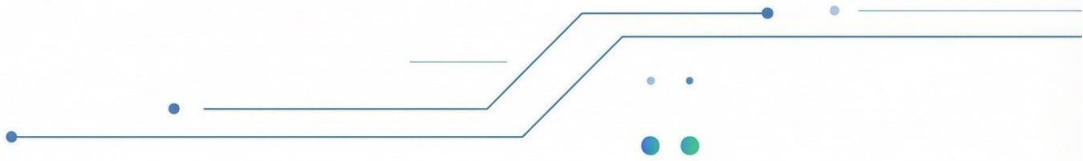


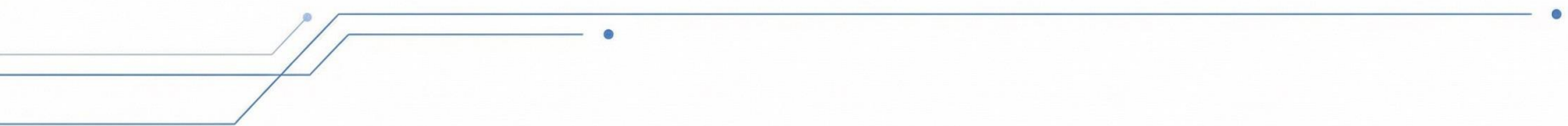
BOOSTER



LA ROBOTIQUE HUMANOÏDE EN CLASSE

L'intelligence incarnée, enseignée avec méthode

De la programmation visuelle à la décision dans le monde réel.



BOOSTER

Plan du cours

Partie 1 - Les fondamentaux

- Introduction à la robotique humanoïde
- Bases de la programmation visuelle
- Intégration des capteurs
- Mouvements et commandes de base
- Prise de décision simple

Partie 2 - Niveau avancé

- Transformation des coordonnées
- Détection visuelle
- Coordonnées dans l'image
- Comprendre ROS 2



BOOSTER

Cours général de robotique humanoïde

**Détection du ballon et suivi par la tête
De la vision à l'action en boucle fermée**



**Vue
d'ensemble**

L'intelligence incarnée en boucle fermée

Perception

Les capteurs
recueillent
les données de
l'environnement.



Décision

Analyser les données
et planifier
les actions.



Action

Exécuter les actions
prévues, par exemple
commander les moteurs.



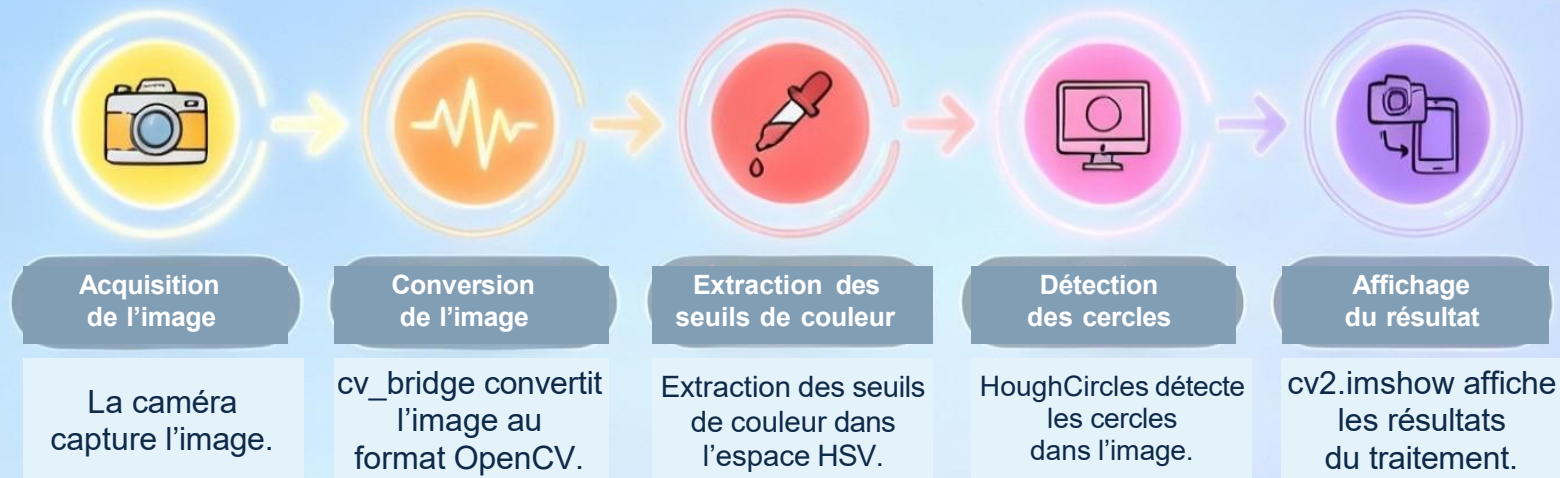
- Perception : recueillir les données de l'environnement.
- Décision : analyser les données et planifier les actions.
- Action : exécuter les actions prévues (commande des moteurs).



