LU2IN013 Groupe 3

Projet Robotique
Cours 2
Modélisation de l'arène, Géométrie, MVC

Nicolas Baskiotis
prenom.nom@sorbonne-universite.fr

Institut des systèmes intelligents et de robotique (ISIR)
Sorbonne Université

S2 (2024-2025)

Etat des lieux

N'oubliez pas !!

Principe Agile : au plus simple et démo à chaque sprint !

- On veut voir le projet avancé, pas attendre 1 mois pour savoir où vous en êtes
- Programmation itérative :
  - on développe des petites fonctionnalités
  - on ajoute au fur et à mesure du projet ce dont on a besoin

  $\Rightarrow$ oui, cela revient à repasser n fois sur le même code mais seulement le bout de code qui le nécessite
  - plus économe que de prévoir trop grand, trop complexe.
- "Voir" le projet $\Rightarrow$ le plus urgent ?

$\Rightarrow$ sortie texte (bof) ou graphique !

Modélisation arène, robot et physique

Robot

- A priori, le robot est simple mais quand même complexe à modéliser (formes, capteurs, roues, . . .)
- Simplifier au maximum au début !

$\Rightarrow$ Rectangle ou point qui se déplace, juste des coordonnées

Arène

- Beaucoup d'éléments peuvent être intégrés, réfléchir au strict nécessaire

$\Rightarrow$ Sol, murs, . . .

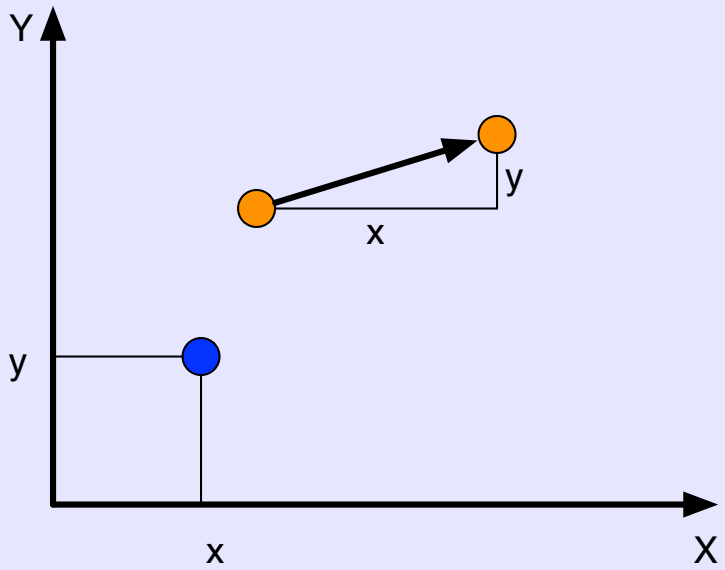
Physique

- On peut simuler les frottements, les roues non alignées, . . .
- Mais dans un premier temps . . . faire simple !

