



VISION → MONDE RÉEL

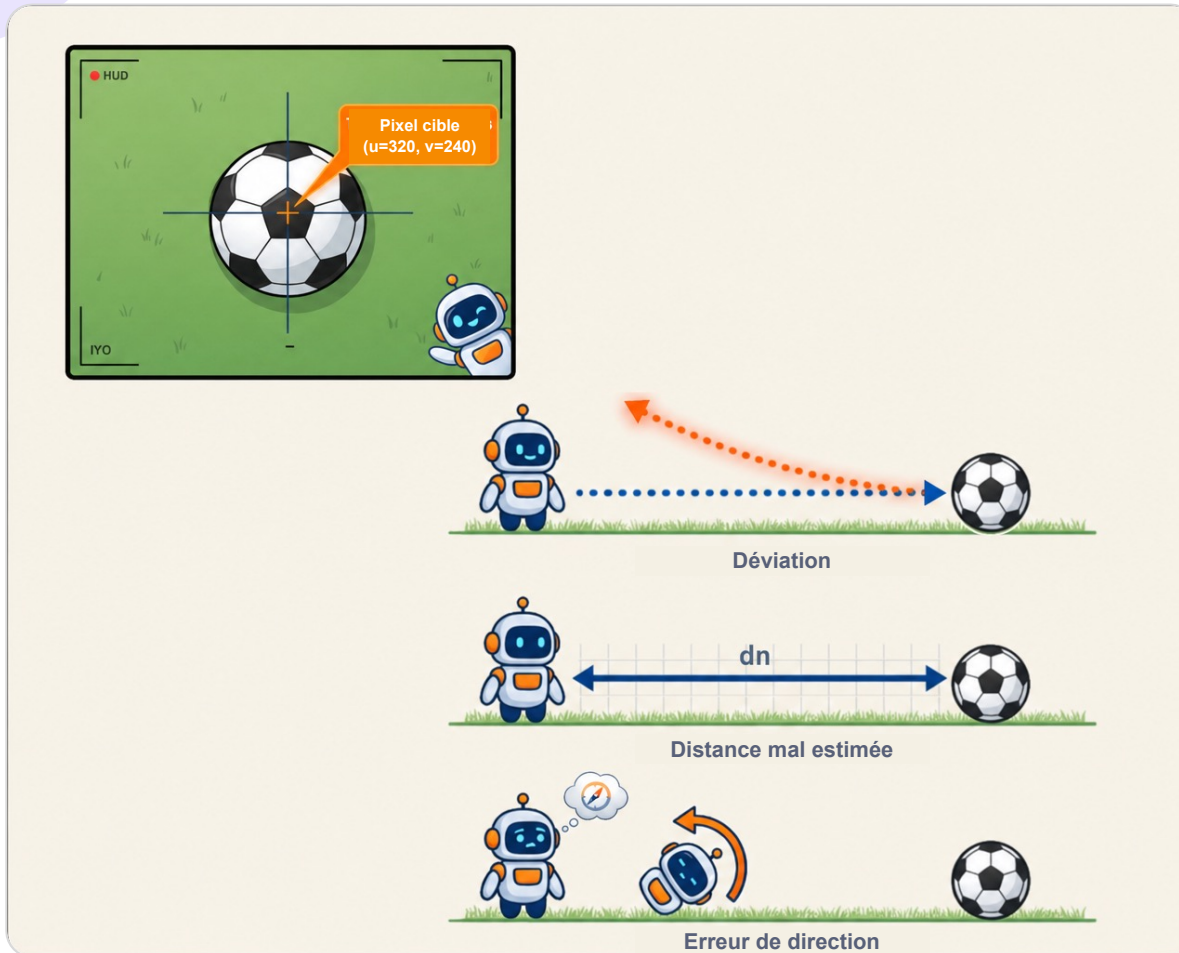
Transformations de coordonnées Du pixel au monde 3D

*Des coordonnées de l'image
au repère spatial du robot*

Modèle sténopé · Rétroprojection · Composition des repères ·
Intersection rayon-sol

Pourquoi le robot dévie-t-il de sa trajectoire ?

LEÇON 14 · Le robot a besoin d'une « règle spatiale »



D'où vient le problème ?

● Déviation de la trajectoire

L'orientation de la caméra diffère de la direction de marche. Se diriger vers le centre du pixel peut donc éloigner le robot de la cible.

● Mauvaise estimation de la distance

L'image indique la présence du ballon et sa direction, mais pas sa distance.

