ROS 2 — BOOTCAMP

Jour 3 — Manipulation

Anatomie · Cinématique · Modélisation · Planification

Etienne Schmitz

Au programme

1. L'**anatomie d'un bras** : joints, degrés de liberté, espaces, redondance, singularités
2. La **cinématique** : directe (FK), inverse (IK) et ses méthodes de résolution
3. La **modélisation dans ROS 2** : URDF, Xacro, SRDF et l'arbre de transformations TF
4. La **planification de mouvement** : OMPL et la boîte à outils MoveIt 2

PARTIE 01 · ANATOMIE D'UN BRAS ROBOTIQUE

De quoi est fait un bras ?

Segments, articulations, degrés de liberté — le vocabulaire de base.

ROS 2 — Bootcamp — Jour 3 Manipulation

Un bras manipulateur est une suite de **segments rigides** (`link`) reliés par des **articulations** (`joint`). La chaîne est **série** : chaque joint dépend de tous ceux qui le précèdent.

Le **SO-101** (notre fil rouge) — 6 joints :

Joint	Rôle
`shoulder_pan`	rotation base
`shoulder_lift`	levée épaule
`elbow_flex`	flexion coude
`wrist_flex`	flexion poignet
`wrist_roll`	rotation poignet
`gripper`	pince

ROS 2 — Bootcamp — Jour 3 Manipulation

Un bras manipulateur est une suite de **segments rigides** (`link`) reliés par des **articulations** (`joint`). La chaîne est **série** : chaque joint dépend de tous ceux qui le précèdent.

- chaque joint apporte **un degré de liberté** (DDL) ;

Le **SO-101** (notre fil rouge) — 6 joints :

Joint	Rôle
`shoulder_pan`	rotation base
`shoulder_lift`	levée épaule
`elbow_flex`	flexion coude
`wrist_flex`	flexion poignet
`wrist_roll`	rotation poignet
`gripper`	pince

ROS 2 — Bootcamp — Jour 3 Manipulation

Un bras manipulateur est une suite de **segments rigides** (`link`) reliés par des **articulations** (`joint`). La chaîne est **série** : chaque joint dépend de tous ceux qui le précèdent.

- chaque joint apporte **un degré de liberté** (DDL) ;
- au bout de la chaîne : l'**effecteur** (la pince, *end-effector*) ;

Le **SO-101** (notre fil rouge) — 6 joints :

Joint	Rôle
`shoulder_pan`	rotation base
`shoulder_lift`	levée épaule
`elbow_flex`	flexion coude
`wrist_flex`	flexion poignet
`wrist_roll`	rotation poignet
`gripper`	pince

ROS 2 — Bootcamp — Jour 3 Manipulation

Un bras manipulateur est une suite de **segments rigides** (`link`) reliés par des **articulations** (`joint`). La chaîne est **série** : chaque joint dépend de tous ceux qui le précèdent.

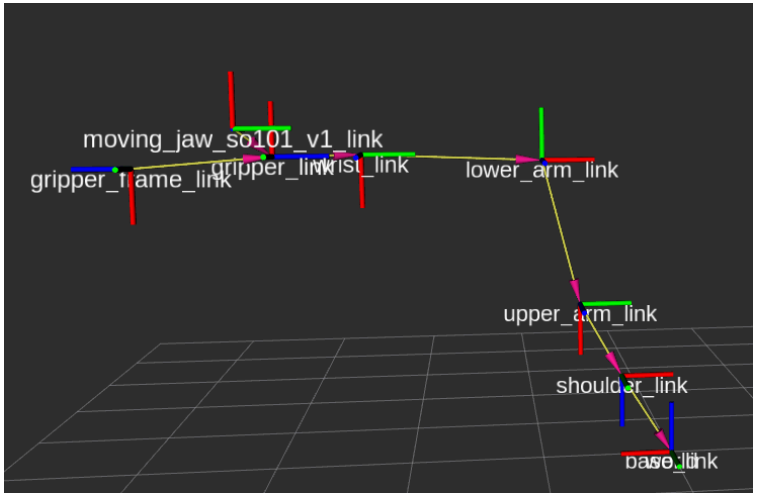
- chaque joint apporte **un degré de liberté** (DDL) ;
- au bout de la chaîne : l'**effecteur** (la pince, *end-effector*) ;
- bouger un joint déplace **tout** ce qui est en aval.

Le **SO-101** (notre fil rouge) — 6 joints :

Joint	Rôle
`shoulder_pan`	rotation base
`shoulder_lift`	levée épaule
`elbow_flex`	flexion coude
`wrist_flex`	flexion poignet
`wrist_roll`	rotation poignet
`gripper`	pince

ROS 2 — Bootcamp — Jour 3
Manipulation

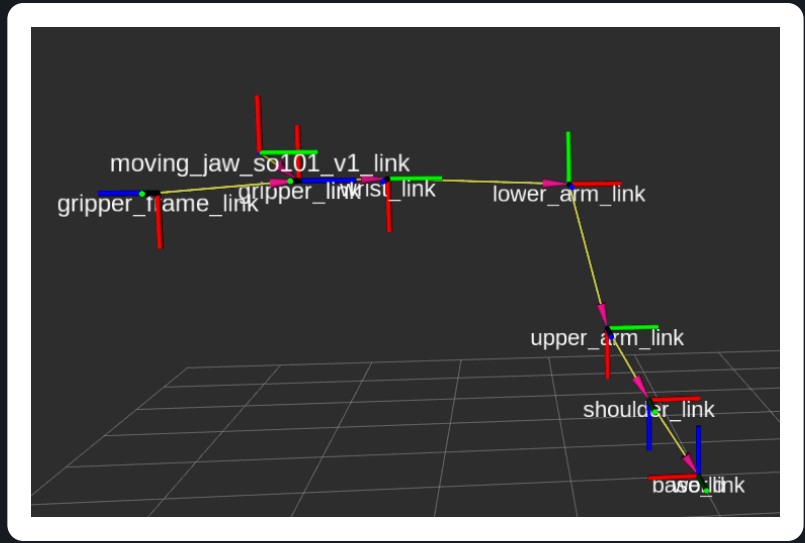
Un joint représente **un mouvement élémentaire** commandé. Deux grandes familles :