1. Rappel des connaissances



Séance précédente : le robot « voyait » le ballon, sans pouvoir transmettre l'information.



Caractéristiques du programme :

- Traitement d'image uniquement avec OpenCV.
- Affichage des résultats sans communication avec le système.
- Programme de vision indépendant (fonctionnement autonome).



Limite : le robot voit le ballon, mais ne peut pas faire réagir sa tête ou son corps.

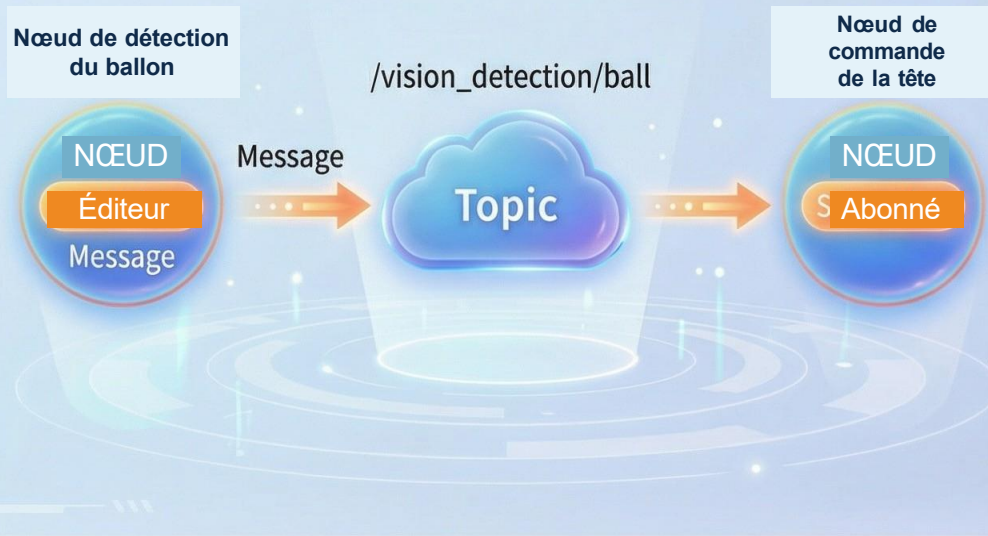


2. Rappel : communiquer par topics dans ROS 2

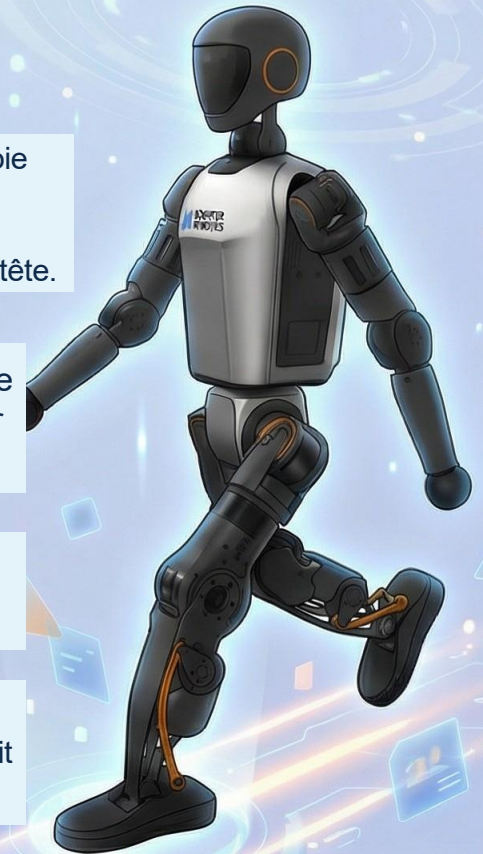


Faire « parler » le robot : le mécanisme des topics ROS 2

Les résultats de détection visuelle servent à d'autres modules : commande de la tête, planification de trajectoire, choix des mouvements, etc. Le nœud de détection doit donc diffuser les coordonnées. ROS 2 utilise pour cela les topics : éditeur (Publisher) → topic → abonné (Subscriber).



- ✓ Le nœud de détection envoie ses résultats aux autres modules, par exemple au nœud de commande de la tête.
- ✓ La communication repose sur le mécanisme éditeur / abonné de ROS 2.
- ✓ Nœud de vision : publication sur /vision_detection/ball
- ✓ Le nœud de la tête s'abonne à ce topic et reçoit les coordonnées du ballon.



3. Logique du nœud éditeur



Des résultats de détection à la diffusion des messages



Capture de l'image
La caméra acquiert les données de l'image.



Couleur et forme
Analyser l'image pour les détecter.



Données JSON
Mettre les résultats au format JSON.



Publication ROS 2
Publier le JSON sur un topic ROS 2.

Module	Ajout au programme	Objectif
Éditeur ROS 2	<code>create_publisher(String, '/vision_detection/ball', 10)</code>	Créer le nœud éditeur