● Erreur de direction

Les repères caméra, tête et base ne sont pas alignés : la droite de l'image n'est pas forcément celle du corps.

La question centrale de cette leçon

À partir d'un seul pixel (u, v) , comment déterminer dans quelle direction et à quelle distance se trouve le ballon par rapport aux pieds du robot ?

Objectifs · Le voyage d'un pixel

Du pixel à la position



01

Comprendre le modèle

Comprendre le modèle sténopé.
Un pixel = un rayon dans l'espace.

02

Maîtriser les repères

Caméra / Tête / Base :
le rôle des trois repères.

03

Écrire les transformations

Composition des matrices :
 $T_{B \leftarrow C} = T_{B \leftarrow H} \cdot T_{H \leftarrow C}$

04

Valider par l'usage

Intersection rayon-sol :
le robot s'arrête juste devant le ballon.

1. L'œil du robot

Une caméra n'est pas un instrument de mesure des distances

Que capture réellement une caméra ?

Une caméra produit une image 2D, mais le monde réel est en 3D.

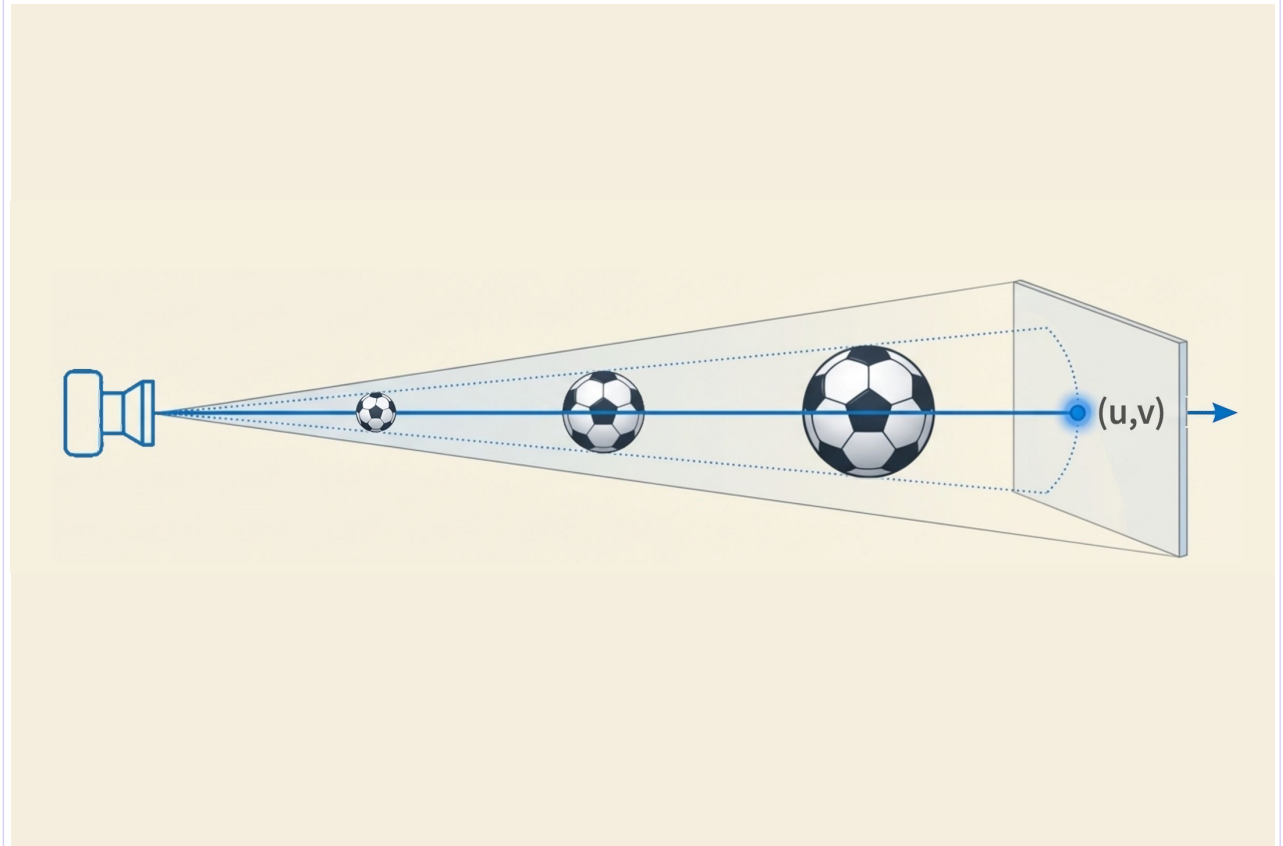
Cela signifie que :

- l'image ne fournit pas directement la distance ;
- chaque pixel représente seulement une direction.