ROS 2 — Bootcamp — Jour 3
Manipulation

Un joint représente **un mouvement élémentaire** commandé. Deux grandes familles :

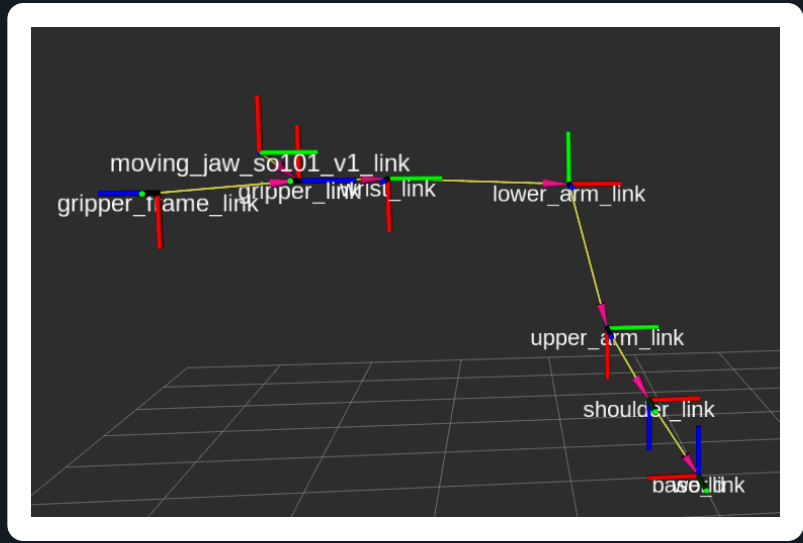
- **rotationnel** (*revolute*) — une rotation autour d'un axe (le cas le plus courant) ;



ROS 2 — Bootcamp — Jour 3
Manipulation

Un joint représente **un mouvement élémentaire** commandé. Deux grandes familles :

- **rotationnel** (*revolute*) — une rotation autour d'un axe (le cas le plus courant) ;
- **linéaire** (*prismatic*) — une translation le long d'un axe.

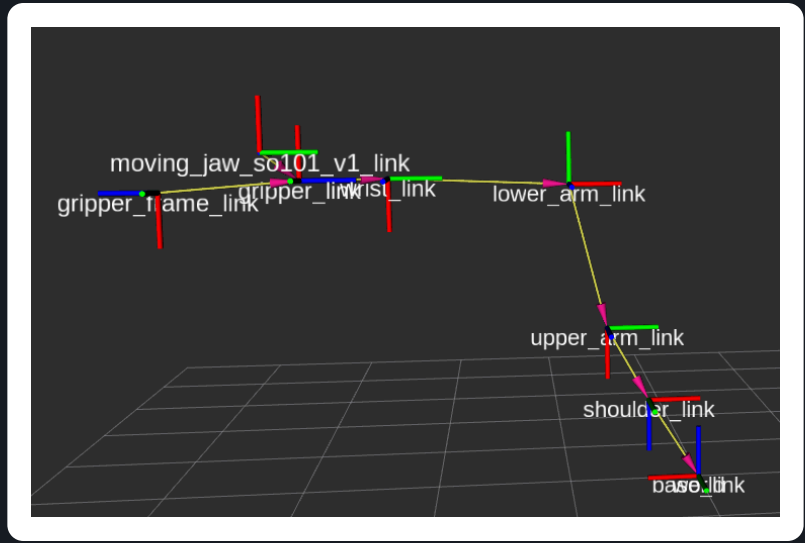


ROS 2 — Bootcamp — Jour 3
Manipulation

Un joint représente **un mouvement élémentaire** commandé. Deux grandes familles :

- **rotationnel** (*revolute*) — une rotation autour d'un axe (le cas le plus courant) ;
- **linéaire** (*prismatic*) — une translation le long d'un axe.

Chaque joint a des **limites** : butées de position, vitesse max, effort max.



Commander un joint

On pilote un joint de **trois façons**, selon ce qu’on impose au moteur :

Commander un joint

On pilote un joint de **trois façons**, selon ce qu'on impose au moteur :

📐 Position

Angle visé, ex. θ = π/2 rad

Commander un joint

On pilote un joint de **trois façons**, selon ce qu'on impose au moteur :

📐 Position

Angle visé, ex. θ = π/2 rad

⏩ Vitesse

Vitesse angulaire, ex. ω = π/4 rad/s

Commander un joint

On pilote un joint de **trois façons**, selon ce qu'on impose au moteur :

📐 Position

Angle visé, ex. θ = π/2 rad

⏩ Vitesse

Vitesse angulaire, ex. ω = π/4 rad/s

💪 Couple

Effort appliqué, ex. τ = 2 N·m

Commander un joint

On pilote un joint de **trois façons**, selon ce qu'on impose au moteur :

📐 Position

Angle visé, ex. θ = π/2 rad

⏩ Vitesse

Vitesse angulaire, ex. ω = π/4 rad/s

💪 Couple

Effort appliqué, ex. τ = 2 N·m

1 joint = 1 DDL

Compter les joints actifs, c'est compter les **degrés de liberté** du bras. Le SO-101 en a 6.

ROS 2 — Bootcamp — Jour 3 Manipulation

Deux façons, équivalentes, de décrire l’état du bras :

ROS 2 — Bootcamp — Jour 3 Manipulation

Deux façons, équivalentes, de décrire l'état du bras :

- **espace des joints** — la liste des angles internes $\boldsymbol{\theta} = (\theta_1, \dots, \theta_n)$;

ROS 2 — Bootcamp — Jour 3 Manipulation

Deux façons, équivalentes, de décrire l'état du bras :

- **espace des joints** — la liste des angles internes $\boldsymbol{\theta} = (\theta_1, \ldots, \theta_n)$;
- **espace cartésien** — la **pose** de l'effecteur : sa **position** (x, y, z) **et** son **orientation**.

