ROS 2 — BOOTCAMP

Jour 4 — Vision

Voir, pour agir · OpenCV · ArUco · YOLO · CNN

Etienne Schmitz

Au programme

1. Ce qu'est une **image**, le **pipeline de perception** et la détection par **couleur** (HSV)
2. **OpenCV** : la boîte à outils de la vision classique
3. Les **marqueurs fiduciaires** (ArUco) et la **pose 6D**
4. La vision par **apprentissage profond** : YOLO & CNN
5. L'intégration en **ROS 2** : du flux caméra au `Detection.msg`

PARTIE 01 · VOIR, DE L'IMAGE À LA DÉCISION

Pourquoi la vision ?

La caméra est le capteur le plus riche — et le plus difficile à interpréter.

Que fait un robot de ses images ?

Une image brute n’est qu’un **tableau de nombres**. Tout l’enjeu : la transformer en **action**. Trois questions, dans l’ordre :

Que fait un robot de ses images ?

Une image brute n'est qu'un **tableau de nombres**. Tout l'enjeu : la transformer en **action**. Trois questions, dans l'ordre :

🔍 **Détecter**

Repérer un objet, une personne, un obstacle.
« Qu'y a-t-il ? »

Que fait un robot de ses images ?

Une image brute n'est qu'un **tableau de nombres**. Tout l'enjeu : la transformer en **action**. Trois questions, dans l'ordre :

🔍 Détecter

Repérer un objet, une personne, un obstacle.
« Qu'y a-t-il ? »

📍 Localiser

Estimer **où** l'objet se trouve dans l'espace.
« Où est-il ? »

Que fait un robot de ses images ?

Une image brute n'est qu'un **tableau de nombres**. Tout l'enjeu : la transformer en **action**. Trois questions, dans l'ordre :

🔍 Détecter

Repérer un objet, une personne, un obstacle.
« Qu'y a-t-il ? »

📍 Localiser

Estimer **où** l'objet se trouve dans l'espace.
« Où est-il ? »

🧭 Décider

Choisir l'action : saisir, éviter, suivre.
« Que faire ? »

Que fait un robot de ses images ?

Une image brute n'est qu'un **tableau de nombres**. Tout l'enjeu : la transformer en **action**. Trois questions, dans l'ordre :

🔍 Détecter

Repérer un objet, une personne, un obstacle.
« Qu'y a-t-il ? »

📍 Localiser

Estimer **où** l'objet se trouve dans l'espace.
« Où est-il ? »

🧭 Décider

Choisir l'action : saisir, éviter, suivre.
« Que faire ? »

⚠️ **Pourquoi c'est difficile**

Éclairage qui change, ombres, reflets, objets **partiellement cachés**, bruit du capteur — et tout ça en **temps réel**.

Le pipeline de perception

Le pipeline de perception

Chaque atelier parcourt ce **même pipeline** — seule la brique **Détection** change :

Le pipeline de perception

Chaque atelier parcourt ce **même pipeline** — seule la brique **Détection** change :

🟦 Formes (OpenCV)

Filtres & contours, détection par couleur.

Le pipeline de perception

Chaque atelier parcourt ce **même pipeline** — seule la brique **Détection** change :

🟦 Formes (OpenCV)

Filtres & contours, détection par couleur.

🎯 ArUco

Marqueurs fiduciaires → pose 6D.

Le pipeline de perception

Chaque atelier parcourt ce **même pipeline** — seule la brique **Détection** change :

🟦 Formes (OpenCV)

Filtres & contours, détection par couleur.

🎯 ArUco

Marqueurs fiduciaires → pose 6D.

🧠 YOLO

Détection par apprentissage profond.

Le pipeline de perception

Chaque atelier parcourt ce **même pipeline** — seule la brique **Détection** change :

🟦 Formes (OpenCV)

Filtres & contours, détection par couleur.

🎯 ArUco

Marqueurs fiduciaires → pose 6D.

🧠 YOLO

Détection par apprentissage profond.

🔢 Chiffres (CNN)

Classification d'images (MNIST).

ROS 2 — Bootcamp — Jour 4 Vision

ROS 2 — Bootcamp — Jour 4 Vision

ROS 2 — Bootcamp — Jour 4 Vision

ROS 2 — Bootcamp — Jour 4 Vision

ROS 2 — Bootcamp — Jour 4 Vision

ROS 2 — Bootcamp — Jour 4 Vision

En mémoire, cette grille est un **tableau NumPy** de forme `(H, W, C)`.

```
import cv2 as cv
img = cv.imread("scene.png")
print(img.shape)     # (720, 1280, 3)
print(img[360, 640]) # [B, G, R] du pixel central
```

> ⚠️ **Piège classique**
>
> OpenCV charge les canaux en **BGR**, pas RGB.

ROS 2 — Bootcamp — Jour 4 Vision

En mémoire, cette grille est un **tableau NumPy** de forme `(H, W, C)`.

- on lit, découpe et modifie l'image comme un tableau

```
import cv2 as cv
img = cv.imread("scene.png")
print(img.shape)     # (720, 1280, 3)
print(img[360, 640]) # [B, G, R] du pixel central
```

⚠️ **Piège classique**

OpenCV charge les canaux en **BGR**, pas RGB.

ROS 2 — Bootcamp — Jour 4 Vision

En mémoire, cette grille est un **tableau NumPy** de forme `(H, W, C)`.

- on lit, découpe et modifie l'image comme un tableau
- accès à un pixel : `img[y, x]` → `[B, G, R]`

```python
import cv2 as cv
img = cv.imread("scene.png")
print(img.shape)      # (720, 1280, 3)
print(img[360, 640]) # [B, G, R] du pixel central
```

> ⚠️ **Piège classique**
>
> OpenCV charge les canaux en **BGR**, pas RGB.

ROS 2 — Bootcamp — Jour 4 Vision

En mémoire, cette grille est un **tableau NumPy** de forme `(H, W, C)`.

- on lit, découpe et modifie l'image comme un tableau
- accès à un pixel : `img[y, x]` → `[B, G, R]`
- découper une zone : `img[y1:y2, x1:x2]`

```python
import cv2 as cv
img = cv.imread("scene.png")
print(img.shape)     # (720, 1280, 3)
print(img[360, 640]) # [B, G, R] du pixel central
```

⚠️ **Piège classique**

OpenCV charge les canaux en **BGR**, pas RGB.

ROS 2 — Bootcamp — Jour 4 Vision

En mémoire, cette grille est un **tableau NumPy** de forme `(H, W, C)`.

- on lit, découpe et modifie l'image comme un tableau
- accès à un pixel : `img[y, x]` → `[B, G, R]`
- découper une zone : `img[y1:y2, x1:x2]`

```python
import cv2 as cv
img = cv.imread("scene.png")
print(img.shape)     # (720, 1280, 3)
print(img[360, 640]) # [B, G, R] du pixel central
```

⚠️ **Piège classique**

OpenCV charge les canaux en **BGR**, pas RGB.

ROS 2 — Bootcamp — Jour 4 Vision

En mémoire, cette grille est un **tableau NumPy** de forme `(H, W, C)`.

- on lit, découpe et modifie l'image comme un tableau
- accès à un pixel : `img[y, x]` → `[B, G, R]`
- découper une zone : `img[y1:y2, x1:x2]`

```python
import cv2 as cv
img = cv.imread("scene.png")
print(img.shape)     # (720, 1280, 3)
print(img[360, 640]) # [B, G, R] du pixel central
```

⚠️ **Piège classique**

OpenCV charge les canaux en **BGR**, pas RGB.

