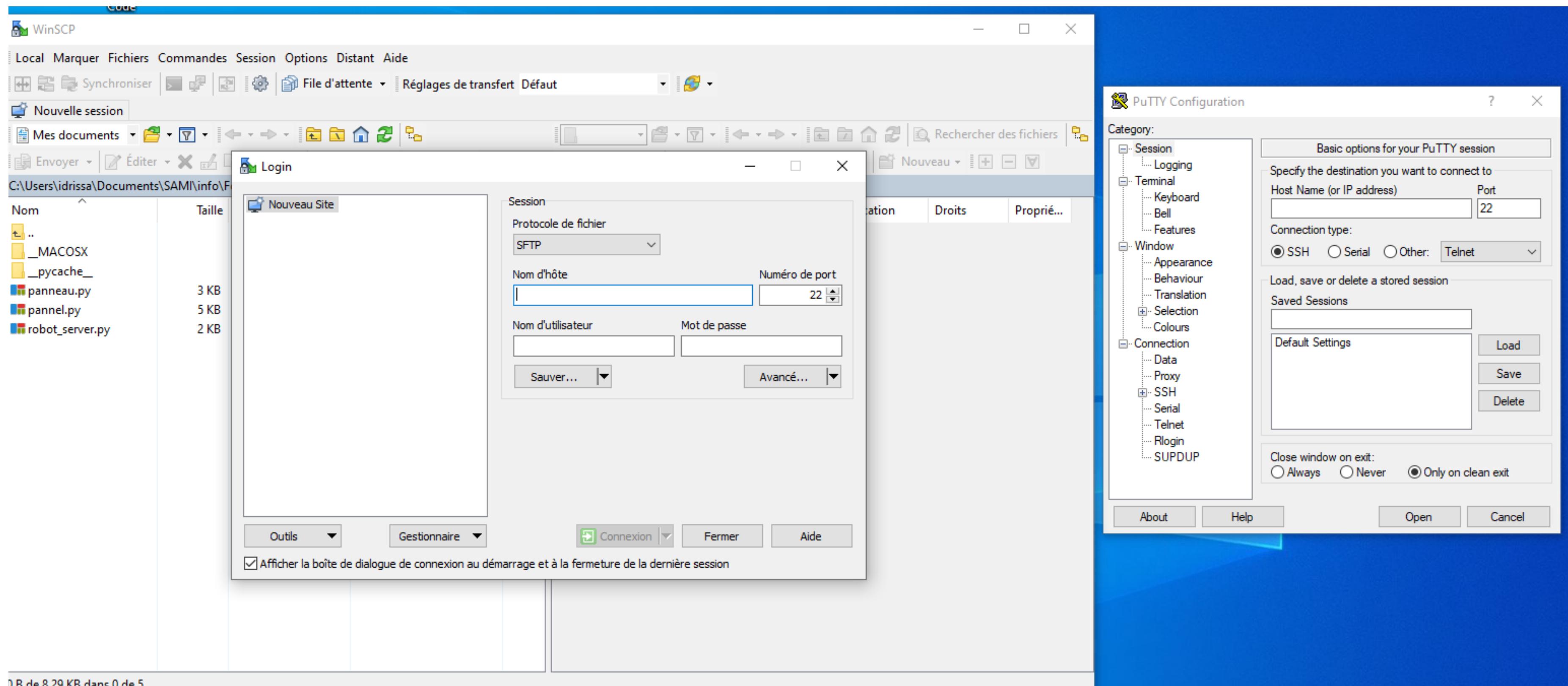
Faire parcourir (de façon optimale grâce aux algorithmes de la partie maths) au robot une liste de coordonnées de points en lui donnant les commandes nécessaires (fournies par la partie automatique).