ROS 2 — Bootcamp — Jour 3 Manipulation

Deux façons, équivalentes, de décrire l'état du bras :

- **espace des joints** — la liste des angles internes $\boldsymbol{\theta} = (\theta_1, \dots, \theta_n)$;
- **espace cartésien** — la **pose** de l'effecteur : sa **position** (x, y, z) **et** son **orientation**.

🔁 **Passer de l'un à l'autre**

L'humain raisonne en cartésien (« attrape la tasse là »), le moteur obéit en joints. Traduire entre les deux, c'est tout l'enjeu de la **cinématique** (FK / IK, partie suivante).

La redondance

En 3D, positionner **et** orienter un effecteur demande **6 DDL** : 3 pour la position, 3 pour l’orientation.

La redondance

En 3D, positionner **et** orienter un effecteur demande **6 DDL** : 3 pour la position, 3 pour l'orientation.

- un bras à **7 DDL ou plus** est dit **redondant** ;

La redondance

En 3D, positionner **et** orienter un effecteur demande **6 DDL** : 3 pour la position, 3 pour l'orientation.

- un bras à **7 DDL ou plus** est dit **redondant** ;
- il existe alors une **infinité** de configurations pour une même pose ;

La redondance

En 3D, positionner **et** orienter un effecteur demande **6 DDL** : 3 pour la position, 3 pour l'orientation.

- un bras à **7 DDL ou plus** est dit **redondant** ;
- il existe alors une **infinité** de configurations pour une même pose ;
- on exploite cette liberté pour : **éviter un obstacle**, **minimiser l'effort**, **rester loin des butées**.

La redondance

En 3D, positionner **et** orienter un effecteur demande **6 DDL** : 3 pour la position, 3 pour l'orientation.

- un bras à **7 DDL ou plus** est dit **redondant** ;
- il existe alors une **infinité** de configurations pour une même pose ;
- on exploite cette liberté pour : **éviter un obstacle**, **minimiser l'effort**, **rester loin des butées**.

Intuition

Comme votre bras : main posée sur la table, vous pouvez encore **bouger le coude**. La main ne bouge pas, la configuration si.

Aparté — les DDL du SO-101

Le SO-101 a **6 articulations**, mais toutes ne servent pas à *positionner* l'effecteur :

Aparté — les DDL du SO-101

Le SO-101 a **6 articulations**, mais toutes ne servent pas à *positionner* l'effecteur :

- **5 DDL** orientent et placent le poignet (`shoulder_pan`, `shoulder_lift`, `elbow_flex`, `wrist_flex`, `wrist_roll`) ;

Aparté — les DDL du SO-101

Le SO-101 a **6 articulations**, mais toutes ne servent pas à *positionner* l'effecteur :

- **5 DDL** orientent et placent le poignet (`shoulder_pan`, `shoulder_lift`, `elbow_flex`, `wrist_flex`, `wrist_roll`) ;
- **1 DDL** pour la **pince** (`gripper`) — il ouvre/ferme, il ne déplace pas l'effecteur.

Aparté — les DDL du SO-101

Le SO-101 a **6 articulations**, mais toutes ne servent pas à *positionner* l'effecteur :

- **5 DDL** orientent et placent le poignet (`shoulder_pan` , `shoulder_lift` , `elbow_flex` , `wrist_flex` , `wrist_roll`) ;
- **1 DDL** pour la **pince** (`gripper`) — il ouvre/ferme, il ne déplace pas l'effecteur.

5 DDL utiles < 6

Une pose 6D **arbitraire** (position + orientation libres) demande 6 DDL. Avec 5, le SO-101 est **sous-actionné** : il n'est **pas redondant**, et l'IK cartésienne 6D y est **dégradée**.

Aparté — les DDL du SO-101

Le SO-101 a **6 articulations**, mais toutes ne servent pas à *positionner* l'effecteur :

- **5 DDL** orientent et placent le poignet (`shoulder_pan`, `shoulder_lift`, `elbow_flex`, `wrist_flex`, `wrist_roll`) ;
- **1 DDL** pour la **pince** (`gripper`) — il ouvre/ferme, il ne déplace pas l'effecteur.

> 🧮 **5 DDL utiles < 6**
>
> Une pose 6D **arbitraire** (position + orientation libres) demande 6 DDL. Avec 5, le SO-101 est **sous-actionné** : il n'est **pas redondant**, et l'IK cartésienne 6D y est **dégradée**.

> En pratique on planifie souvent **dans l'espace des joints** plutôt que d'imposer une pose 6D complète.

Les singularités

Une **singularité** est une configuration où le bras **perd un degré de liberté de contrôle** de l'effecteur dans l'espace cartésien.

Les singularités

Une **singularité** est une configuration où le bras **perd un degré de liberté de contrôle** de l'effecteur dans l'espace cartésien.

📍 De position

Impossible de translater l'effecteur le long d'un certain axe.

Les singularités

Une **singularité** est une configuration où le bras **perd un degré de liberté de contrôle** de l'effecteur dans l'espace cartésien.

📍 De position

Impossible de translater l'effecteur le long d'un certain axe.

🧭 D'orientation

Impossible de tourner autour d'un certain axe (deux axes alignés).

Les singularités

Une **singularité** est une configuration où le bras **perd un degré de liberté de contrôle** de l'effecteur dans l'espace cartésien.

📍 De position

Impossible de translater l'effecteur le long d'un certain axe.

🧭 D'orientation

Impossible de tourner autour d'un certain axe (deux axes alignés).

♾️ Cinématique

Le Jacobien J devient **non inversible** → vitesses articulaires qui explosent.

Les singularités

Une **singularité** est une configuration où le bras **perd un degré de liberté de contrôle** de l'effecteur dans l'espace cartésien.

📍 De position

Impossible de translater l'effecteur le long d'un certain axe.

🧭 D'orientation

Impossible de tourner autour d'un certain axe (deux axes alignés).

♾️ Cinématique

Le Jacobien J devient **non inversible** → vitesses articulaires qui explosent.