Espaces colorimétriques

Espaces colorimétriques

BGR / RGB

Les trois canaux varient **ensemble** avec la lumière → fragile pour la couleur.

Espaces colorimétriques

BGR / RGB

Les trois canaux varient **ensemble** avec la lumière → fragile pour la couleur.

Niveaux de gris

Une seule intensité — suffisant pour **formes** et **contours**.

Espaces colorimétriques

BGR / RGB

Les trois canaux varient **ensemble** avec la lumière → fragile pour la couleur.

Niveaux de gris

Une seule intensité — suffisant pour **formes** et **contours**.

HSV

Teinte / **S**aturation / **V**aleur — la couleur est **isolée** de la luminosité.

Espaces colorimétriques

BGR / RGB

Les trois canaux varient **ensemble** avec la lumière → fragile pour la couleur.

Niveaux de gris

Une seule intensité — suffisant pour **formes** et **contours**.

HSV

Teinte / **S**aturation / **V**aleur — la couleur est **isolée** de la luminosité.

Quand utiliser quoi ?

Espaces colorimétriques

BGR / RGB

Les trois canaux varient **ensemble** avec la lumière → fragile pour la couleur.

Niveaux de gris

Une seule intensité — suffisant pour **formes** et **contours**.

HSV

Teinte / **S**aturation / **V**aleur — la couleur est **isolée** de la luminosité.

Quand utiliser quoi ?

1. **Gris** → formes et contours

Espaces colorimétriques

BGR / RGB

Les trois canaux varient **ensemble** avec la lumière → fragile pour la couleur.

Niveaux de gris

Une seule intensité — suffisant pour **formes** et **contours**.

HSV

Teinte / **S**aturation / **V**aleur — la couleur est **isolée** de la luminosité.

Quand utiliser quoi ?

1. **Gris** → formes et contours
2. **BGR / RGB** → affichage

Espaces colorimétriques

BGR / RGB

Les trois canaux varient **ensemble** avec la lumière → fragile pour la couleur.

Niveaux de gris

Une seule intensité — suffisant pour **formes** et **contours**.

HSV

Teinte / **S**aturation / **V**aleur — la couleur est **isolée** de la luminosité.

Quand utiliser quoi ?

1. **Gris** → formes et contours
2. **BGR / RGB** → affichage
3. **HSV** → détecter une couleur

ROS 2 — Bootcamp — Jour 4 Vision

On bascule en **HSV**, puis on garde les pixels dont la **teinte** est dans une plage.

```
hsv = cv.cvtColor(img, cv.COLOR_BGR2HSV)
lo = (35, 80, 80)    # vert : H ≈ 35-85
hi = (85, 255, 255)
masque = cv.inRange(hsv, lo, hi)
```

🎯 **Astuce**

Le rouge est « à cheval » sur H = 0 → souvent **deux** plages à fusionner.

ROS 2 — Bootcamp — Jour 4 Vision

On bascule en **HSV**, puis on garde les pixels dont la **teinte** est dans une plage.

- la **teinte** (H) reste stable même si l'éclairage change

```
hsv = cv.cvtColor(img, cv.COLOR_BGR2HSV)
lo = (35, 80, 80)    # vert : H ≈ 35-85
hi = (85, 255, 255)
masque = cv.inRange(hsv, lo, hi)
```

🎯 **Astuce**

Le rouge est « à cheval » sur H = 0 → souvent **deux** plages à fusionner.

ROS 2 — Bootcamp — Jour 4 Vision

On bascule en **HSV**, puis on garde les pixels dont la **teinte** est dans une plage.

- la **teinte** (H) reste stable même si l'éclairage change
- `inRange` renvoie un **masque** binaire (objet = blanc)

```
hsv = cv.cvtColor(img, cv.COLOR_BGR2HSV)
lo = (35, 80, 80)    # vert : H ≈ 35-85
hi = (85, 255, 255)
masque = cv.inRange(hsv, lo, hi)
```

🎯 **Astuce**

Le rouge est « à cheval » sur H = 0 → souvent **deux** plages à fusionner.

ROS 2 — Bootcamp — Jour 4 Vision

On bascule en **HSV**, puis on garde les pixels dont la **teinte** est dans une plage.

- la **teinte** (H) reste stable même si l'éclairage change
- `inRange` renvoie un **masque** binaire (objet = blanc)
- bien plus **robuste** que seuiller en BGR

```
hsv = cv.cvtColor(img, cv.COLOR_BGR2HSV)
lo = (35, 80, 80)    # vert : H ≈ 35-85
hi = (85, 255, 255)
masque = cv.inRange(hsv, lo, hi)
```

🎯 **Astuce**

Le rouge est « à cheval » sur H = 0 → souvent **deux** plages à fusionner.

ROS 2 — Bootcamp — Jour 4 Vision

On bascule en **HSV**, puis on garde les pixels dont la **teinte** est dans une plage.

- la **teinte** (H) reste stable même si l'éclairage change
- `inRange` renvoie un **masque** binaire (objet = blanc)
- bien plus **robuste** que seuiller en BGR

```
hsv = cv.cvtColor(img, cv.COLOR_BGR2HSV)
lo = (35, 80, 80)    # vert : H ≈ 35-85
hi = (85, 255, 255)
masque = cv.inRange(hsv, lo, hi)
```

🎯 **Astuce**

Le rouge est « à cheval » sur H = 0 → souvent **deux** plages à fusionner.

ROS 2 — Bootcamp — Jour 4 Vision

On bascule en **HSV**, puis on garde les pixels dont la **teinte** est dans une plage.

- la **teinte** (H) reste stable même si l'éclairage change
- `inRange` renvoie un **masque** binaire (objet = blanc)
- bien plus **robuste** que seuiller en BGR

```
hsv = cv.cvtColor(img, cv.COLOR_BGR2HSV)
lo = (35, 80, 80)    # vert : H ≈ 35-85
hi = (85, 255, 255)
masque = cv.inRange(hsv, lo, hi)
```

🎯 **Astuce**

Le rouge est « à cheval » sur H = 0 → souvent **deux** plages à fusionner.

ROS 2 — Bootcamp — Jour 4 Vision

On bascule en **HSV**, puis on garde les pixels dont la **teinte** est dans une plage.

- la **teinte** (H) reste stable même si l'éclairage change
- `inRange` renvoie un **masque** binaire (objet = blanc)
- bien plus **robuste** que seuiller en BGR

```
hsv = cv.cvtColor(img, cv.COLOR_BGR2HSV)
lo = (35, 80, 80)    # vert : H ≈ 35-85
hi = (85, 255, 255)
masque = cv.inRange(hsv, lo, hi)
```

🎯 **Astuce**

Le rouge est « à cheval » sur H = 0 → souvent **deux** plages à fusionner.

PARTIE 02 · OPENCV, LA BOÎTE À OUTILS

La vision classique, à la main

Atelier « Formes ».

OpenCV, c’est quoi ?

Une **bibliothèque open-source** de vision par ordinateur (C++ / **Python**), pensée pour le **temps réel**. Le couteau suisse de l’image.

OpenCV, c’est quoi ?

Une **bibliothèque open-source** de vision par ordinateur (C++ / **Python**), pensée pour le **temps réel**. Le couteau suisse de l’image.

📥 E/S

Lire / écrire images & vidéos, accès caméra, affichage.

OpenCV, c’est quoi ?

Une **bibliothèque open-source** de vision par ordinateur (C++ / **Python**), pensée pour le **temps réel**. Le couteau suisse de l’image.

📥 E/S

Lire / écrire images & vidéos, accès caméra, affichage.

🎛️ Filtres

Flou, seuillage, morphologie, détection de bords (Canny).