### Démarche

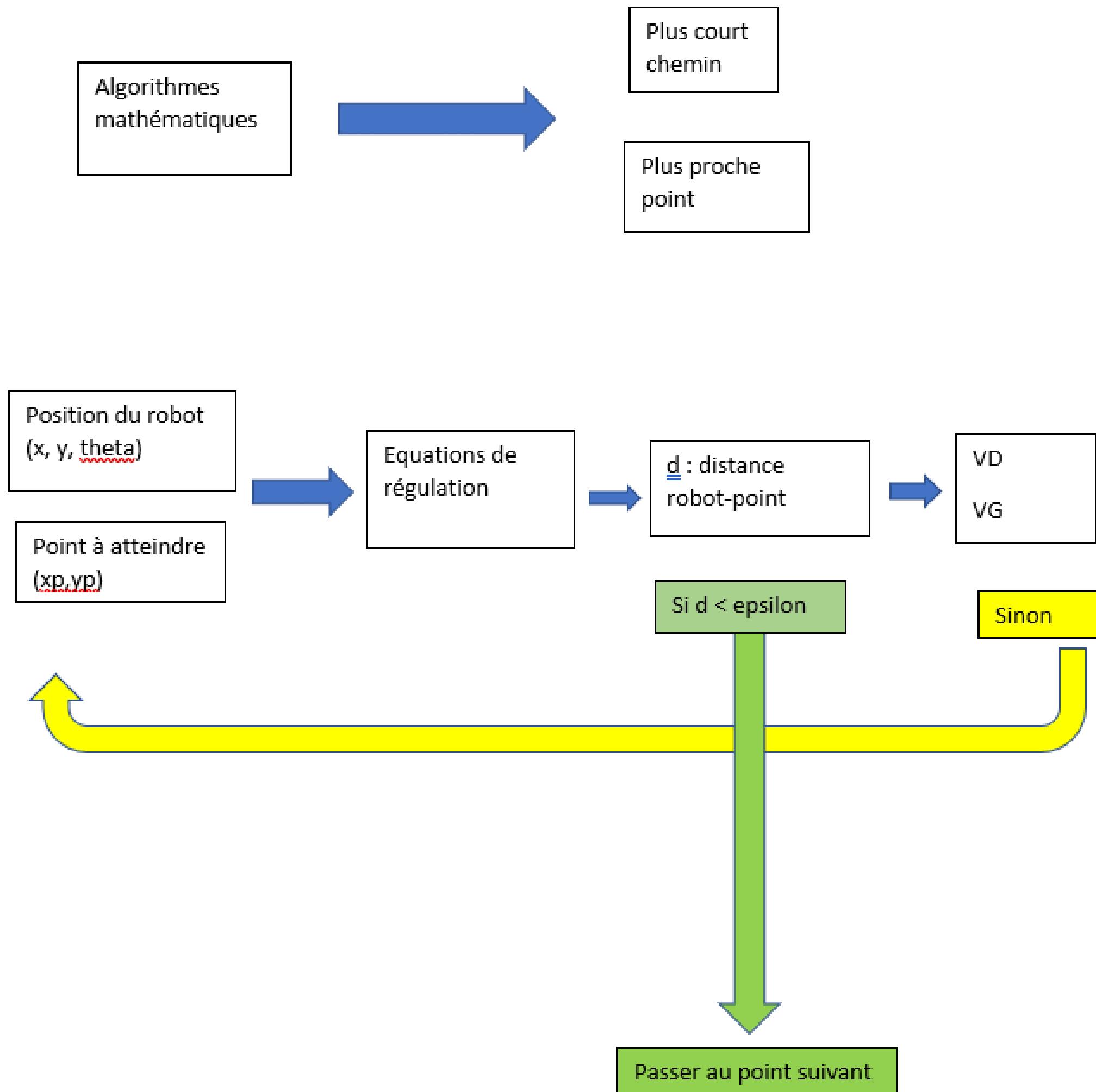
- Mise en place des programmes sur le robot grâce à PuTTy et WinSCP pour assurer la communication entre l'ordinateur et le robot
- Mécanisme de localisation d'un robot
- Interface graphique