La véritable nature d'un pixel

Un pixel (u, v) correspond à un rayon dans l'espace, partant du centre de la caméra. Il indique dans quelle direction regarder, mais pas à quelle distance.

Cône de vision · Un même pixel correspond à une infinité de points dans l'espace



2. De la 3D à la 2D · Le modèle sténopé

Modèle de caméra sténopé · Triangles semblables

Le modèle de caméra le plus simple

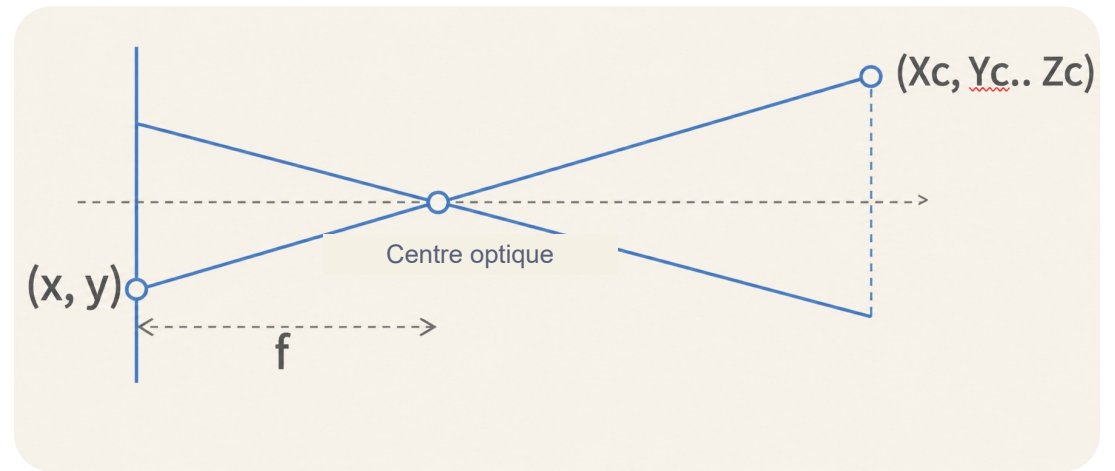
Il décrit la formation de l'image à l'aide d'un minuscule trou idéal : le sténopé.

Malgré sa simplicité, il exprime la relation géométrique essentielle de la formation des images et sert de base aux modèles de caméra plus complexes.

Principe essentiel

Utiliser les triangles semblables pour projeter un point de l'espace 3D sur le plan image 2D.

Triangles semblables · Focale f · Image (x, y) · Point 3D (X_c, Y_c, Z_c)



Formule clé : la projection perspective

Pour un point 3D (X_c, Y_c, Z_c) , les coordonnées projetées sur le plan image sont :

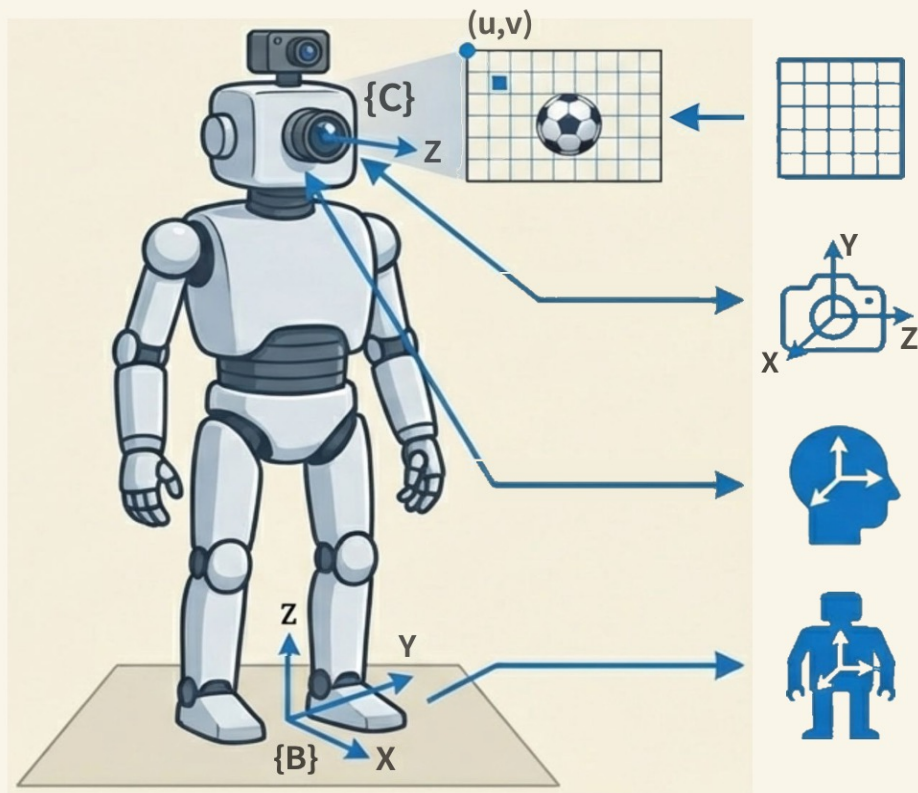
$$x = f \cdot X_c / Z_c \qquad y = f \cdot Y_c / Z_c$$