Construction JSON	<code>msg = {"x": cx, "y": cy, "conf": confidence}</code>	Regrouper les résultats
Publication	<code>publish(String(data=json.dumps(msg)))</code>	Diffuser les coordonnées
Journalisation	<code>print("[INFO] Published:", msg)</code>	Débuguer et vérifier

4. Où se trouve le code correspondant ?



À chaque image reçue :

1. Convertir l'image ROS au format OpenCV.
2. Détecter le ballon rouge avec detect_red_ball.
3. Publier les résultats sous forme de chaîne JSON.

```
def image_callback(self, msg):
    try:
        cv_image = self.bridge.imgmsg_to_cv2(msg, "bgr8")
    except CvBridgeError as e:
        self.get_logger().error(f"CvBridge Error: {e}")
        return

    result = detect_red_ball(cv_image)

    if result:
        cx, cy, confidence = result
        msg_dict = {"x": cx, "y": cy, "conf": confidence}
        json_str = json.dumps(msg_dict)

        self.publisher_.publish(String(data=json_str))
        print(f"[INFO] Published: {msg_dict}")
    else:
        print("[WARN] No red ball detected")

# Fenêtre de visualisation pour la démonstration
cv2.imshow("Detection Result", cv_image)
cv2.waitKey(1)
```



 Code de référence : ball_detection_publish.py

```
self.publisher_ = self.create_publisher(String,
    '/vision_detection/ball', 10)
self.subscription = self.create_subscription(Image,
    '/camera/image_raw', self.image_callback, 10)

self.get_logger().info("Node started")
self.get_logger().info("Publishing topic:
    /vision_detection/ball")
self.get_logger().info("Subscribing topic:
    /camera/image_raw")
```



Vérification

 Ouvrir un nouveau terminal pour vérifier :

```
ros2 topic list # Lister les topics
ros2 topic echo /vision_detection/ball # Voir les données
```