# PuTTy et winSCP

## Communication avec le robot



# Synthèse



# Interface

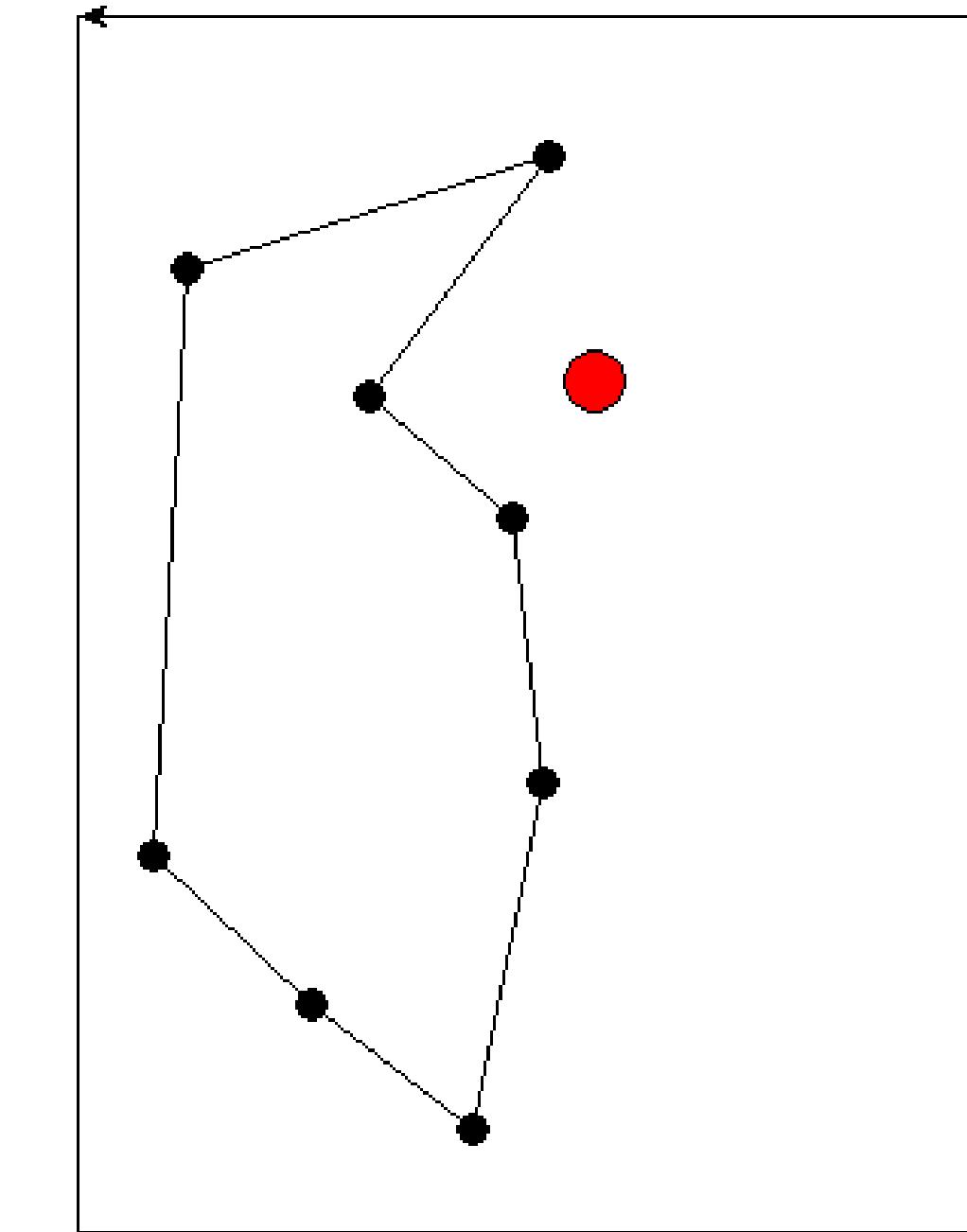
Bureau d'étude SAMI

Connect

Vitesse roue droite 0.00  
Vitesse roue gauche 0.00

Arrêter

Avancer



# Résultat