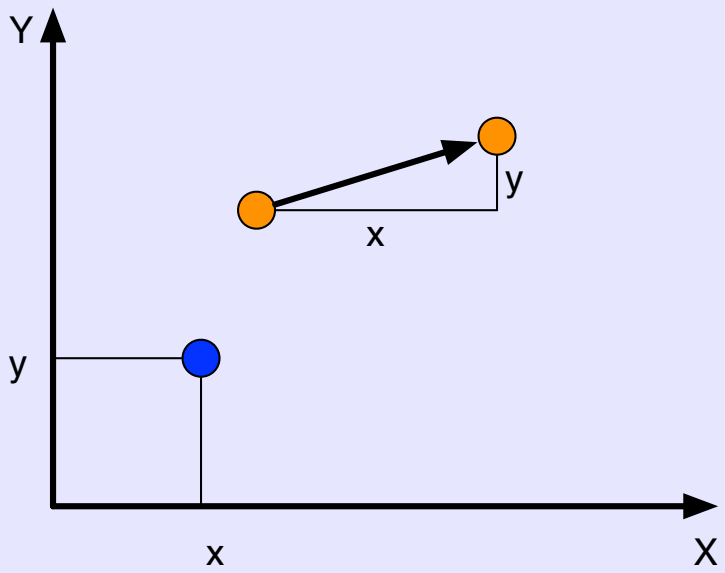
Plan

1. Géométrie
2. Exemple de senseur : senseur de distance
3. Model-View-Controller

Repérage dans l'espace 2D



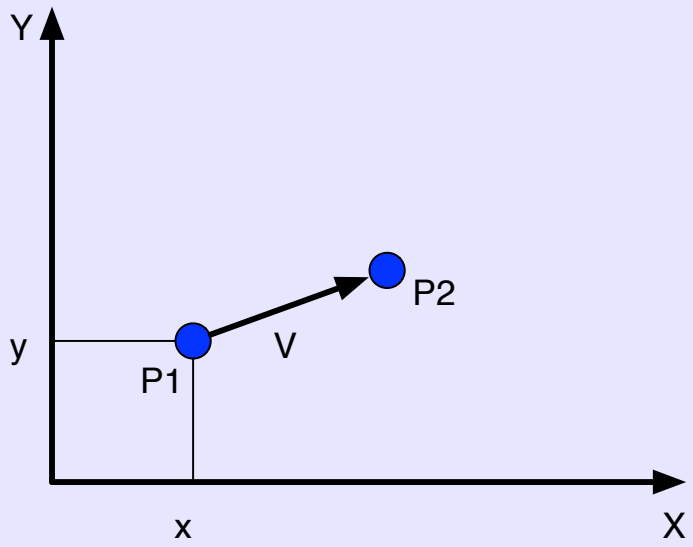
Repérage dans l'espace 2D



Faire une classe ou non ?

- Une classe alourdit le code (surtout en python)
- Mais permet de également de factoriser
  - Mêmes opérations sur les points et les vecteurs (translation, rotation, . . .)
- Toujours un compromis à trouver