OpenCV, c’est quoi ?

Une **bibliothèque open-source** de vision par ordinateur (C++ / **Python**), pensée pour le **temps réel**. Le couteau suisse de l’image.

📥 E/S

Lire / écrire images & vidéos, accès caméra, affichage.

🎛️ Filtres

Flou, seuillage, morphologie, détection de bords (Canny).

📐 Géométrie

Redimensionner, tourner, recadrer, transformations de perspective.

OpenCV, c’est quoi ?

Une **bibliothèque open-source** de vision par ordinateur (C++ / **Python**), pensée pour le **temps réel**. Le couteau suisse de l’image.

📥 E/S

Lire / écrire images & vidéos, accès caméra, affichage.

🎛️ Filtres

Flou, seuillage, morphologie, détection de bords (Canny).

📐 Géométrie

Redimensionner, tourner, recadrer, transformations de perspective.

🧩 Formes

Contours, features, mise en correspondance de motifs.

OpenCV, c’est quoi ?

Une **bibliothèque open-source** de vision par ordinateur (C++ / **Python**), pensée pour le **temps réel**. Le couteau suisse de l’image.

📥 E/S

Lire / écrire images & vidéos, accès caméra, affichage.

🎛️ Filtres

Flou, seuillage, morphologie, détection de bords (Canny).

📐 Géométrie

Redimensionner, tourner, recadrer, transformations de perspective.

🧩 Formes

Contours, features, mise en correspondance de motifs.

🎯 Caméra

Calibration, intrinsèques, marqueurs **ArUco**, `solvePnP`.

OpenCV, c’est quoi ?

Une **bibliothèque open-source** de vision par ordinateur (C++ / **Python**), pensée pour le **temps réel**. Le couteau suisse de l’image.

📥 E/S

Lire / écrire images & vidéos, accès caméra, affichage.

🎛️ Filtres

Flou, seuillage, morphologie, détection de bords (Canny).

📐 Géométrie

Redimensionner, tourner, recadrer, transformations de perspective.

🧩 Formes

Contours, features, mise en correspondance de motifs.

🎯 Caméra

Calibration, intrinsèques, marqueurs **ArUco**, `solvePnP`.

🧠 DNN

Module pour faire tourner des réseaux pré-entraînés.

ROS 2 — Bootcamp — Jour 4 Vision

De l'image brute à une **forme géométrique**, étape par étape.

```
gris = cv.cvtColor(img, cv.COLOR_BGR2GRAY)
flou = cv.GaussianBlur(gris, (5,5), 0)
_, b = cv.threshold(flou, 0, 255,
        cv.THRESH_BINARY+cv.THRESH_OTSU)
cnts,_ = cv.findContours(b,
        cv.RETR_EXTERNAL, cv.CHAIN_APPROX_SIMPLE)
```

ROS 2 — Bootcamp — Jour 4 Vision

De l'image brute à une **forme géométrique**, étape par étape.

- **gris** : on abandonne la couleur

```
gris = cv.cvtColor(img, cv.COLOR_BGR2GRAY)
flou = cv.GaussianBlur(gris, (5,5), 0)
_, b = cv.threshold(flou, 0, 255,
        cv.THRESH_BINARY+cv.THRESH_OTSU)
cnts,_ = cv.findContours(b,
        cv.RETR_EXTERNAL, cv.CHAIN_APPROX_SIMPLE)
```

ROS 2 — Bootcamp — Jour 4 Vision

De l'image brute à une **forme géométrique**, étape par étape.

- **gris** : on abandonne la couleur
- **flou** gaussien : réduire le bruit

```
gris = cv.cvtColor(img, cv.COLOR_BGR2GRAY)
flou = cv.GaussianBlur(gris, (5,5), 0)
_, b = cv.threshold(flou, 0, 255,
        cv.THRESH_BINARY+cv.THRESH_OTSU)
cnts,_ = cv.findContours(b,
        cv.RETR_EXTERNAL, cv.CHAIN_APPROX_SIMPLE)
```

ROS 2 — Bootcamp — Jour 4 Vision

De l'image brute à une **forme géométrique**, étape par étape.

- **gris** : on abandonne la couleur
- **flou** gaussien : réduire le bruit
- **seuillage** / **Canny** : isoler les bords

```
gris = cv.cvtColor(img, cv.COLOR_BGR2GRAY)
flou = cv.GaussianBlur(gris, (5,5), 0)
_, b = cv.threshold(flou, 0, 255,
       cv.THRESH_BINARY+cv.THRESH_OTSU)
cnts,_ = cv.findContours(b,
       cv.RETR_EXTERNAL, cv.CHAIN_APPROX_SIMPLE)
```

ROS 2 — Bootcamp — Jour 4 Vision

De l'image brute à une **forme géométrique**, étape par étape.

- **gris** : on abandonne la couleur
- **flou** gaussien : réduire le bruit
- **seuillage** / **Canny** : isoler les bords
- **contours** : `findContours`

```
gris = cv.cvtColor(img, cv.COLOR_BGR2GRAY)
flou = cv.GaussianBlur(gris, (5,5), 0)
_, b = cv.threshold(flou, 0, 255,
       cv.THRESH_BINARY+cv.THRESH_OTSU)
cnts,_ = cv.findContours(b,
       cv.RETR_EXTERNAL, cv.CHAIN_APPROX_SIMPLE)
```

ROS 2 — Bootcamp — Jour 4 Vision

De l'image brute à une **forme géométrique**, étape par étape.

- **gris** : on abandonne la couleur
- **flou** gaussien : réduire le bruit
- **seuillage** / **Canny** : isoler les bords
- **contours** : `findContours`
- **forme** : `approxPolyDP` → nb de sommets

```
gris = cv.cvtColor(img, cv.COLOR_BGR2GRAY)
flou = cv.GaussianBlur(gris, (5,5), 0)
_, b = cv.threshold(flou, 0, 255,
       cv.THRESH_BINARY+cv.THRESH_OTSU)
cnts,_ = cv.findContours(b,
       cv.RETR_EXTERNAL, cv.CHAIN_APPROX_SIMPLE)
```

ROS 2 — Bootcamp — Jour 4 Vision

De l'image brute à une **forme géométrique**, étape par étape.

- **gris** : on abandonne la couleur
- **flou** gaussien : réduire le bruit
- **seuillage** / **Canny** : isoler les bords
- **contours** : `findContours`
- **forme** : `approxPolyDP` → nb de sommets

```
gris = cv.cvtColor(img, cv.COLOR_BGR2GRAY)
flou = cv.GaussianBlur(gris, (5,5), 0)
_, b = cv.threshold(flou, 0, 255,
       cv.THRESH_BINARY+cv.THRESH_OTSU)
cnts,_ = cv.findContours(b,
       cv.RETR_EXTERNAL, cv.CHAIN_APPROX_SIMPLE)
```

ROS 2 — Bootcamp — Jour 4 Vision

De l'image brute à une **forme géométrique**, étape par étape.

- **gris** : on abandonne la couleur
- **flou** gaussien : réduire le bruit
- **seuillage** / **Canny** : isoler les bords
- **contours** : `findContours`
- **forme** : `approxPolyDP` → nb de sommets

```
gris = cv.cvtColor(img, cv.COLOR_BGR2GRAY)
flou = cv.GaussianBlur(gris, (5,5), 0)
_, b = cv.threshold(flou, 0, 255,
        cv.THRESH_BINARY+cv.THRESH_OTSU)
cnts,_ = cv.findContours(b,
        cv.RETR_EXTERNAL, cv.CHAIN_APPROX_SIMPLE)
```

ROS 2 — Bootcamp — Jour 4 Vision

De l'image brute à une **forme géométrique**, étape par étape.