 Résultat attendu :

```
{"x": 320, "y": 240, "conf": 0.95}
```

5. Comprendre les coordonnées dans l'image

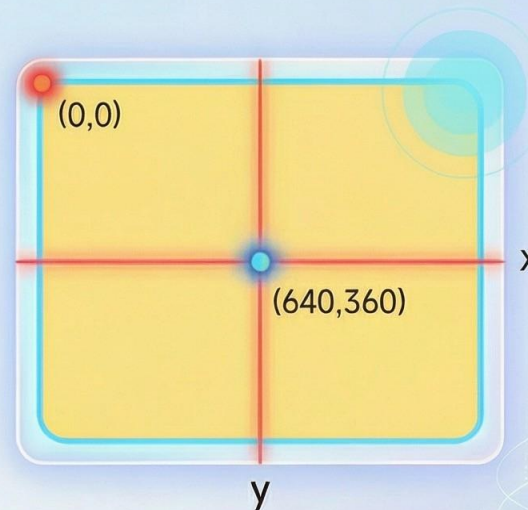
Les coordonnées dans l'image vue par le robot

Paramètre	Signification	Unité	Plage type
x	Position horizontale (gauche → droite)	Pixel	0-1280
y	Position verticale (haut → bas)	Pixel	0-720

- L'image de la caméra est une image en deux dimensions.
- En haut à gauche : (0, 0) ; en bas à droite : (1280, 720).
- Après détection du ballon, le programme calcule sa position à partir des résultats OpenCV, dans le repère des pixels de l'image.
- Ce sont les coordonnées d'image, ou coordonnées de détection.

- À partir des coordonnées en pixels, comment le robot sait-il dans quelle direction tourner ?
- Faut-il connaître le centre de l'image pour savoir si le ballon est à gauche ou à droite ?

Schéma du repère de l'image



 La caméra fournit des coordonnées d'image, mais les moteurs ont besoin d'angles : la transformation des coordonnées résout ce problème.

6. Principe de transformation des coordonnées



Pourquoi transformer les coordonnées ?

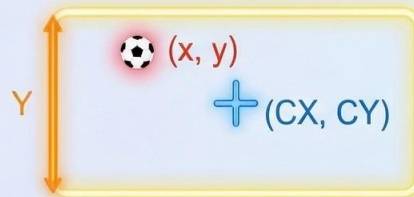
Vision	Commande
Coordonnées détectées : x, y (pixels)	Commande du robot : yaw, pitch (angle/vitesse)
Unité : pixels (1280 × 720)	Unité : radians ou vitesse angulaire (rad/s)
Position du ballon dans l'image	Direction dans laquelle orienter la tête

Les coordonnées en pixels de la caméra et celles de la tête sont différentes. La caméra indique si le ballon est à gauche ou à droite : c'est une information visuelle. Le moteur attend un angle ou une vitesse angulaire. Il faut donc convertir l'écart en pixels en un mouvement de rotation.

Calculer l'écart en pixels

$dx_pix = x - CX$ # $CX = 640$

$dy_pix = y - CY$ # $CY = 360$



Écart dans l'image	Condition	Mouvement de la tête
Ballon à gauche ($x < CX$)	$dx_pix < 0$	À gauche ($v_yaw > 0$)
Ballon à droite ($x > CX$)	$dx_pix > 0$	À droite ($v_yaw < 0$)
Ballon au-dessus ($y < CY$)	$dy_pix < 0$	Vers le haut ($v_pitch < 0$)
Ballon en dessous ($y > CY$)	$dy_pix > 0$	Vers le bas ($v_pitch > 0$)