⚠️ **Intuition : le bras tendu**

Bras complètement **étiré**, vous voulez avancer la main encore un peu : impossible sans plier le coude. Près de cette limite, un tout petit déplacement cartésien exige une vitesse articulaire **énorme** — le planificateur doit éviter ces zones.

PARTIE 02 · CINÉMATIQUE

Cinématique directe & inverse

FK : des angles vers une pose. IK : une pose vers des angles. Le calcul que MoveIt résout pour vous.

FK vs IK — vue d’ensemble

Deux problèmes **inverses** l’un de l’autre, qui relient les deux espaces vus en Partie 01 :

FK vs IK — vue d'ensemble

Deux problèmes **inverses** l'un de l'autre, qui relient les deux espaces vus en Partie 01 :

➡️ Cinématique directe (FK)

On **connaît les angles** θ → on calcule la **pose** de l'effecteur. Toujours **une seule** réponse.

FK vs IK — vue d'ensemble

Deux problèmes **inverses** l'un de l'autre, qui relient les deux espaces vus en Partie 01 :

➡️ Cinématique directe (FK)

On **connaît les angles** θ → on calcule la **pose** de l'effecteur. Toujours **une seule** réponse.

⬅️ Cinématique inverse (IK)

On **veut une pose** T^* → on cherche les **angles** θ. Zéro, une, plusieurs ou une infinité de réponses.

FK vs IK — vue d'ensemble

Deux problèmes **inverses** l'un de l'autre, qui relient les deux espaces vus en Partie 01 :

➡️ Cinématique directe (FK)

On **connaît les angles** θ → on calcule la **pose** de l'effecteur. Toujours **une seule** réponse.

⬅️ Cinématique inverse (IK)

On **veut une pose** T^* → on cherche les **angles** θ. Zéro, une, plusieurs ou une infinité de réponses.

> FK est facile et déterministe. **IK est le problème difficile** — et celui qu'on résout en permanence pour piloter un bras.

Cinématique directe (FK)

Chaque joint i porte une transformation homogène $T_{i-1}^{i}(\theta_i)$. La pose de la pince dans le repère base est le **produit** des transformations :

$$T_0^n(\boldsymbol{\theta}) = \prod_{i=1}^{n} T_{i-1}^{i}(\theta_i) = \begin{bmatrix} R(\boldsymbol{\theta}) & \mathbf{p}(\boldsymbol{\theta}) \\ \mathbf{0} & 1 \end{bmatrix}$$

Cinématique directe (FK)

Chaque joint i porte une transformation homogène $T_{i-1}^{i}(\theta_i)$. La pose de la pince dans le repère base est le **produit** des transformations :

$$T_0^n(\boldsymbol{\theta}) = \prod_{i=1}^{n} T_{i-1}^{i}(\theta_i) = \begin{bmatrix} R(\boldsymbol{\theta}) & \mathbf{p}(\boldsymbol{\theta}) \\ \mathbf{0} & 1 \end{bmatrix}$$

FK : on connaît $\boldsymbol{\theta}$ → on calcule la pose T_0^n. Direct, une seule solution.

Cinématique directe (FK)

Chaque joint i porte une transformation homogène $T_{i-1}^{i}(\theta_i)$. La pose de la pince dans le repère base est le **produit** des transformations :

$$T_0^n(\boldsymbol{\theta}) = \prod_{i=1}^{n} T_{i-1}^{i}(\theta_i) = \begin{bmatrix} R(\boldsymbol{\theta}) & \mathbf{p}(\boldsymbol{\theta}) \\ \mathbf{0} & 1 \end{bmatrix}$$

FK : on connaît $\boldsymbol{\theta}$ → on calcule la pose T_0^n. Direct, une seule solution.

> R est la matrice de **rotation** (orientation), $\mathbf{p}$ le vecteur **position** de l'effecteur.

Cinématique inverse (IK)

Problème inverse : on **veut** une pose T^*, on cherche les angles $\boldsymbol{\theta}$:

$$\text{trouver } \boldsymbol{\theta} \;\; \text{t.q. } \; T_0^n(\boldsymbol{\theta}) = T^*$$

Cinématique inverse (IK)

Problème inverse : on **veut** une pose T^*, on cherche les angles $\boldsymbol{\theta}$:

$$\text{trouver } \boldsymbol{\theta} \;\text{ t.q. }\; T_0^n(\boldsymbol{\theta}) = T^*$$

- **Plusieurs** solutions (coude haut / coude bas) ;

Cinématique inverse (IK)

Problème inverse : on **veut** une pose T^*, on cherche les angles $\boldsymbol{\theta}$:

$$\text{trouver } \boldsymbol{\theta} \;\; \text{t.q. } \; T_0^n(\boldsymbol{\theta}) = T^*$$

- **Plusieurs** solutions (coude haut / coude bas) ;
- **aucune** solution (cible hors d'atteinte ou contrainte géométrique) ;

Cinématique inverse (IK)

Problème inverse : on **veut** une pose T^*, on cherche les angles $\boldsymbol{\theta}$:

$$\text{trouver } \boldsymbol{\theta} \;\; \text{t.q. } \; T_0^n(\boldsymbol{\theta}) = T^*$$

- **Plusieurs** solutions (coude haut / coude bas) ;
- **aucune** solution (cible hors d'atteinte ou contrainte géométrique) ;
- une **infinité** si le bras est redondant → on parle de **null-space**.

Cinématique inverse (IK)

Problème inverse : on **veut** une pose T^*, on cherche les angles $\boldsymbol{\theta}$:

$$\text{trouver } \boldsymbol{\theta} \;\text{ t.q. }\; T_0^n(\boldsymbol{\theta}) = T^*$$

- **Plusieurs** solutions (coude haut / coude bas) ;
- **aucune** solution (cible hors d'atteinte ou contrainte géométrique) ;
- une **infinité** si le bras est redondant → on parle de **null-space**.

> SO-101 : bras **5 DOF** → on ne peut pas imposer une **orientation arbitraire**. Viser une pose 6D directe échoue souvent ; on **résout l'IK une fois** puis on planifie **dans l'espace des joints** (mesuré en TP : pose 6D 0/5 vs but articulaire 5/5).