- **gris** : on abandonne la couleur
- **flou** gaussien : réduire le bruit
- **seuillage** / **Canny** : isoler les bords
- **contours** : `findContours`
- **forme** : `approxPolyDP` → nb de sommets

```
gris = cv.cvtColor(img, cv.COLOR_BGR2GRAY)
flou = cv.GaussianBlur(gris, (5,5), 0)
_, b = cv.threshold(flou, 0, 255,
       cv.THRESH_BINARY+cv.THRESH_OTSU)
cnts,_ = cv.findContours(b,
         cv.RETR_EXTERNAL, cv.CHAIN_APPROX_SIMPLE)
```

ROS 2 — Bootcamp — Jour 4 Vision

De l'image brute à une **forme géométrique**, étape par étape.

- **gris** : on abandonne la couleur
- **flou** gaussien : réduire le bruit
- **seuillage** / **Canny** : isoler les bords
- **contours** : `findContours`
- **forme** : `approxPolyDP` → nb de sommets

```
gris = cv.cvtColor(img, cv.COLOR_BGR2GRAY)
flou = cv.GaussianBlur(gris, (5,5), 0)
_, b = cv.threshold(flou, 0, 255,
        cv.THRESH_BINARY+cv.THRESH_OTSU)
cnts,_ = cv.findContours(b,
        cv.RETR_EXTERNAL, cv.CHAIN_APPROX_SIMPLE)
```

OpenCV : forces et limites

OpenCV : forces et limites

✅ Atouts

Rapide et **temps réel**, déterministe, **aucun entraînement**, explicable — on sait pourquoi ça marche (ou non).

OpenCV : forces et limites

✅ Atouts

Rapide et **temps réel**, déterministe, **aucun entraînement**, explicable — on sait pourquoi ça marche (ou non).

⚠️ Limites

Sensible à l'**éclairage**, réglages **manuels** par scène, pose **2D** seulement, fragile dès que l'objet varie.

OpenCV : forces et limites

✅ Atouts

Rapide et **temps réel**, déterministe, **aucun entraînement**, explicable — on sait pourquoi ça marche (ou non).

⚠️ Limites

Sensible à l'**éclairage**, réglages **manuels** par scène, pose **2D** seulement, fragile dès que l'objet varie.

💡 **Bon réflexe**

Commencer **simple** avec OpenCV. Si l'objet est trop variable, on ajoute un **repère** (ArUco) ou on passe à l'**apprentissage profond**.

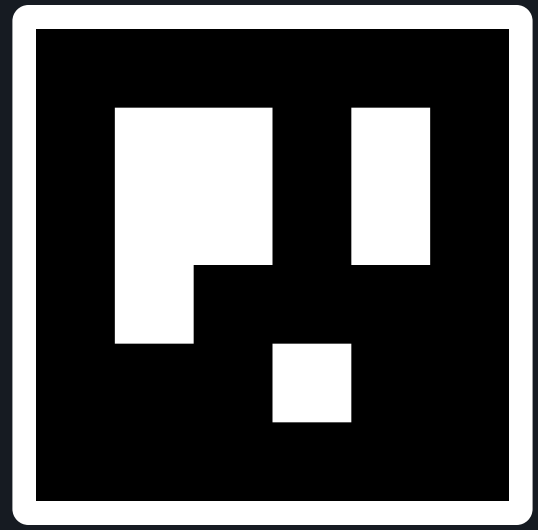
PARTIE 03 · MARQUEURS FIDUCIAIRES

Des repères conçus pour être détectés

Atelier « ArUco ».

ROS 2 — Bootcamp — Jour 4 Vision

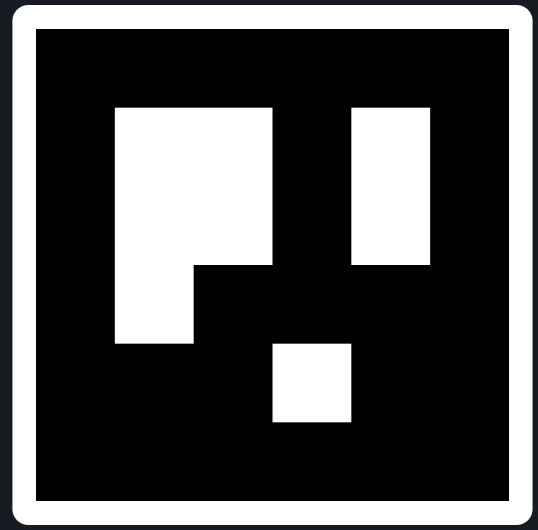
Un motif carré noir et blanc qui encode un **identifiant**.



ROS 2 — Bootcamp — Jour 4 Vision

Un motif carré noir et blanc qui encode un **identifiant**.

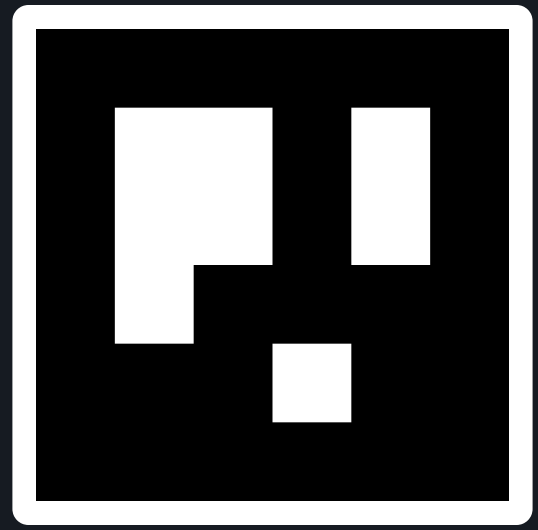
- la **classe** = l'ID du marqueur (gratuit)



ROS 2 — Bootcamp — Jour 4 Vision

Un motif carré noir et blanc qui encode un **identifiant**.

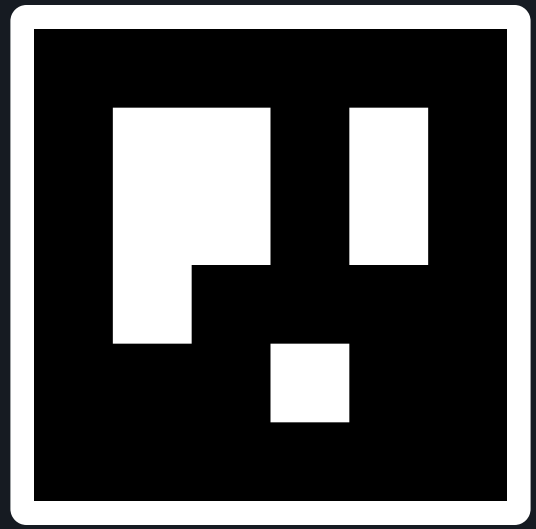
- la **classe** = l'ID du marqueur (gratuit)
- les 4 **coins** se détectent de façon fiable



ROS 2 — Bootcamp — Jour 4 Vision

Un motif carré noir et blanc qui encode un **identifiant**.