Gestion des déplacements

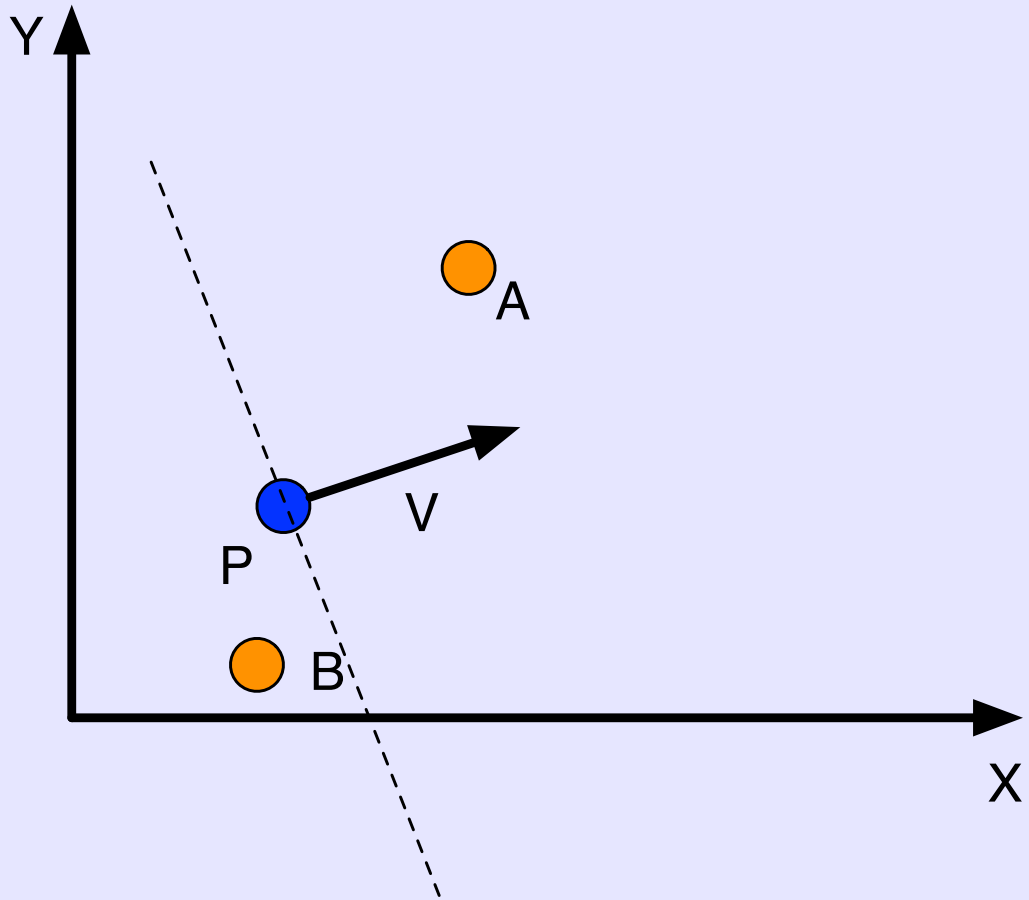


- Déplacements discrets (P: position, V: vitesse):

$$P_2 = P_1 + V, \quad \begin{cases} P_2.x = P_1.x + V.x \\ P_2.y = P_1.y + V.y \end{cases}$$

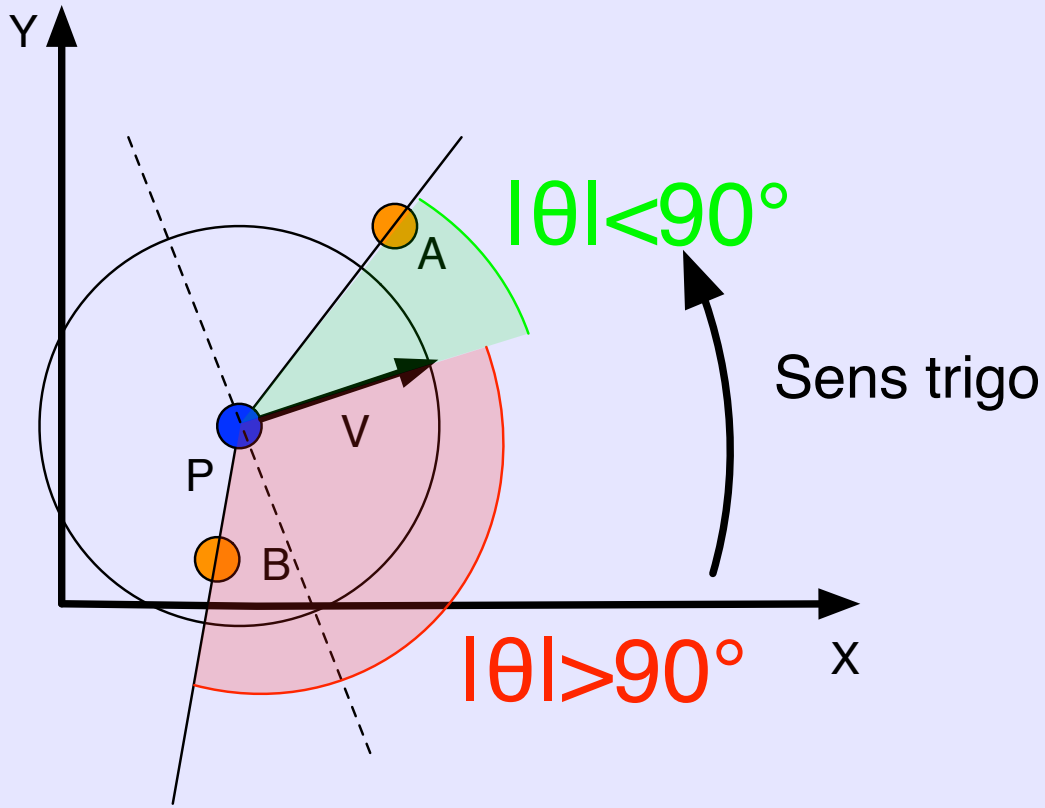
- En physique: $\vec{v} = \dot{\vec{x}} \approx \frac{\vec{x}_{t+1} - \vec{x}_t}{\delta_t}$, pour nous : $\delta_t = 1$ (unité arbitraire)
- En utilisant des vecteurs suffisamment petits: modélisation d'un déplacement continu