7. Transformer les coordonnées : de la détection à la commande de la tête

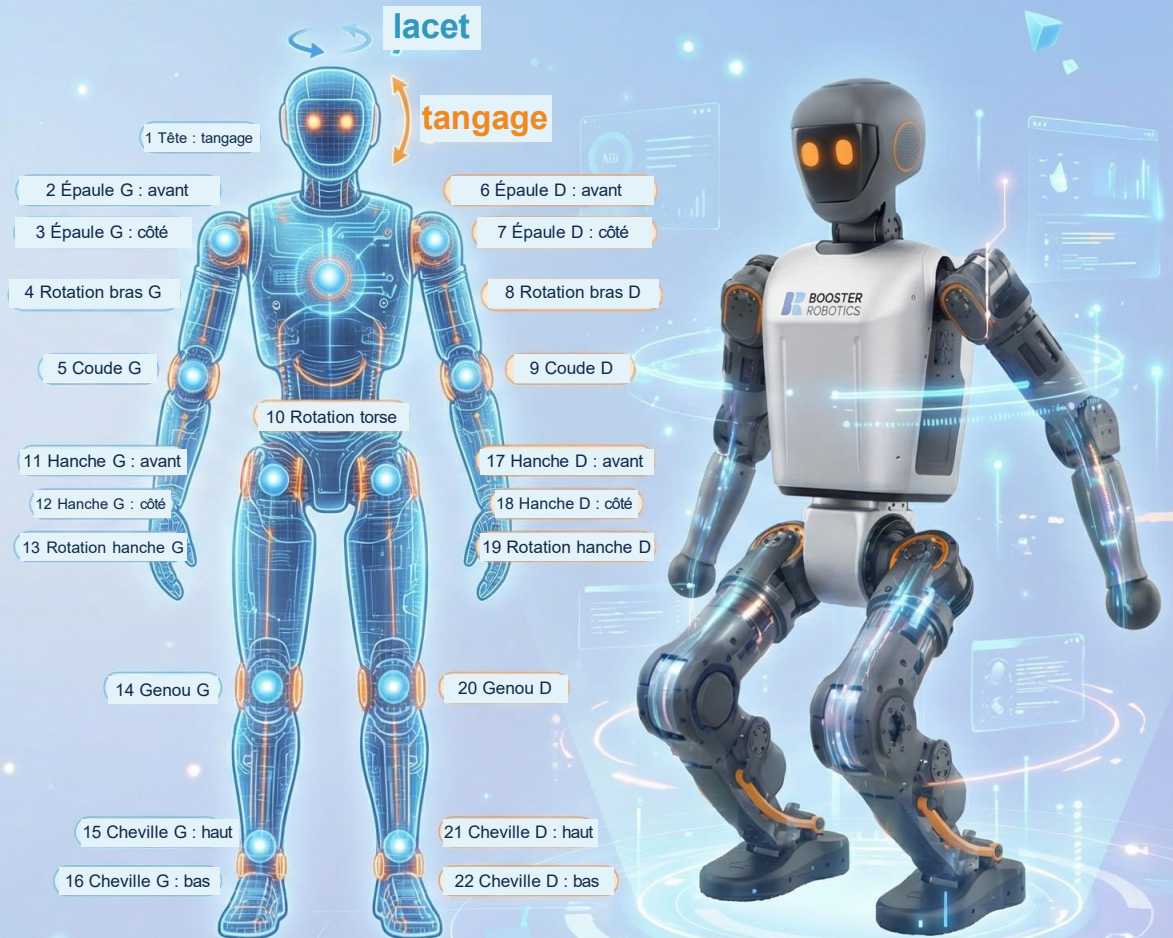
Coordonnées détectées et commandes

Élément	Coordonnées détectées (en pixels)	Commandes
Nom	x, y	yaw, pitch
Unité	Pixels	Vitesse angulaire (rad/s)
Source	Image de la caméra	Interface des moteurs
Plage	0-1280 / 0-720	$\pm 0,785$ (yaw) ; $-0,3$ à 1 (pitch)
Sens	Où se trouve le ballon	Comment la tête bouge



API de commande de la tête

Nom de l'interface	RotateHead
Description	Envoie la commande de la tête. Vitesse maximale de rotation : 20 rad/s.
Paramètre pitch	Angle vertical, positif vers le bas. Unité : rad ; plage : $-0,3$ à 1 rad.
Paramètre yaw	Angle horizontal, positif vers la gauche. Unité : rad ; plage : $-0,785$ à $0,785$ rad.





8. S'abonner au topic et piloter la rotation de la tête

Code de référence : head_follow_ball.py

```
#1 Décoder le message JSON
data = json.loads(msg.data)
x, y, conf = data["x"], data["y"], data["conf"]

#2 Ignorer les cibles peu fiables
if conf < 0.4:
    node.get_logger().info("Low confidence, ignoring this detection.")
    return

#3 Calculer l'écart en pixels
dx_pix = x - CX
dy_pix = y - CY

#4 Normaliser l'écart dans [-1, 1]
dx_norm = dx_pix / CX
dy_norm = dy_pix / CY

#5 Lisser : valeur =  $\alpha$ ·actuelle + (1- $\alpha$ )·précédente
dx = alpha * dx_norm + (1 - alpha) * prev_dx
dy = alpha * dy_norm + (1 - alpha) * prev_dy
prev_dx, prev_dy = dx, dy

#6 Convertir en vitesse angulaire
vyaw, vpitch = pixel_to_angle(dx, dy)

#7 Envoyer la commande
send_head_control(vyaw, vpitch)
```

Calcul de l'écart
Comparer la position du ballon
au centre de l'image
pour guider les corrections.