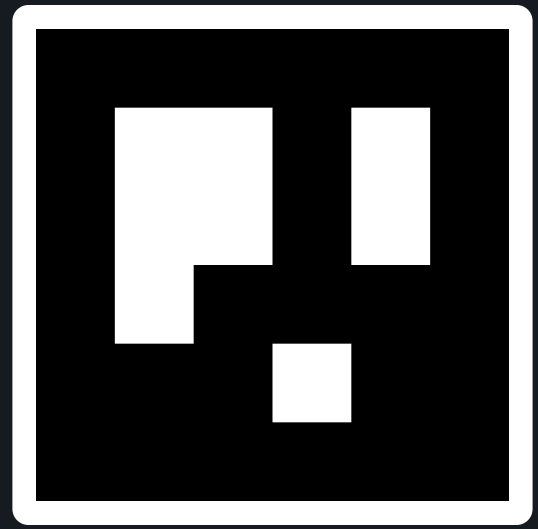
- la **classe** = l'ID du marqueur (gratuit)
- les 4 **coins** se détectent de façon fiable
- appartiennent à un **dictionnaire** (ex. 4×4_50)



ROS 2 — Bootcamp — Jour 4 Vision

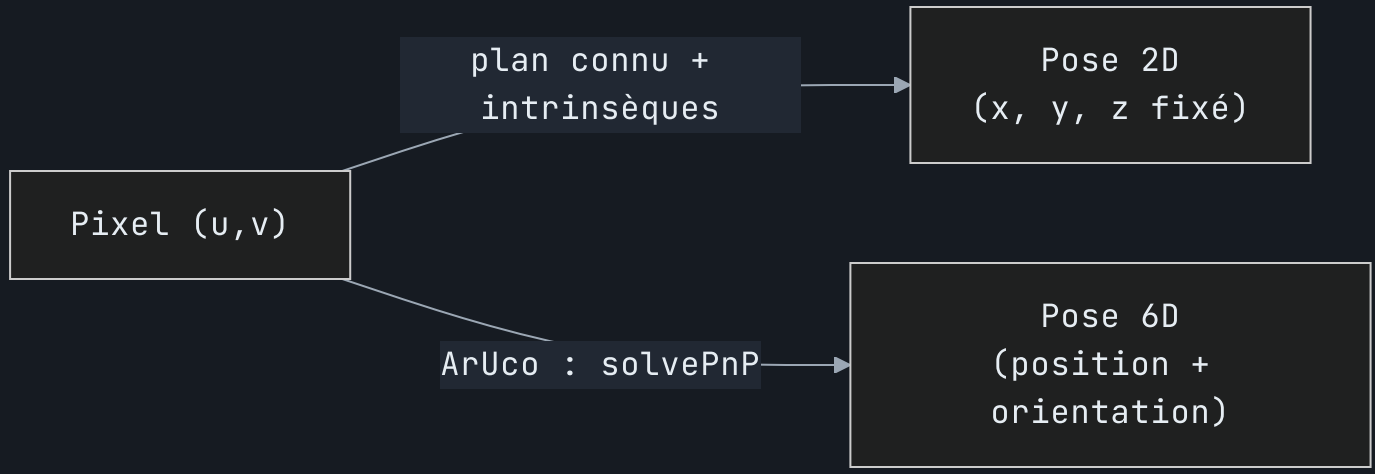
Un motif carré noir et blanc qui encode un **identifiant**.

- la **classe** = l'ID du marqueur (gratuit)
- les 4 **coins** se détectent de façon fiable
- appartiennent à un **dictionnaire** (ex. 4×4_50)

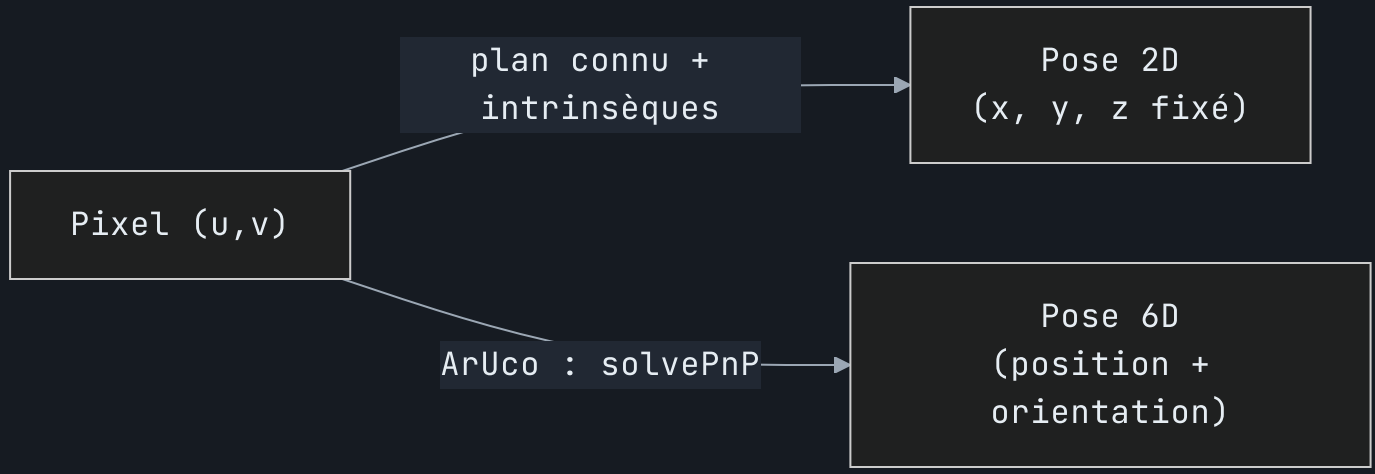


> 🎯 **Pose 6D gratuite**
>
> Avec la **taille** du marqueur et les **intrinsèques** caméra, `solvePnP` donne position *et* orientation.

Pose 2D vs pose 6D



Pose 2D vs pose 6D



ArUco est votre détecteur **« filet de sécurité »** : robuste, déterministe, 6D.

PARTIE 04 · VISION PAR APPRENTISSAGE

Apprendre au lieu de programmer

Ateliers « YOLO » et « Chiffres ».

Trois tâches de vision profonde

Trois tâches de vision profonde

🏷️ Classification

Une **classe** pour toute l'image.
« cette vignette = un 7 »
→ CNN · **atelier Chiffres**

Trois tâches de vision profonde

🏷️ Classification

Une **classe** pour toute l'image.
« cette vignette = un 7 »
→ CNN · **atelier Chiffres**

🎁 Détection

Plusieurs **boîtes** + classes + scores.
« un colis ici, une bouteille là »
→ YOLO · **atelier YOLO**

Trois tâches de vision profonde

🏷️ Classification

Une **classe** pour toute l'image.
« cette vignette = un 7 »
→ CNN · **atelier Chiffres**

🎁 Détection

Plusieurs **boîtes** + classes + scores.
« un colis ici, une bouteille là »
→ YOLO · **atelier YOLO**

🧩 Segmentation

Une classe par **pixel**.
« ces pixels = le colis »
→ hors-programme

Trois tâches de vision profonde

🏷️ Classification

Une **classe** pour toute l'image.
« cette vignette = un 7 »
→ CNN · **atelier Chiffres**

🎁 Détection

Plusieurs **boîtes** + classes + scores.
« un colis ici, une bouteille là »
→ YOLO · **atelier YOLO**

🧩 Segmentation

Une classe par **pixel**.
« ces pixels = le colis »
→ hors-programme

🎯 **Pour le robot**

Pour **saisir** un objet, il faut une **boîte** (où ?) — donc la **détection**. La classification seule ne localise pas. Vos deux ateliers IA couvrent ces deux tâches ; la segmentation, on la cite pour la culture.

Le CNN, c’est quoi ?

Un **réseau de neurones convolutif** (CNN) s’inspire du **cortex visuel** : au lieu de programmer des filtres à la main, il **apprend tout seul** les motifs utiles (traits, courbes, formes).