ROS 2 — Bootcamp — Jour 3 Manipulation

Quand l'IK renvoie **plusieurs** solutions, laquelle prendre ? On les **classe** selon des critères :

ROS 2 — Bootcamp — Jour 3 Manipulation

Quand l'IK renvoie **plusieurs** solutions, laquelle prendre ? On les **classe** selon des critères :

- **continuité** — la plus proche de la pose actuelle (mouvement minimal) ;

ROS 2 — Bootcamp — Jour 3 Manipulation

Quand l'IK renvoie **plusieurs** solutions, laquelle prendre ? On les **classe** selon des critères :

- **continuité** — la plus proche de la pose actuelle (mouvement minimal) ;
- **limites articulaires** — rester dans les butées ;

ROS 2 — Bootcamp — Jour 3 Manipulation

Quand l'IK renvoie **plusieurs** solutions, laquelle prendre ? On les **classe** selon des critères :

- **continuité** — la plus proche de la pose actuelle (mouvement minimal) ;
- **limites articulaires** — rester dans les butées ;
- **collisions** — écarter les configurations qui heurtent le robot ou la scène ;

ROS 2 — Bootcamp — Jour 3 Manipulation

Quand l'IK renvoie **plusieurs** solutions, laquelle prendre ? On les **classe** selon des critères :

- **continuité** — la plus proche de la pose actuelle (mouvement minimal) ;
- **limites articulaires** — rester dans les butées ;
- **collisions** — écarter les configurations qui heurtent le robot ou la scène ;
- **distance aux singularités** — privilégier les poses « robustes ».

ROS 2 — Bootcamp — Jour 3 Manipulation

Quand l'IK renvoie **plusieurs** solutions, laquelle prendre ? On les **classe** selon des critères :

- **continuité** — la plus proche de la pose actuelle (mouvement minimal) ;
- **limites articulaires** — rester dans les butées ;
- **collisions** — écarter les configurations qui heurtent le robot ou la scène ;
- **distance aux singularités** — privilégier les poses « robustes ».

> 🎯 **Pas juste « une » solution**
>
> Trouver des angles ne suffit pas : il faut **la bonne** configuration pour le contexte. C'est ce tri que MoveIt fait à votre place.

Résoudre l’IK : trois familles de méthodes

Résoudre l'IK : trois familles de méthodes

📐 Analytique

Équations trigonométriques **exactes**. Rapide et précis, mais difficile à généraliser aux bras complexes.

Résoudre l'IK : trois familles de méthodes

📐 Analytique

Équations trigonométriques **exactes**. Rapide et précis, mais difficile à généraliser aux bras complexes.

🔢 Numérique

Itère via le Jacobien : $\Delta\theta = J^{+} \cdot \Delta x$. Général, mais peut **échouer près des singularités**.

Résoudre l'IK : trois familles de méthodes

📐 Analytique

Équations trigonométriques **exactes**. Rapide et précis, mais difficile à généraliser aux bras complexes.

🔢 Numérique

Itère via le Jacobien : $\Delta\theta = J^{+} \cdot \Delta x$. Général, mais peut **échouer près des singularités**.

🧠 Apprentissage / IA

Un réseau entraîné à résoudre l'IK. Rapide et généralisable, au prix de **beaucoup de données**.

Résoudre l'IK : trois familles de méthodes

📐 Analytique

Équations trigonométriques **exactes**. Rapide et précis, mais difficile à généraliser aux bras complexes.

🔢 Numérique

Itère via le Jacobien : $\Delta\theta = J^{+} \cdot \Delta x$. Général, mais peut **échouer près des singularités**.

🧠 Apprentissage / IA

Un réseau entraîné à résoudre l'IK. Rapide et généralisable, au prix de **beaucoup de données**.

⚙️ **En pratique dans MoveIt**

Le solveur par défaut est **numérique** : `KDL` (ou `TracIK`, plus robuste). C'est lui qui fait ce calcul pour vous, à chaque planification.

PARTIE 03 · MODÉLISATION DANS ROS 2

URDF · Xacro · SRDF · TF

Comment ROS 2 « connaît » le robot : sa géométrie, sa sémantique, ses repères.

Trois fichiers, trois rôles

Pour qu'un logiciel raisonne sur le bras, il faut le **décrire**. ROS 2 sépare cette description en couches complémentaires :

Trois fichiers, trois rôles

Pour qu'un logiciel raisonne sur le bras, il faut le **décrire**. ROS 2 sépare cette description en couches complémentaires :

🧱 URDF

La **géométrie** : links, joints, formes, masses. *« À quoi ressemble le robot ? »*

Trois fichiers, trois rôles

Pour qu'un logiciel raisonne sur le bras, il faut le **décrire**. ROS 2 sépare cette description en couches complémentaires :

🧱 URDF

La **géométrie** : links, joints, formes, masses. *« À quoi ressemble le robot ? »*

🏷️ SRDF

La **sémantique** pour la planification : groupes, poses, collisions. *« Comment on le pilote ? »*

Trois fichiers, trois rôles

Pour qu'un logiciel raisonne sur le bras, il faut le **décrire**. ROS 2 sépare cette description en couches complémentaires :

🧱 URDF

La **géométrie** : links, joints, formes, masses. *« À quoi ressemble le robot ? »*

🏷️ SRDF

La **sémantique** pour la planification : groupes, poses, collisions. *« Comment on le pilote ? »*

🌳 TF

Les **repères** et leurs relations spatiales, dans le temps. *« Où est chaque pièce, maintenant ? »*

Trois fichiers, trois rôles

Pour qu'un logiciel raisonne sur le bras, il faut le **décrire**. ROS 2 sépare cette description en couches complémentaires :

🧱 URDF

La **géométrie** : links, joints, formes, masses. *« À quoi ressemble le robot ? »*

🏷️ SRDF

La **sémantique** pour la planification : groupes, poses, collisions. *« Comment on le pilote ? »*

🌳 TF

Les **repères** et leurs relations spatiales, dans le temps. *« Où est chaque pièce, maintenant ? »*

> URDF + SRDF sont **statiques** (des fichiers). TF est **dynamique** : il vit à l'exécution et change à chaque mouvement.