Se repérer dans l'espace



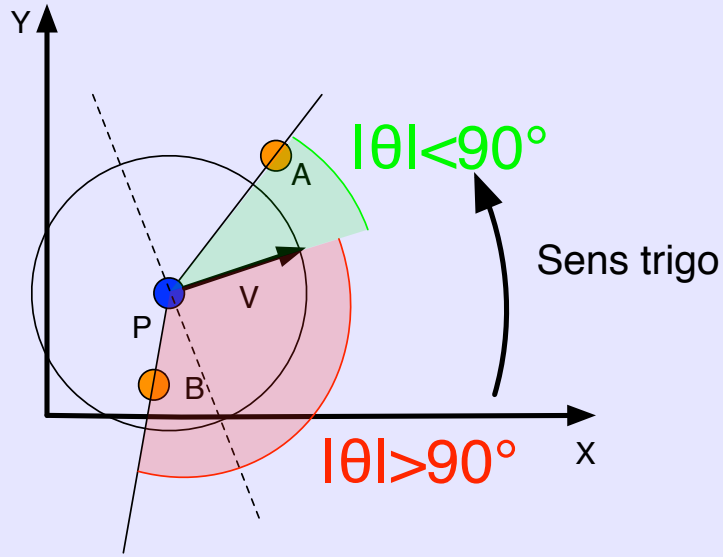
- Un objet est caractérisé par sa position P et sa vitesse V
- Qu'est ce qui est devant, qu'est ce qui est derrière l'objet?
- Qu'est ce qui est à droite, qu'est ce qui est à gauche?

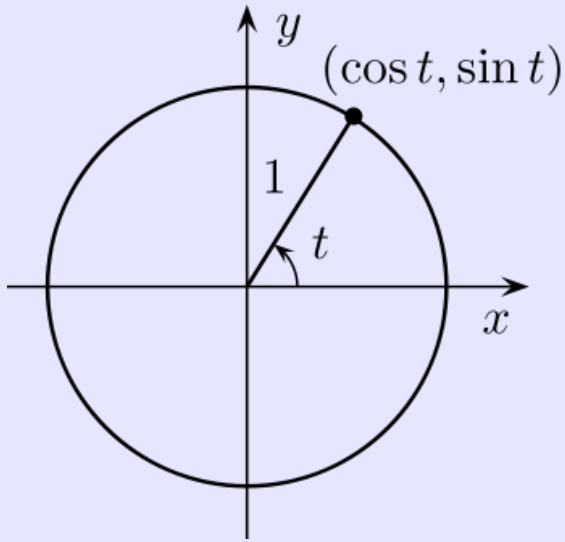
Solution complète: calcul d'angle



- Devant/Derrière: calculer les angles $\widehat{V, PA}$ et $\widehat{V, PB}$
- Gauche/Droite: calculer les angles $\widehat{V, PA}$ et $\widehat{V, PB}$ avec le signe !