- 1998 **LeNet** (Yann LeCun) — lecture des chiffres postaux : le tout premier CNN utile
- 2012 **AlexNet** — victoire à **ImageNet**, l'essor du deep learning moderne
- auj. brique de base de toute la vision profonde — **YOLO** compris

Le CNN, c’est quoi ?

Un **réseau de neurones convolutif** (CNN) s’inspire du **cortex visuel** : au lieu de programmer des filtres à la main, il **apprend tout seul** les motifs utiles (traits, courbes, formes).

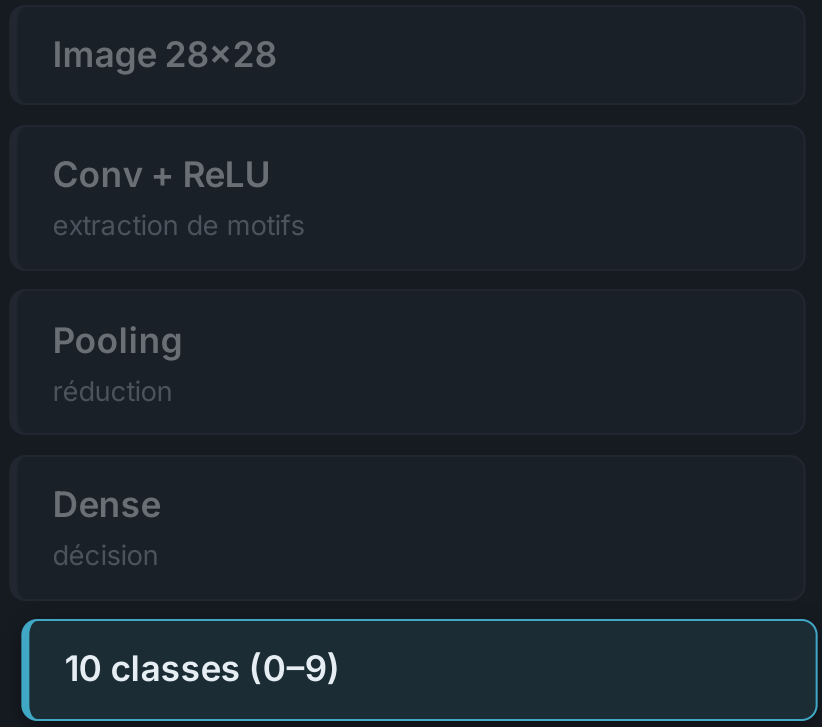
- **1998** **LeNet** (Yann LeCun) — lecture des chiffres postaux : le tout premier CNN utile
- **2012** **AlexNet** — victoire à **ImageNet**, l'essor du deep learning moderne
- **auj.** brique de base de toute la vision profonde — **YOLO** compris

🔢 **L’atelier « Chiffres » = un LeNet moderne**

Vous referez, en 2026, l'expérience fondatrice de 1998 : un CNN qui lit des chiffres.

ROS 2 — Bootcamp — Jour 4 Vision

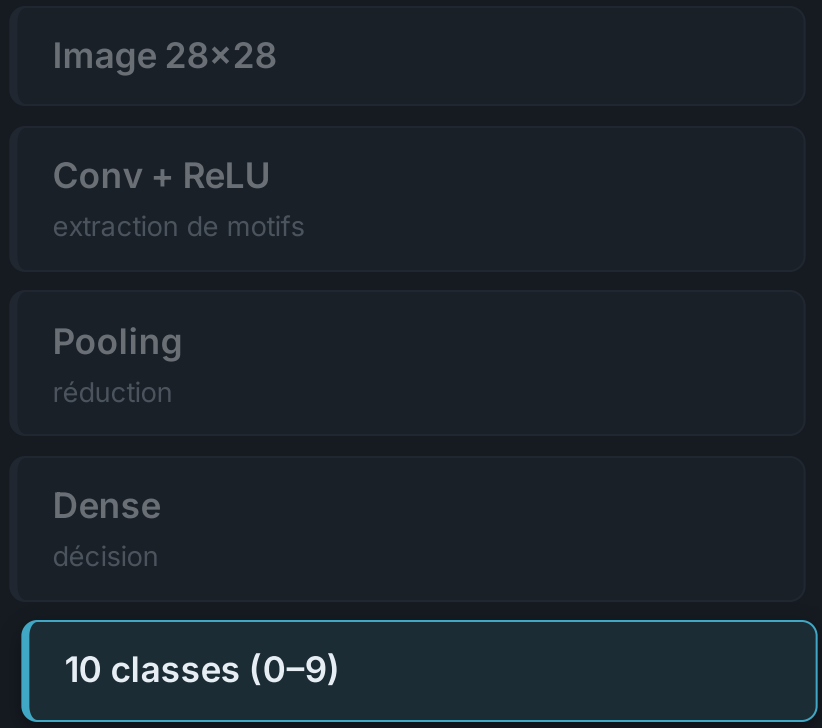
Un **CNN** apprend tout seul les motifs utiles (traits, courbes, textures) par **convolutions** successives.



ROS 2 — Bootcamp — Jour 4 Vision

Un **CNN** apprend tout seul les motifs utiles (traits, courbes, textures) par **convolutions** successives.

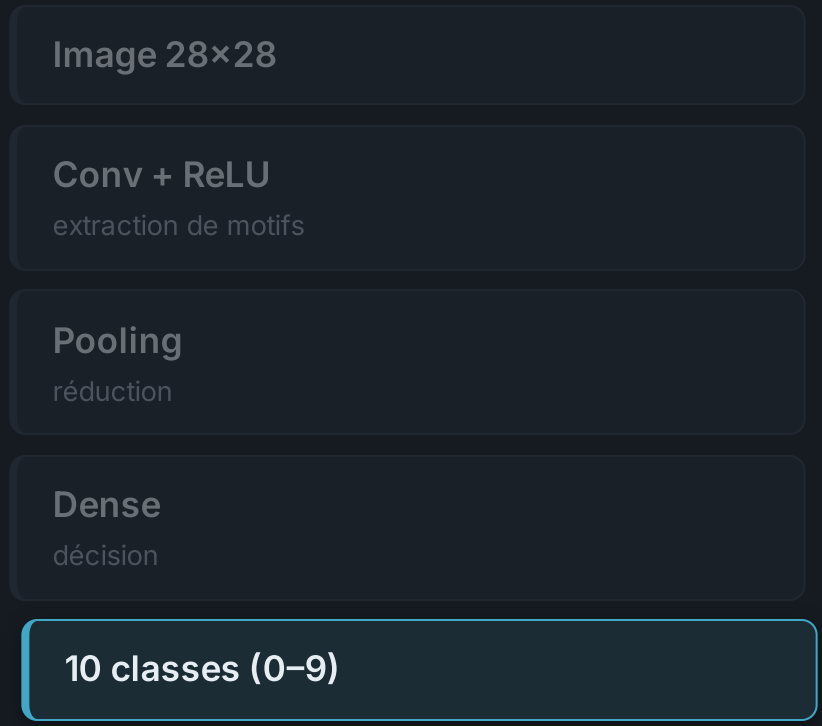
- couches **conv** → extraction de motifs



ROS 2 — Bootcamp — Jour 4 Vision

Un **CNN** apprend tout seul les motifs utiles (traits, courbes, textures) par **convolutions** successives.