ROS 2 — Bootcamp — Jour 3 Manipulation

Le **URDF** (*Unified Robot Description Format*) est un fichier **XML** qui décrit la structure du robot :

ROS 2 — Bootcamp — Jour 3 Manipulation

Le **URDF** (*Unified Robot Description Format*) est un fichier **XML** qui décrit la structure du robot :

- les `link` (segments) et les `joint` (axes + limites) ;

ROS 2 — Bootcamp — Jour 3 Manipulation

Le **URDF** (*Unified Robot Description Format*) est un fichier **XML** qui décrit la structure du robot :

- les `link` (segments) et les `joint` (axes + limites) ;
- des **formes** (box, cylinder, sphere) ou des **meshes** ;

ROS 2 — Bootcamp — Jour 3 Manipulation

Le **URDF** (*Unified Robot Description Format*) est un fichier **XML** qui décrit la structure du robot :

- les `link` (segments) et les `joint` (axes + limites) ;
- des **formes** (box, cylinder, sphere) ou des **meshes** ;
- deux modèles par link : **visuel** (le rendu) et **collision** (souvent simplifié).

ROS 2 — Bootcamp — Jour 3 Manipulation

Le **URDF** (*Unified Robot Description Format*) est un fichier **XML** qui décrit la structure du robot :

- les `link` (segments) et les `joint` (axes + limites) ;
- des **formes** (box, cylinder, sphere) ou des **meshes** ;
- deux modèles par link : **visuel** (le rendu) et **collision** (souvent simplifié).

Il est publié sur le topic `/robot_description` et consommé par **RViz**, **Gazebo** et **MoveIt**.

ROS 2 — Bootcamp — Jour 3 Manipulation

Le **URDF** (*Unified Robot Description Format*) est un fichier **XML** qui décrit la structure du robot :

- les `link` (segments) et les `joint` (axes + limites) ;
- des **formes** (box, cylinder, sphere) ou des **meshes** ;
- deux modèles par link : **visuel** (le rendu) et **collision** (souvent simplifié).

Il est publié sur le topic `/robot_description` et consommé par **RViz**, **Gazebo** et **MoveIt**.

> 🧩 **Visuel ≠ collision**
>
> Le maillage **visuel** peut être détaillé ; le maillage de **collision** est simplifié (boîtes, cylindres) pour que la détection de collisions reste rapide.

ROS 2 — Bootcamp — Jour 3
Manipulation

Écrire un URDF à la main devient vite **répétitif** (6 joints quasi identiques…). **Xacro** génère le XML à partir de **macros**, **propriétés** et **includes**.

```
<xacro:property name="arm_len" value="0.12"/>

<xacro:macro name="segment" params="name len">
  <link name="${name}">
    <visual>
      <geometry>
        <cylinder length="${len}" radius="0.02"/>
      </geometry>
    </visual>
  </link>
</xacro:macro>

<!— une ligne → un link complet —>
<xacro:segment name="elbow" len="${arm_len}"/>
```

ROS 2 — Bootcamp — Jour 3 Manipulation

Écrire un URDF à la main devient vite **répétitif** (6 joints quasi identiques…). **Xacro** génère le XML à partir de **macros**, **propriétés** et **includes**.

- **propriétés** réutilisables (longueurs, limites) ;

```
<xacro:property name="arm_len" value="0.12"/>

<xacro:macro name="segment" params="name len">
  <link name="${name}">
    <visual>
      <geometry>
        <cylinder length="${len}" radius="0.02"/>
      </geometry>
    </visual>
  </link>
</xacro:macro>

<!— une ligne → un link complet —>
<xacro:segment name="elbow" len="${arm_len}"/>
```

ROS 2 — Bootcamp — Jour 3 Manipulation

Écrire un URDF à la main devient vite **répétitif** (6 joints quasi identiques…). **Xacro** génère le XML à partir de **macros**, **propriétés** et **includes**.

- **propriétés** réutilisables (longueurs, limites) ;
- **macros** : un patron de link/joint instancié N fois ;

```
<xacro:property name="arm_len" value="0.12"/>

<xacro:macro name="segment" params="name len">
  <link name="${name}">
    <visual>
      <geometry>
        <cylinder length="${len}" radius="0.02"/>
      </geometry>
    </visual>
  </link>
</xacro:macro>

<!— une ligne → un link complet —>
<xacro:segment name="elbow" len="${arm_len}"/>
```

ROS 2 — Bootcamp — Jour 3 Manipulation

Écrire un URDF à la main devient vite **répétitif** (6 joints quasi identiques…). **Xacro** génère le XML à partir de **macros**, **propriétés** et **includes**.

- **propriétés** réutilisables (longueurs, limites) ;
- **macros** : un patron de link/joint instancié N fois ;
- **arguments** : un même `.xacro` produit des **variantes** (sim vs réel, avec/sans pince).

```
<xacro:property name="arm_len" value="0.12"/>

<xacro:macro name="segment" params="name len">
  <link name="${name}">
    <visual>
      <geometry>
        <cylinder length="${len}" radius="0.02"/>
      </geometry>
    </visual>
  </link>
</xacro:macro>

<!— une ligne → un link complet —>
<xacro:segment name="elbow" len="${arm_len}"/>
```

ROS 2 — Bootcamp — Jour 3 Manipulation

Écrire un URDF à la main devient vite **répétitif** (6 joints quasi identiques…). **Xacro** génère le XML à partir de **macros**, **propriétés** et **includes**.

- **propriétés** réutilisables (longueurs, limites) ;
- **macros** : un patron de link/joint instancié N fois ;
- **arguments** : un même `.xacro` produit des **variantes** (sim vs réel, avec/sans pince).