Détail du calcul d'angle



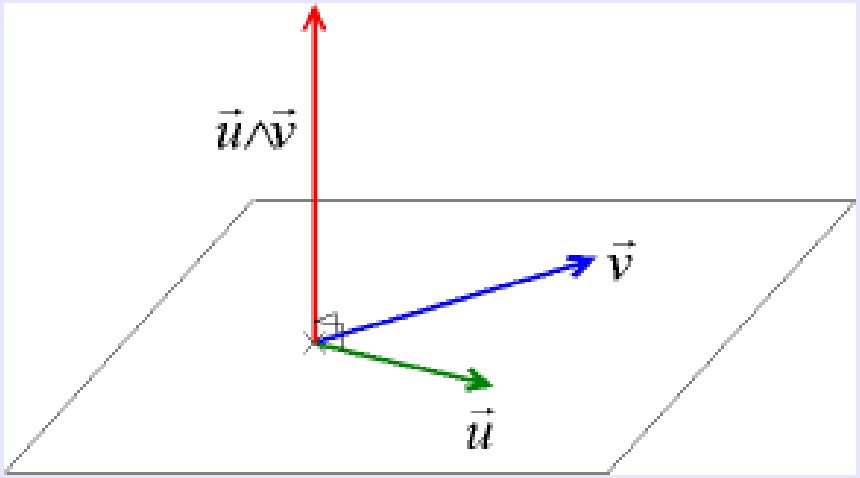


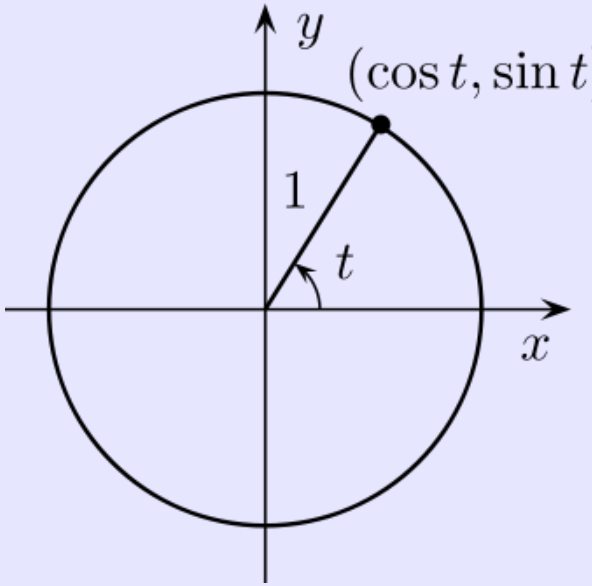
- Produit scalaire $U \cdot V = \langle U, V \rangle = \|U\|\|V\| \cos(\widehat{U, V}) = U_x V_x + U_y V_y$
- Corollaire:

$$\widehat{U, V} = \text{acos}\left(\frac{U \cdot V}{\|U\|\|V\|}\right)$$

- Attention: $\widehat{U, V} \in [0,\ \pi]$, pas de signe ici...
- Le signe de $U \cdot V$ permet de résoudre le pb devant/derrière

Détail du calcul d'angle (suite)





- Produit vectoriel: $\|U \wedge V\| = \|U\|\|V\| \sin(\widehat{U, V})$
- Corollaire:

$$\widehat{U, V} = \text{asin}\left(\frac{U \wedge V}{\|U\|\|V\|}\right) \qquad U \wedge V = \begin{bmatrix} u_2 v_3 - u_3 v_2 \\ u_3 v_1 - u_1 v_3 \\ u_1 v_2 - u_2 v_1 \end{bmatrix}$$

- L'étude de $u_1 v_2 - u_2 v_1$ permet de connaitre le signe de l'angle...

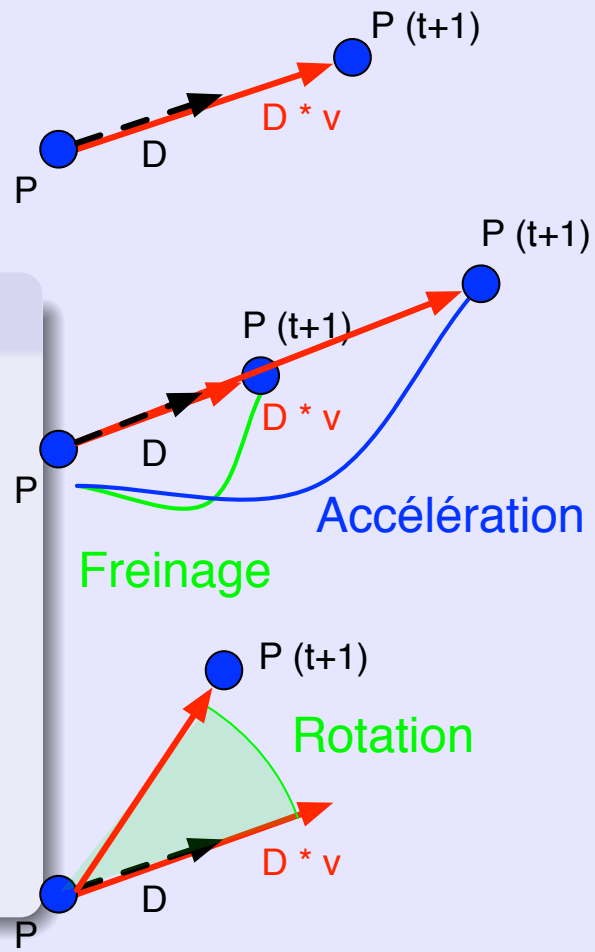
Une pré-définition possible du robot

Le robot peut être défini géométriquement par :

- Sa position: P
- Sa direction (vecteur unitaire) : D
  Conservation de la direction même à l'arrêt
- Sa vitesse (scalaire): $v \in [0,\ vmax]$

La commande du robot peut être sur 2 axes:

- Accélération/Freinage: modification de v
- Commande de direction: modification de D



Attention : définition différente de l'IRL

Est-ce un problème ?