Point crucial : on divise par la profondeur Z_c . L'information 3D est « comprimée » ; notre objectif est de la retrouver.

3. Quatre repères de coordonnées

Définir une « carte » commune pour la formation de l'image

Les quatre repères du robot



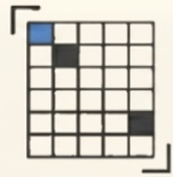
Repère	Notation	Rôle
Image	(u, v)	Coordonnées en pixels sur l'écran. L'origine est en haut à gauche.
Caméra	$\{C\}$ X_c Y_c Z_c	Origine au centre optique de la caméra. L'axe Z pointe vers l'avant.
Tête	$\{H\}$	Solidaire de la tête, il tourne avec elle et relie la caméra au corps.
Base	$\{B\}$	Situé au niveau des pieds, il constitue le « monde » du robot et le repère de commande des déplacements.

En une phrase

Du pixel aux coordonnées de base, les quatre repères se transmettent les données dans l'ordre.

4. Les étapes de la transformation

Pixel $\rightarrow$ Caméra $\{C\}$ $\rightarrow$ Tête $\{H\}$ $\rightarrow$ Base $\{B\}$



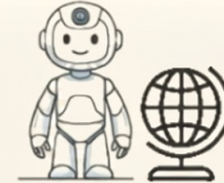
(u, v)
Coordonnées
en pixels



$\{C\}$
Repère de
la caméra



$\{H\}$
Repère
de la tête



$\{B\}$
Repère
de la base

1 Pixel $\rightarrow$ Caméra

Rétroprojection : retirer les paramètres intrinsèques

Étape 1 · Transformer (u, v) en un rayon dans le repère caméra.

Données requises : f_x, f_y, C_x, C_y .

2 Caméra $\rightarrow$ Tête $\rightarrow$ Base

Composition des matrices de paramètres extrinsèques

Étape 2 · Deux transformations :

$T_{H \leftarrow C}$: statique, issue du fichier de configuration.

$T_{B \leftarrow H}$: dynamique, issue de `/head_pose`.

3 Rayon $\rightarrow$ Position réelle

Intersection avec le plan du sol $Z_B = 0$

Étape 3 · Le ballon est au sol : cette seconde contrainte permet de calculer x et y , à l'intersection du rayon avec le sol.

5. Théorie · Rétroprojection du modèle sténopé

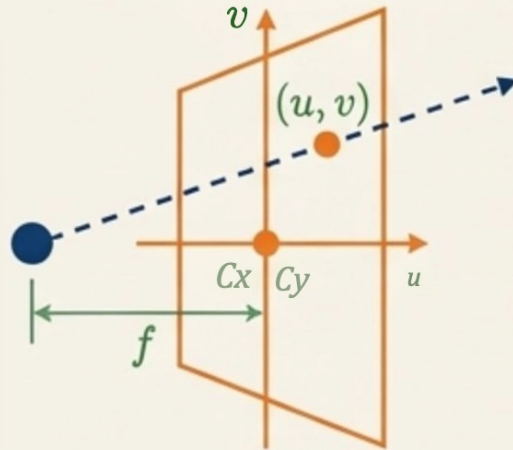
Étape 1 · Retirer les paramètres intrinsèques pour obtenir la « direction normalisée »

Pixel $\rightarrow$ Direction normalisée $(x, y, 1)$

$$x_n = (u - c_x) / f_x$$

$$y_n = (v - c_y) / f_y$$

$$d_C = [x_n, y_n, 1]^T$$



Principe essentiel

Retirer l'effet des paramètres intrinsèques :

- f_x et f_y définissent l'échelle ;
- C_x et C_y définissent le centre de l'image.