> Au lancement, Xacro est **développé en URDF pur** : c'est toujours ce dernier que ROS 2 consomme.

```
<xacro:property name="arm_len" value="0.12"/>

<xacro:macro name="segment" params="name len">
  <link name="${name}">
    <visual>
      <geometry>
        <cylinder length="${len}" radius="0.02"/>
      </geometry>
    </visual>
  </link>
</xacro:macro>

<!— une ligne → un link complet —>
<xacro:segment name="elbow" len="${arm_len}"/>
```

ROS 2 — Bootcamp — Jour 3 Manipulation

L'URDF décrit la **géométrie**. Le **SRDF** (*Semantic Robot Description Format*) ajoute ce qu'il faut pour **planifier**, généré par le *MoveIt Setup Assistant* :

ROS 2 — Bootcamp — Jour 3 Manipulation

L'URDF décrit la **géométrie**. Le **SRDF** (*Semantic Robot Description Format*) ajoute ce qu'il faut pour **planifier**, généré par le *MoveIt Setup Assistant* :

- **groupes** : `arm` (chaîne `base_link → gripper_link`), `gripper` (joint `gripper`) ;

ROS 2 — Bootcamp — Jour 3 Manipulation

L'URDF décrit la **géométrie**. Le **SRDF** (*Semantic Robot Description Format*) ajoute ce qu'il faut pour **planifier**, généré par le *MoveIt Setup Assistant* :

- **groupes** : `arm` (chaîne `base_link → gripper_link`), `gripper` (joint `gripper`) ;
- **poses nommées** : `home`, `ready`, `gripper_open`, `gripper_close` ;

ROS 2 — Bootcamp — Jour 3 Manipulation

L'URDF décrit la **géométrie**. Le **SRDF** (*Semantic Robot Description Format*) ajoute ce qu'il faut pour **planifier**, généré par le *MoveIt Setup Assistant* :

- **groupes** : `arm` (chaîne `base_link → gripper_link`), `gripper` (joint `gripper`) ;
- **poses nommées** : `home`, `ready`, `gripper_open`, `gripper_close` ;
- **end-effector** : `gripper_ee` (parent `gripper_link`) ;

ROS 2 — Bootcamp — Jour 3 Manipulation

L'URDF décrit la **géométrie**. Le **SRDF** (*Semantic Robot Description Format*) ajoute ce qu'il faut pour **planifier**, généré par le *MoveIt Setup Assistant* :

- **groupes** : `arm` (chaîne `base_link → gripper_link`), `gripper` (joint `gripper`) ;
- **poses nommées** : `home`, `ready`, `gripper_open`, `gripper_close` ;
- **end-effector** : `gripper_ee` (parent `gripper_link`) ;
- **matrice de collisions désactivées** (paires toujours/jamais en contact).

ROS 2 — Bootcamp — Jour 3 Manipulation

L'URDF décrit la **géométrie**. Le **SRDF** (*Semantic Robot Description Format*) ajoute ce qu'il faut pour **planifier**, généré par le *MoveIt Setup Assistant* :

- **groupes** : `arm` (chaîne `base_link → gripper_link`), `gripper` (joint `gripper`) ;
- **poses nommées** : `home`, `ready`, `gripper_open`, `gripper_close` ;
- **end-effector** : `gripper_ee` (parent `gripper_link`) ;
- **matrice de collisions désactivées** (paires toujours/jamais en contact).

🚫 **Pourquoi désactiver des collisions ?**

Des links **toujours en contact** (voisins dans la chaîne) ou dont la collision est **géométriquement impossible** sont ignorés. Sinon MoveIt croirait le robot constamment en collision avec lui-même.

ROS 2 — Bootcamp — Jour 3 Manipulation

TF2 gère les relations spatiales (**position + orientation**) entre tous les repères : épaule, coude, poignet, pince… et les objets de la scène.

ROS 2 — Bootcamp — Jour 3 Manipulation

TF2 gère les relations spatiales (**position + orientation**) entre tous les repères : épaule, coude, poignet, pince… et les objets de la scène. Chaque arête est une transformation ; les composer donne la pose **de n'importe quel repère dans n'importe quel autre**.

ROS 2 — Bootcamp — Jour 3 Manipulation

TF2 gère les relations spatiales (**position + orientation**) entre tous les repères : épaule, coude, poignet, pince… et les objets de la scène. Chaque arête est une transformation ; les composer donne la pose **de n'importe quel repère dans n'importe quel autre**.

⏱️ **Pourquoi un buffer dans le temps ?**

TF garde un **historique** des transformations. On peut demander « où était la pince **à l'instant t** ? » — indispensable pour recaler une **mesure capteur** arrivée avec un peu de retard sur la pose du robot au moment exact de la mesure.

L’arbre TF du SO-101

Chaque nœud est un repère, chaque arête une transformation. En les composant on obtient la pose de **n’importe quel repère dans n’importe quel autre**.

L’arbre TF du SO-101

Chaque nœud est un repère, chaque arête une transformation. En les composant on obtient la pose de **n’importe quel repère dans n’importe quel autre**.

> Même arbre TF qu’au **Jour 2** : la base mobile LeKiwi et le bras partagent le même mécanisme de repères.

Qui publie la TF du bras ?

Le nœud `robot_state_publisher` fait le lien entre la description statique et les repères vivants :

Qui publie la TF du bras ?

Le nœud `robot_state_publisher` fait le lien entre la description statique et les repères vivants :

📄 Entrée 1 — URDF

La structure : quels links, quels joints, quels axes.

Qui publie la TF du bras ?

Le nœud `robot_state_publisher` fait le lien entre la description statique et les repères vivants :

📄 Entrée 1 — URDF

La structure : quels links, quels joints, quels axes.

📡 Entrée 2 — `/joint_states`

Les angles courants de chaque joint, publiés en continu.

Qui publie la TF du bras ?

Le nœud `robot_state_publisher` fait le lien entre la description statique et les repères vivants :

📄 Entrée 1 — URDF

La structure : quels links, quels joints, quels axes.

📡 Entrée 2 — `/joint_states`

Les angles courants de chaque joint, publiés en continu.

🌳 Sortie — TF

Il applique la FK et publie la pose de **chaque repère**.

Qui publie la TF du bras ?