Rotation sur un vecteur

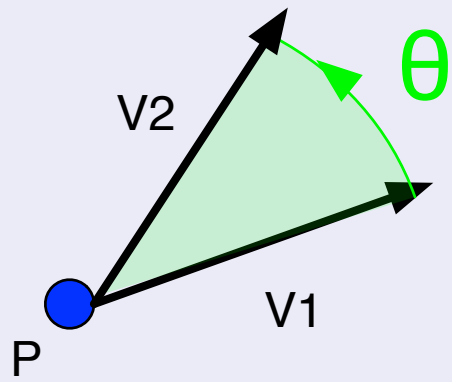
Définition de la rotation d'un vecteur:

Soit un vecteur $V = \begin{bmatrix} v_x \\ v_y \end{bmatrix}$, la rotation d'angle θ est obtenu en utilisant la matrice de rotation:

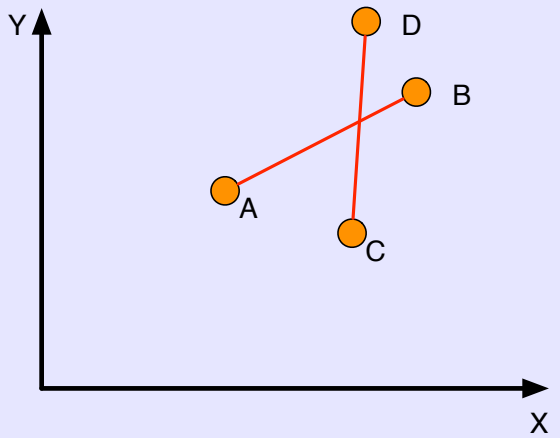
$$R = \begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix}, \quad V' = RV$$

C'est à dire en utilisant la mise à jour:

- $v'_x = v_x \cos(\theta) - v_y \sin(\theta)$
- $v'_y = v_x \sin(\theta) + v_y \cos(\theta)$



Approfondissement: collisions



(Problématique de base dans les cartes graphiques/moteur physique)

Comment détecter la collision de deux vecteurs?

- Si C et D sont à gauche et à droite de AB
- ET que A et B sont à gauche et à droite de CD

Résultat:

S'ils sont de part et d'autre, l'un des produit vectoriel est positif, l'autre négatif...

$$(AB \wedge AC)(AB \wedge AD) < 0 \text{ ET } (CD \wedge CA)(CD \wedge CB) < 0$$

Résumé des méthodes possibles de la classe `Vecteur`

- Addition, soustraction
  - génération d'un nouveau vecteur
  - auto-opérateur
- produit scalaire
- produit vectoriel (composante en z)
- multiplication par un scalaire
- rotation
- calcul de la norme
- clonage
- test d'égalité (structurelle)