Le vecteur $(x, y, 1)$ obtenu indique une direction dans le repère caméra.

Programme : pixel_to_3d_node.py Extrait principal

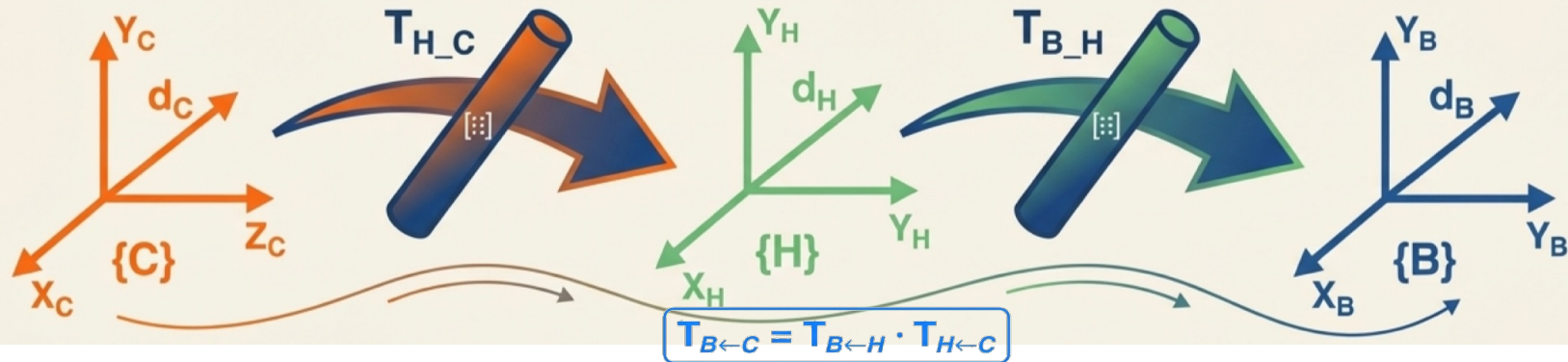
```
# Étape 1 : pixel -> direction normalisée
# Rétroprojection (modèle sténopé) :
#   x_norm = (u - cx) / fx
#   y_norm = (v - cy) / fy
#   z_norm = 1
# On obtient une direction, pas une position.
x_norm = (u - cx) / fx
y_norm = (v - cy) / fy
z_norm = 1.0

# Direction du rayon dans le repère caméra
ray_cam = np.array([x_norm, y_norm, z_norm],
                   dtype=np.float32)
```

Note de traduction : coquille corrigée dans la formule de y_n , conformément au code : $(v - cy) / fy$.

6. Caméra → Tête → Base · Composition des matrices

Statique et dynamique : deux transformations de coordonnées



$T_{H \leftarrow C}$ · Caméra → Tête

Paramètres extrinsèques statiques

- La caméra est fixée à la tête : sa pose relative est constante.
- La matrice est étalonnée une fois, puis réutilisée.
- Elle provient du fichier camera_extrinsics.yaml.

$T_{B \leftarrow H}$ · Tête → Base

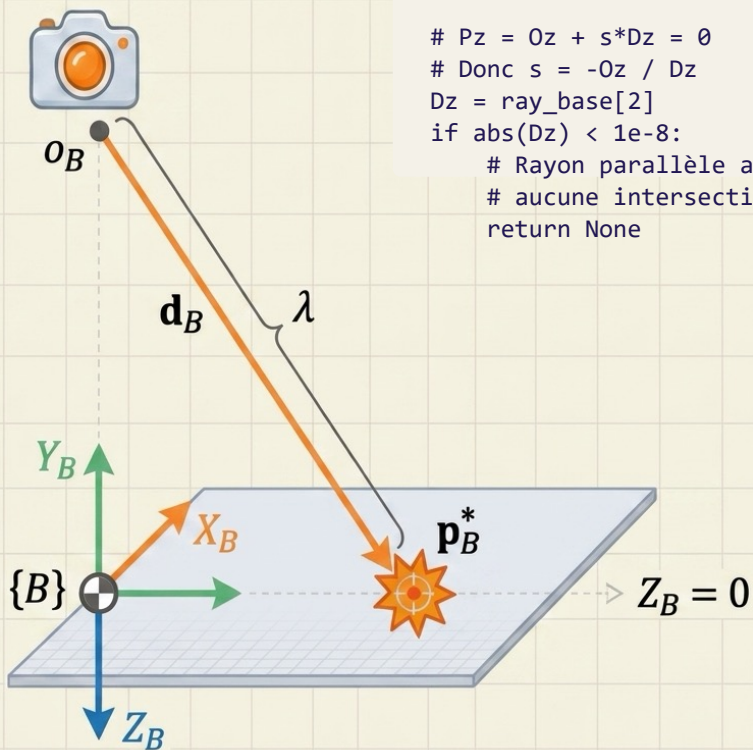
Paramètres extrinsèques dynamiques

- La tête bouge : elle s'incline et tourne à gauche ou à droite.
- La matrice évolue à chaque instant.
- Elle provient du topic /head_pose, en temps réel.