Position du ballon
Déterminer ses coordonnées
dans l'image pour
les calculs suivants.

Améliorer l'écart mesuré
Réduire le bruit pour obtenir
une commande plus précise.

Vitesse angulaire
Adapter la vitesse du robot
pour suivre le ballon
en douceur.

Mouvement de la tête
Garder le ballon dans
le champ de vision.

```
def pixel_to_angle(dx, dy):
```

Convertir l'écart normalisé (dx, dy) en vitesse de rotation.

```
.....
dx < 0 → ballon à gauche → tête à gauche (yaw > 0)
dx > 0 → ballon à droite → tête à droite (yaw < 0)
dy < 0 → ballon au-dessus → vers le haut (pitch < 0)
dy > 0 → ballon en dessous → vers le bas (pitch > 0)
```

```
vyaw = -Kx * dx
vpitch = Ky * dy
```

Limiter pour éviter une rotation excessive

```
vyaw = max(min(vyaw, MAX_YAW), -MAX_YAW)
vpitch = max(min(vpitch, MAX_PITCH), -MAX_PITCH)
```

```
return vyaw, vpitch
```

9. Améliorer la détection du ballon rouge



Aider le robot à « voir » la cible avec plus de précision

1 S'abonner au topic de la caméra :
`/camera/camera/color/image_raw`



2 Détecter la cible ronde et rouge (le ballon) ; mieux gérer l'éclairage et le bruit.



3 Lisser les coordonnées sur plusieurs images pour réduire les tremblements.



4 Publier les coordonnées lissées sur :
`/vision_detection/ball`



5 Les utiliser ensuite pour le suivi stable et la gestion de la perte de cible.



Code de référence : `ball_detection_optimized.py`

```
# =====
```

```
# 5. Fonction de lissage (sortie stable)
```

```
def smooth_value(x, y, conf):  
    """
```

Moyenner les coordonnées des images récentes pour obtenir une sortie fluide et stable.
Principe : conserver l'historique dans une file et calculer une moyenne pondérée à chaque étape.
 """

```
    history_x.append(x)  
    history_y.append(y)  
    history_conf.append(conf)
```

```
    avg_x = int(sum(history_x) / len(history_x))  
    avg_y = int(sum(history_y) / len(history_y))  
    avg_conf = round(sum(history_conf) / len(history_conf), 2)
```

```
    return avg_x, avg_y, avg_conf
```



10. Bilan du cours et prochaine séance

Ce que nous avons appris

- Comprendre le rôle des topics ROS 2, de la détection visuelle à l'exécution des actions.
- Faire suivre le ballon par la tête du robot grâce à une commande en boucle fermée.
- Trouver l'équilibre entre les paramètres de l'algorithme et la réponse du mouvement.

À la prochaine séance

- Garder le robot stable pendant le suivi.
- Rechercher automatiquement le ballon lorsqu'il est perdu.
- Obtenir un comportement intelligent de suivi réellement stable.



**À bientôt
pour le prochain cours !**



BOOSTER

Pourquoi l'intelligence incarnée ?

Le monde physique est conçu pour les humains. Pour travailler avec nous, les robots doivent comprendre les mouvements, l'espace et les interactions, au-delà du code.

Notre approche pédagogique

Notre conviction pour enseigner l'IA et la robotique :
partir de l'intuition, passer à l'ingénierie, puis agir dans le monde réel.

Nous contacter

- Tél. : +33 (0)4 90 60 05 68
- E-mail : contact@erm-automatismes.com
- Site web : www.erm-automatismes.com