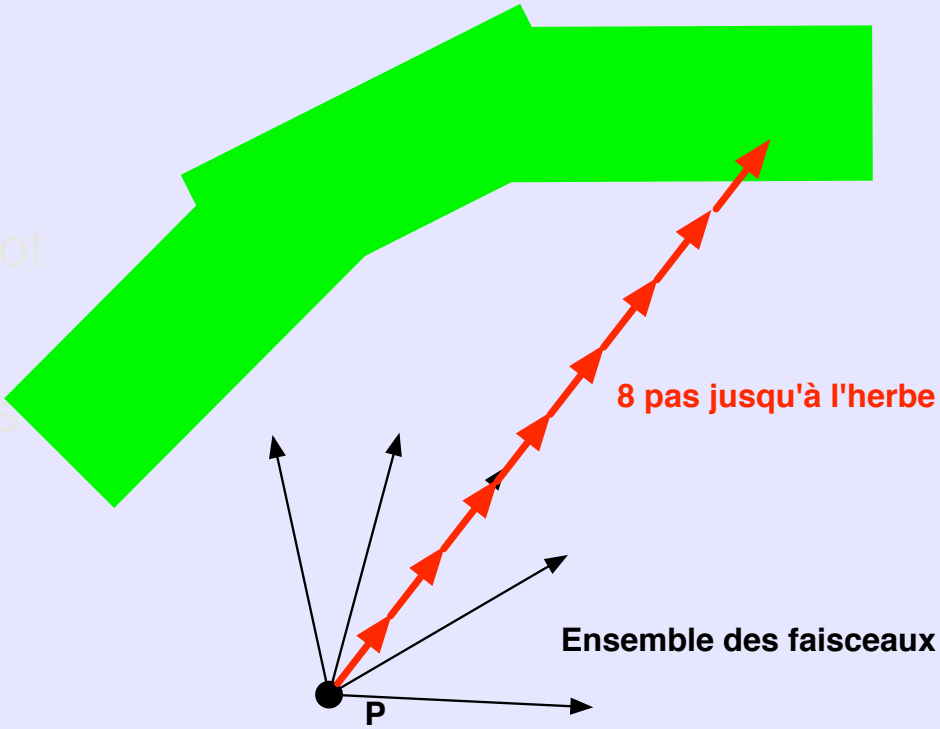
Plan

1. Géométrie
2. **Exemple de senseur : senseur de distance**
3. Model-View-Controller

Senseur de distance

- Permet de savoir la distance (bruitée) d'un obstacle devant le robot.
- Comment le faire algorithmiquement ?

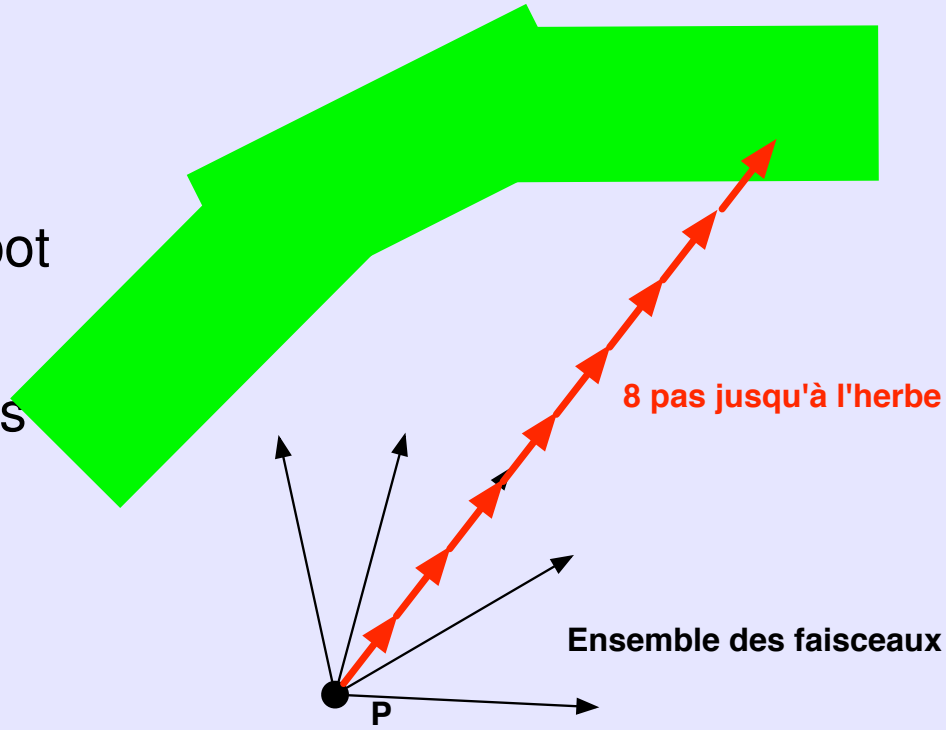
- Partir du robot
- Dans la direction D devant le robot
- Avancer d'un pas EPS
- Recommencer tant qu'il n'y a pas d'obstacle
- Renvoyer le nombre de pas effectués



Senseur de distance

- Permet de savoir la distance (bruitée) d'un obstacle devant le robot.
- Comment le faire algorithmiquement ?