7. Arrivée à destination · Le rayon rencontre le sol

Principe · Retrouver la distance grâce à une hypothèse physique

Intersection avec le sol ($z_B = 0$) · Théorie et mise en œuvre



```
# Pz = Oz + s*Dz = 0
# Donc s = -Oz / Dz
Dz = ray_base[2]
if abs(Dz) < 1e-8:
    # Rayon parallèle au sol :
    # aucune intersection
    return None
```

La contrainte physique supplémentaire

Données

Centre optique de la caméra (origine) : o_B

Direction du rayon : d_B

Contrainte

Le ballon est au sol $\rightarrow z_B = 0$

Résolution

1. Calculer le paramètre :

$$\lambda = -o_{Bz} / d_{Bz}$$

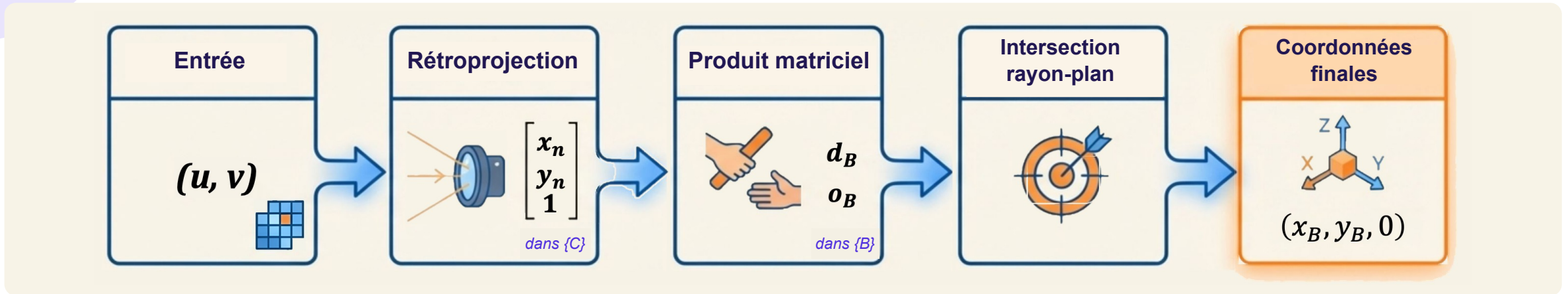
2. En déduire la position finale :

$$p_B^* = o_B + \lambda \cdot d_B$$

Intuition : déterminer jusqu'où le rayon doit aller (la valeur de λ) pour atteindre le plan du sol $z = 0$.

8. Chaîne complète · Du pixel à la position

Pixel → Rétroprojection → Composition → Intersection rayon-sol → Base



Programme : pixel_to_3d_node.py · Fonctions principales

- 1 `image_callback(u, v)` : recevoir le pixel cible
- 2 `pixel_to_normalized` : retirer les intrinsèques → d_C
- 3 `T_B_C = T_B_H · T_H_C` : composer les matrices
- 4 `ray_ground_intersect` : intersection avec $Z_B = 0$
- 5 `publish(Point: x, y, 0)` : publier dans le repère de base

Objectif atteint !

Le robot sait enfin où se trouve le ballon dans le monde réel.

Ses actions peuvent désormais s'exprimer ainsi : « avancer vers (x, y) ».



9. En pratique · Les rotations par quaternions

Inutile de redémontrer les formules : utiliser une bibliothèque scientifique éprouvée

En théorie

$$\mathbf{R}(\mathbf{q}) = \begin{bmatrix} 1 - 2(y^2 + z^2) & 2(xy - zw) & 2(xz + yw) \\ 2(xy + zw) & 1 - 2(x^2 + z^2) & 2(yz - xw) \\ 2(xz - yw) & 2(yz + xw) & 1 - 2(x^2 + y^2) \end{bmatrix}.$$

En pratique

```
from scipy.spatial.transform import Rotation as R

# q = [x, y, z, w] issu du message ROS
# Une seule ligne effectue tous les calculs !
rotation_matrix = R.from_quat(q).as_matrix()
```



En théorie

Un quaternion $q = (w, x, y, z)$ permet de décrire une rotation 3D :

- pas de blocage de cardan ;
- chaque quaternion unitaire correspond à une rotation ;
- quatre nombres décrivent une orientation 3D.