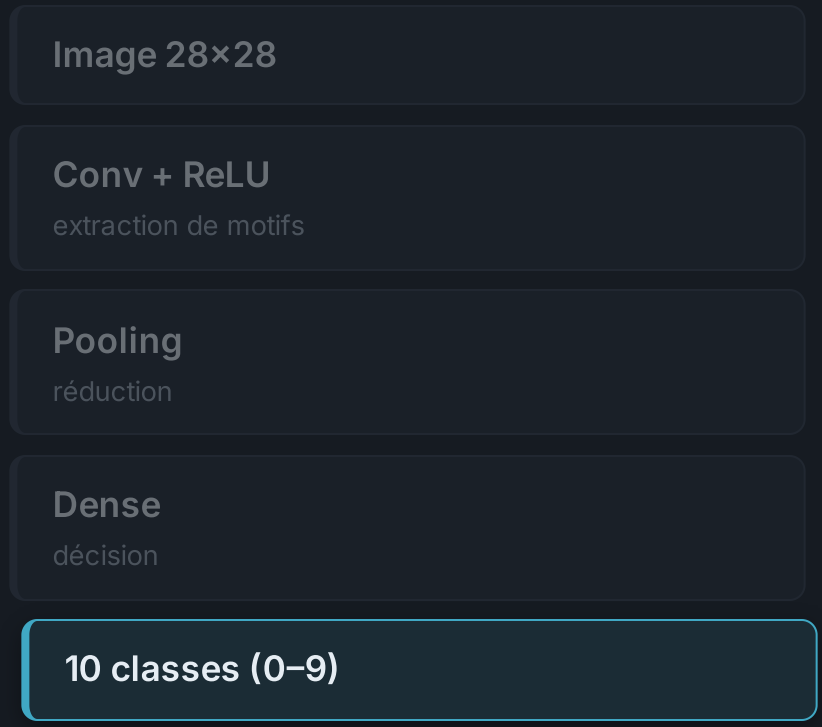
- couches **conv** → extraction de motifs
- **pooling** → réduction



ROS 2 — Bootcamp — Jour 4 Vision

Un **CNN** apprend tout seul les motifs utiles (traits, courbes, textures) par **convolutions** successives.

- couches **conv** → extraction de motifs
- **pooling** → réduction
- couches **denses** → décision

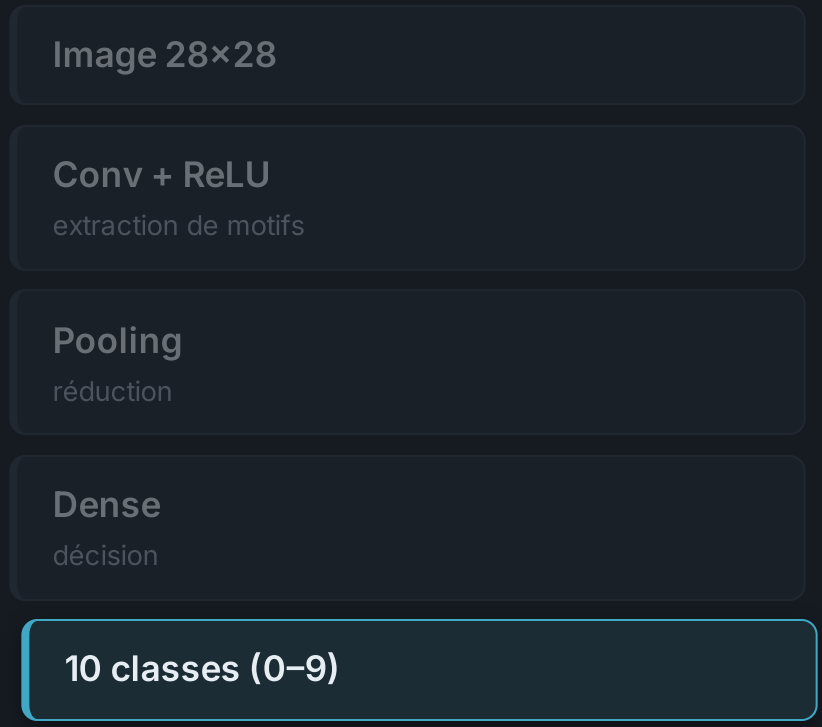


ROS 2 — Bootcamp — Jour 4 Vision

Un **CNN** apprend tout seul les motifs utiles (traits, courbes, textures) par **convolutions** successives.

- couches **conv** → extraction de motifs
- **pooling** → réduction
- couches **denses** → décision

> 🔢 **C'est l'atelier « Chiffres »**
>
> Vous entraînerez ce réseau pour lire les **chiffres** de l'entrepôt. Et la même brique de convolution est le **cœur de YOLO**.



ROS 2 — Bootcamp — Jour 4 Vision

L'entraînement est une **boucle** : on corrige les poids un peu à chaque passage sur les données.

1. **Données** annotées (images + étiquettes)
2. **Prédiction**, puis **perte** (erreur)
3. **Rétropropagation** → màj des poids
4. On répète, **epoch** après epoch

↻ **Pourquoi plusieurs epochs ?**

Un seul passage ne suffit pas : on **repasse** sur les données (étapes 2→3) jusqu'à ce que la perte se **stabilise**.

🏷️ **L'annotation gratuite**

En simulation, la pose des objets est **connue** → on génère **et** annote tout seul. C'est le **dataset de l'atelier YOLO**.

YOLO, c’est quoi ?

YOLO = *You Only Look Once*. Un détecteur qui regarde l’image **une seule fois** et sort **toutes** les boîtes d’un coup — d’où sa vitesse, idéale pour le **temps réel**.

- avant **R-CNN** — précis mais **lent** : plusieurs passes par image
- 2015 **YOLO** (Joseph Redmon) — détection **temps réel** en une seule passe
- → v11 versions successives, aujourd'hui maintenu par **Ultralytics**

YOLO, c’est quoi ?

YOLO = *You Only Look Once*. Un détecteur qui regarde l’image **une seule fois** et sort **toutes** les boîtes d’un coup — d’où sa vitesse, idéale pour le **temps réel**.

- **avant** **R-CNN** — précis mais **lent** : plusieurs passes par image
- **2015** **YOLO** (Joseph Redmon) — détection **temps réel** en une seule passe
- **→ v11** versions successives, aujourd'hui maintenu par **Ultralytics**

⚡ **Pourquoi ça compte pour un robot**

« Une seule passe » = assez rapide pour tourner sur le **flux caméra** en direct.

ROS 2 — Bootcamp — Jour 4 Vision

```python
from ultralytics import YOLO
model = YOLO("yolo11n.pt")
for b in model("scene.png")[0].boxes:
    print(model.names[int(b.cls)], float(b.conf))
```

ROS 2 — Bootcamp — Jour 4 Vision

- localise **et** classe plusieurs objets d’un coup

```
from ultralytics import YOLO
model = YOLO("yolo11n.pt")
for b in model("scene.png")[0].boxes:
    print(model.names[int(b.cls)], float(b.conf))
```

ROS 2 — Bootcamp — Jour 4 Vision

- localise **et** classe plusieurs objets d'un coup
- modèle **pré-entraîné** (COCO, 80 classes)

```
from ultralytics import YOLO
model = YOLO("yolo11n.pt")
for b in model("scene.png")[0].boxes:
    print(model.names[int(b.cls)], float(b.conf))
```

ROS 2 — Bootcamp — Jour 4 Vision

- localise **et** classe plusieurs objets d'un coup
- modèle **pré-entraîné** (COCO, 80 classes)
- **transfer learning** : ré-entraîner sur vos objets

```python
from ultralytics import YOLO
model = YOLO("yolo11n.pt")
for b in model("scene.png")[0].boxes:
    print(model.names[int(b.cls)], float(b.conf))
```