- Partir du robot
- Dans la direction D devant le robot
- Avancer d'un pas EPS
- Recommencer tant qu'il n'y a pas d'obstacle
- Renvoyer le nombre de pas effectués



Le problème du bruit

- Dans le modèle
- Dans les données des capteurs

$\Rightarrow$ modélisation aléatoire
2 classes intéressantes

- `random`
  - bruit uniforme, tirage d'entier, tirage gaussien
- `numpy.random`
  - Gestion de matrice aléatoire
  - Tirage selon la plupart des lois usuelles

Plan

1. Géométrie
2. Exemple de senseur : senseur de distance
3. **Model-View-Controller**

Lasagne, spaghetti ou ravioli ?

- Spaghetti :
  - Code imbriqué, compliqué, pas (ou très peu) de classes
  - méthodes longues, et trop complexes.

  $\Rightarrow$ le pire ! à éviter
- Lasagne :
  - Code structuré, beaucoup de différentes classes bien réparties
  - mais changer une couche nécessite de changer toutes les couches (souvent le problème de trop de petites classes).

  $\Rightarrow$ Trop d'interdépendances, peu mieux faire
- Ravioli :
  - Bonne séparation des classes et des concepts, peut être réutilisable
  - chacun stand-alone, on peut changer indépendamment chaque classe

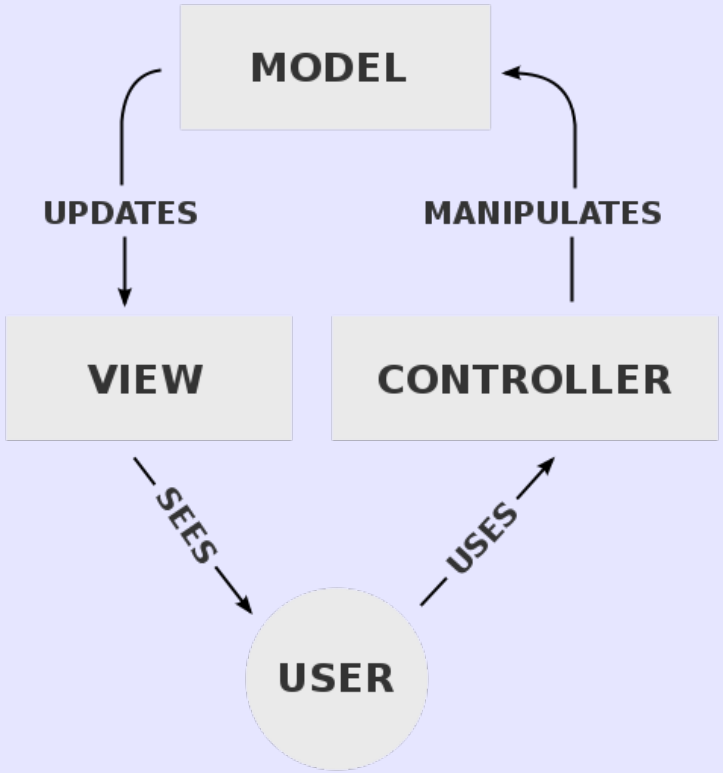
  $\Rightarrow$ Bien mais attention aux foisonnements de classes, il peut être difficile de comprendre les interactions.

Vers un système indépendant

- On veut un système flexible
  - Pouvoir brancher ou débrancher la simulation facilement sans intervenir dans le code
  - Pouvoir changer de système de visualisation
- Il existe un modèle standard pour gérer cette situation: MVC Model View Controller

- La vue est en charge de l'affichage du modèle.
- Pour chaque élément (robot, arène,...), la vue sait comment l'afficher.
- Le modèle est en charge de la simulation et de son fonctionnement.
- Le Contrôleur permet d'interagir avec la simulation (si besoin).

Dans le cadre du projet, vous pouvez considérer que le modèle est la simulation ou la réalité, le contrôleur le module de contrôle du robot.



credit: wikipedia