En pratique

```
from scipy.spatial.transform import Rotation as R

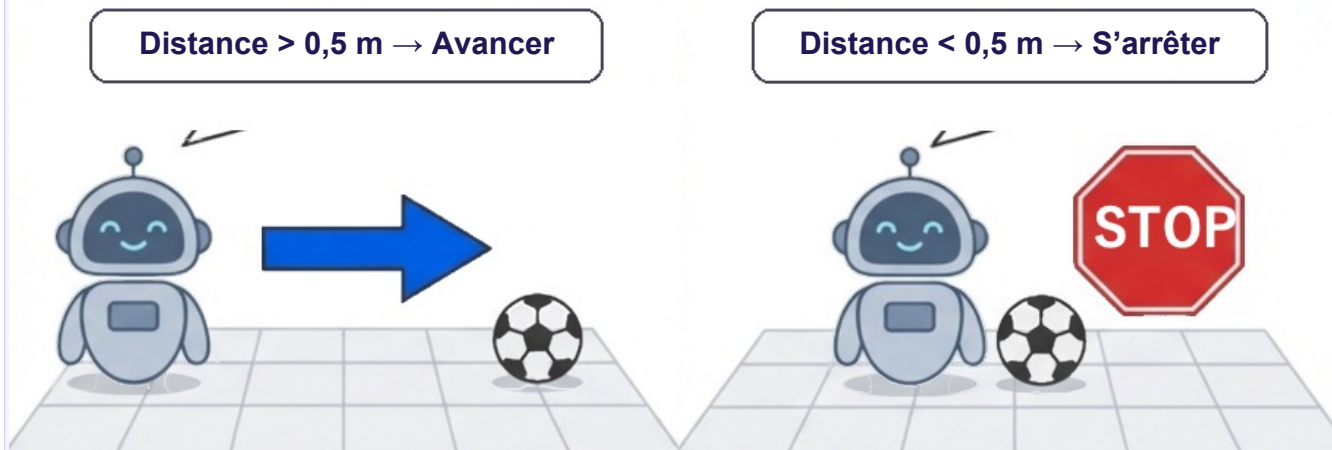
rot = R.from_quat([x, y, z, w])

mat = rot.as_matrix() # Matrice de rotation 3x3
```

10. Vérifier la compréhension de l'espace

Programme de référence : lesson14_stand_position.py

Comportement clé : ajuster la position du robot selon la distance réelle au ballon



Pourquoi ne pas frapper le ballon ?

Cette leçon ne cherche pas à démontrer les capacités de mouvement du robot : elle vérifie que sa compréhension de l'espace est correcte.

Le robot effectue donc une seule tâche : ajuster sa position selon la distance réelle au ballon.

Critère de validation

Si le robot s'arrête précisément devant le ballon, la chaîne de transformation fonctionne !

Pour un dernier étalonnage, placer une règle au sol et comparer les mesures aux coordonnées x, y, z.

11. Architecture du code

Un système complexe en trois couches : perception / décision / exécution

Principe de conception

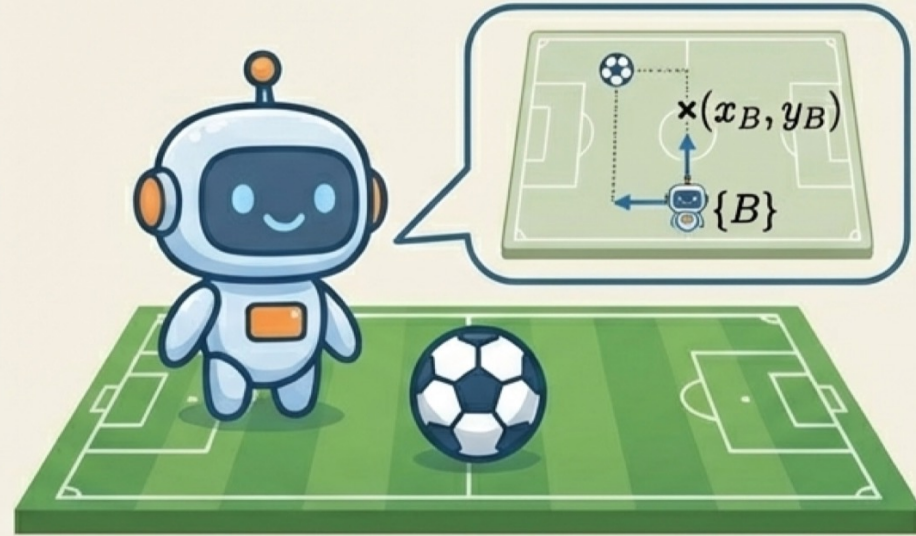
Un système complexe ne repose pas sur un programme unique qui fait tout : il associe plusieurs modules, chacun doté d'un rôle clair, organisés en couches distinctes.