ROS 2 — Bootcamp — Jour 4 Vision

- localise **et** classe plusieurs objets d'un coup
- modèle **pré-entraîné** (COCO, 80 classes)
- **transfer learning** : ré-entraîner sur vos objets

```python
from ultralytics import YOLO
model = YOLO("yolo11n.pt")
for b in model("scene.png")[0].boxes:
    print(model.names[int(b.cls)], float(b.conf))
```

ROS 2 — Bootcamp — Jour 4 Vision

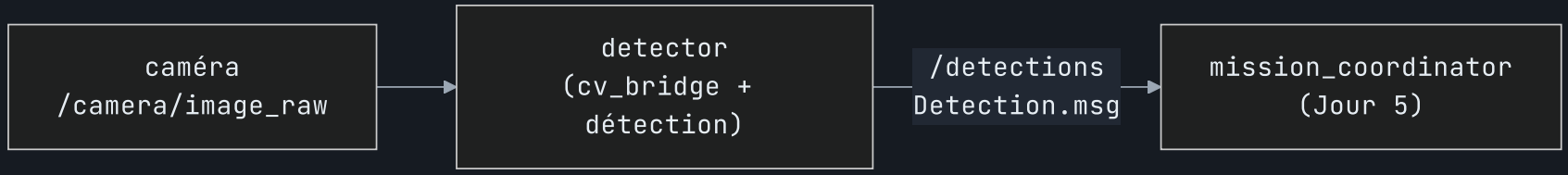
- localise **et** classe plusieurs objets d'un coup
- modèle **pré-entraîné** (COCO, 80 classes)
- **transfer learning** : ré-entraîner sur vos objets

```python
from ultralytics import YOLO
model = YOLO("yolo11n.pt")
for b in model("scene.png")[0].boxes:
    print(model.names[int(b.cls)], float(b.conf))
```

PARTIE 05 · VISION DANS ROS 2

Brancher la perception au robot

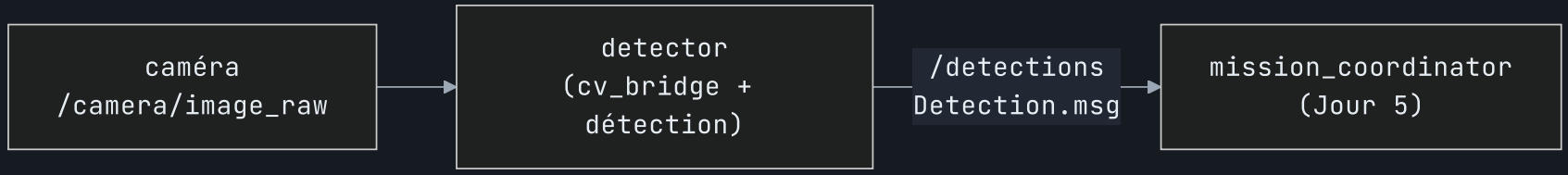
Du flux caméra à `/detections`



`cv_bridge` convertit l'`Image` ROS en tableau OpenCV ; on détecte, puis on **publie**.

```
frame = CvBridge().imgmsg_to_cv2(msg, "bgr8")   # sensor_msgs/Image → tableau BGR
```

Du flux caméra à `/detections`



`cv_bridge` convertit l'`Image` ROS en tableau OpenCV ; on détecte, puis on **publie**.

```
frame = CvBridge().imgmsg_to_cv2(msg, "bgr8")   # sensor_msgs/Image → tableau BGR
```

> `DetectionArray` = liste de `Detection` (classe + score + boîte 2D + `pose`). **Quelle que soit la méthode**, la sortie est la même → le Jour 5 branche votre perception sans rien changer.

OpenCV vs IA — quel outil quand ?

OpenCV vs IA — quel outil quand ?

🟦 OpenCV / ArUco

- **Données** : aucune
- **Mise en place** : immédiate, déterministe
- **Explicabilité** : totale (on sait pourquoi)
- **Limite** : éclairage, objets variables

→ *objet simple ou marqué*

OpenCV vs IA — quel outil quand ?

🟦 OpenCV / ArUco

- **Données** : aucune
- **Mise en place** : immédiate, déterministe
- **Explicabilité** : totale (on sait pourquoi)
- **Limite** : éclairage, objets variables

→ *objet simple ou marqué*

🧠 Apprentissage profond

- **Données** : dataset annoté requis
- **Mise en place** : entraînement + calcul
- **Explicabilité** : faible (boîte noire)
- **Atout** : robuste à la variété

→ *objets nombreux ou variables*

OpenCV vs IA — quel outil quand ?

🟦 OpenCV / ArUco

- **Données** : aucune
- **Mise en place** : immédiate, déterministe
- **Explicabilité** : totale (on sait pourquoi)
- **Limite** : éclairage, objets variables

→ *objet simple ou marqué*

🧠 Apprentissage profond

- **Données** : dataset annoté requis
- **Mise en place** : entraînement + calcul
- **Explicabilité** : faible (boîte noire)
- **Atout** : robuste à la variété

→ *objets nombreux ou variables*

💡 **Bon réflexe**

Commencer **simple** (ArUco, valeur sûre), monter en gamme **si nécessaire**.

À vous de jouer — lequel choisir ?

À vous de jouer — lequel choisir ?

1. 🟦 **Formes (OpenCV)** — si vous voulez la **vision classique** sans IA : filtres, contours, couleur.

À vous de jouer — lequel choisir ?

1. 🟦 **Formes (OpenCV)** — si vous voulez la **vision classique** sans IA : filtres, contours, couleur.

2. 🎯 **ArUco (pose 6D)** — si vous voulez la **valeur sûre** : robuste, déterministe, pose complète.

À vous de jouer — lequel choisir ?

1. 🟦 **Formes (OpenCV)** — si vous voulez la **vision classique** sans IA : filtres, contours, couleur.
2. 🎯 **ArUco (pose 6D)** — si vous voulez la **valeur sûre** : robuste, déterministe, pose complète.
3. 🧠 **YOLO (détection)** — si vous voulez de l'**IA appliquée** : modèle pré-entraîné puis ré-entraîné sur vos objets.

À vous de jouer — lequel choisir ?

1. 🟦 **Formes (OpenCV)** — si vous voulez la **vision classique** sans IA : filtres, contours, couleur.
2. 🎯 **ArUco (pose 6D)** — si vous voulez la **valeur sûre** : robuste, déterministe, pose complète.
3. 🧠 **YOLO (détection)** — si vous voulez de l'**IA appliquée** : modèle pré-entraîné puis ré-entraîné sur vos objets.
4. 🔢 **Chiffres (CNN)** — si vous voulez **entraîner un réseau de A à Z** (MNIST).

Ce que vous repartez avec ce soir

Ce que vous repartez avec ce soir

✅ **Un nœud** `detector`

Qui voit la caméra et publie un `DetectionArray` sur `/detections`.

Ce que vous repartez avec ce soir

✅ Un nœud `detector`

Qui voit la caméra et publie un `DetectionArray` sur `/detections`.

🔗 Une brique prête pour le Jour 5

Le `mission_coordinator` consomme `/detections` — il **branche votre perception sans rien changer**.

Ce que vous repartez avec ce soir

✅ Un nœud `detector`

Qui voit la caméra et publie un `DetectionArray` sur `/detections`.

🔗 Une brique prête pour le Jour 5

Le `mission_coordinator` consomme `/detections` — il **branche votre perception sans rien changer**.

🚀 **Demain — Jour 5 : intégration**

Votre `detector` remplace le nœud factice du Jour 1 : voir → localiser → **agir** (pick & place).

ROS 2 — BOOTCAMP

Questions ?

ROS 2 — Bootcamp · Perpignan 2026

ros2.etienne-schmitz.com