Architecture du programme · Perception / Décision / Exécution

	Couche	Fichier du programme	Rôle
Perception	Perception	ball_detection_publish.py	« Voir » : publier la position du ballon
Décision	Décision	pixel_to_3d_node.py	« Réfléchir » : convertir les coordonnées
Exécution	Exécution	lesson14_stand_position.py	« Agir » : avancer vers la cible

Bilan · Voir, puis comprendre

Voir → Comprendre : le robot possède désormais une représentation de l'espace



Voir

Obtenir les pixels (u, v)

Caméra → Image → Algorithme de vision

Comprendre

Retrouver la position réelle (x, y)

Transformation des coordonnées + contrainte du sol

Décider

Décider dans l'espace réel

Les coordonnées (x, y) déterminent la direction et la distance à parcourir.

Récapitulatif · La prochaine leçon

Bilan et perspectives

Les trois étapes de cette progression

1

Voir la cible

Obtenir les coordonnées en pixels (u, v).

2

Comprendre sa position réelle

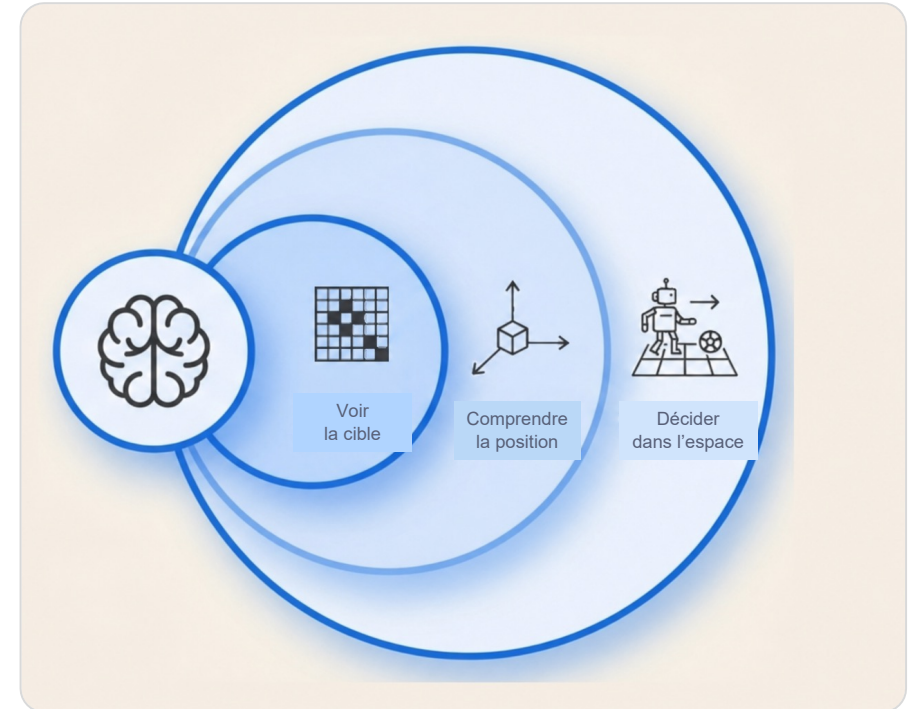
Calculer les coordonnées de base (x, y).

3

Décider dans l'espace réel

Exécuter les actions selon la position (x, y).

Nous avons appris à retrouver des informations 3D à partir d'une image 2D et découvert une compétence fondamentale en robotique : transformer les coordonnées.



Prochaine leçon · Dernière étape du L4

Objectif final du parcours : le robot poursuit le ballon et frappe dedans de façon autonome !

À TRÈS BIENTÔT

Rendez-vous au prochain cours !

*Prochaine leçon : poursuite du ballon et tir autonomes
(dernière étape du L4)*

Booster Robotics · IA incarnée au lycée · Module L4