Le nœud `robot_state_publisher` fait le lien entre la description statique et les repères vivants :

📄 Entrée 1 — URDF

La structure : quels links, quels joints, quels axes.

📡 Entrée 2 — `/joint_states`

Les angles courants de chaque joint, publiés en continu.

🌳 Sortie — TF

Il applique la FK et publie la pose de **chaque repère**.

> En clair : **URDF (forme) +** `/joint_states` **(angles) → FK → arbre TF**. C'est la cinématique directe de la Partie 02, appliquée en continu.

Commander le matériel — `ros2_control`

Entre « je veux cette trajectoire » et les moteurs, une couche standard **uniforme et temps réel** : `ros2_control`. Trois briques :

Commander le matériel — `ros2_control`

Entre « je veux cette trajectoire » et les moteurs, une couche standard **uniforme et temps réel** : `ros2_control`. Trois briques :

🔌 Hardware interface

Le pont bas-niveau : **lit** l'état, **écrit** les commandes. `gz_ros2_control` en sim, driver Feetech en réel.

Commander le matériel — `ros2_control`

Entre « je veux cette trajectoire » et les moteurs, une couche standard **uniforme et temps réel** : `ros2_control`. Trois briques :

🔌 Hardware interface

Le pont bas-niveau : **lit** l'état, **écrit** les commandes. `gz_ros2_control` en sim, driver Feetech en réel.

🎛️ Controller manager

Charge les contrôleurs et **arbitre** l'accès aux joints.

Commander le matériel — `ros2_control`

Entre « je veux cette trajectoire » et les moteurs, une couche standard **uniforme et temps réel** : `ros2_control`. Trois briques :

🔌 Hardware interface

Le pont bas-niveau : **lit** l'état, **écrit** les commandes. `gz_ros2_control` en sim, driver Feetech en réel.

🎛️ Controller manager

Charge les contrôleurs et **arbitre** l'accès aux joints.

🧩 Contrôleurs

JTC (suivre une trajectoire) pour le bras, un contrôleur de **pince** dédié.

`ros2_control` — command vs state

Deux types d'**interfaces** entre les contrôleurs et le matériel :

`ros2_control` — command vs state

Deux types d'**interfaces** entre les contrôleurs et le matériel :

✍️ command interfaces

Ce qu'on **écrit** : les **consignes** envoyées au matériel (position, vitesse, effort).

`ros2_control` — command vs state

Deux types d'**interfaces** entre les contrôleurs et le matériel :

✍️ command interfaces

Ce qu'on **écrit** : les **consignes** envoyées au matériel (position, vitesse, effort).

📖 state interfaces

Ce qu'on **lit** : les **mesures** remontées par le matériel (position, vitesse réelles).

`ros2_control` — command vs state

Deux types d'**interfaces** entre les contrôleurs et le matériel :

✍️ command interfaces

Ce qu'on **écrit** : les **consignes** envoyées au matériel (position, vitesse, effort).

📖 state interfaces

Ce qu'on **lit** : les **mesures** remontées par le matériel (position, vitesse réelles).

🔁 **La boucle se referme**

Le `joint_state_broadcaster` republie l'état sur `/joint_states` — repris par TF et MoveIt. Et **MoveIt envoie ses trajectoires à ce JTC.**

PARTIE 04 · PLANIFICATION & MOVEIT 2

Planifier un mouvement

Trouver un chemin faisable, sans collision — puis l'orchestrer avec MoveIt 2.

Le problème de la planification

Aller d’une configuration de départ à une cible, ce n’est pas une ligne droite dans l’espace des joints :

Le problème de la planification

Aller d’une configuration de départ à une cible, ce n’est pas une ligne droite dans l’espace des joints :

🧱 Sans collision

Éviter les obstacles, la scène… et le robot lui-même.

Le problème de la planification

Aller d'une configuration de départ à une cible, ce n'est pas une ligne droite dans l'espace des joints :

🧱 Sans collision

Éviter les obstacles, la scène… et le robot lui-même.

📏 Dans les limites

Respecter butées, vitesses et efforts de chaque joint.

Le problème de la planification

Aller d'une configuration de départ à une cible, ce n'est pas une ligne droite dans l'espace des joints :

🧱 Sans collision

Éviter les obstacles, la scène… et le robot lui-même.

📏 Dans les limites

Respecter butées, vitesses et efforts de chaque joint.

✅ Faisable

Une trajectoire continue, réellement exécutable par le bras.

Le problème de la planification

Aller d'une configuration de départ à une cible, ce n'est pas une ligne droite dans l'espace des joints :

🧱 Sans collision

Éviter les obstacles, la scène… et le robot lui-même.

📏 Dans les limites

Respecter butées, vitesses et efforts de chaque joint.

✅ Faisable

Une trajectoire continue, réellement exécutable par le bras.

> L'espace des configurations est **vaste et plein de trous** (les collisions). On délègue la recherche à une bibliothèque spécialisée : **OMPL**.

ROS 2 — Bootcamp — Jour 3 Manipulation

OMPL **cherche une trajectoire faisable** entre deux configurations en respectant les contraintes et en évitant les obstacles.

ROS 2 — Bootcamp — Jour 3 Manipulation

OMPL **cherche une trajectoire faisable** entre deux configurations en respectant les contraintes et en évitant les obstacles.

- algorithmes par **échantillonnage** : **RRT**, **PRM**, **KPIECE**, **LBTRRT**… ;

ROS 2 — Bootcamp — Jour 3 Manipulation

OMPL **cherche une trajectoire faisable** entre deux configurations en respectant les contraintes et en évitant les obstacles.

- algorithmes par **échantillonnage** : **RRT**, **PRM**, **KPIECE**, **LBTRRT**… ;
- la plupart sont **probabilistes** : ils tirent des configurations au hasard ;

ROS 2 — Bootcamp — Jour 3 Manipulation

OMPL **cherche une trajectoire faisable** entre deux configurations en respectant les contraintes et en évitant les obstacles.

- algorithmes par **échantillonnage** : **RRT**, **PRM**, **KPIECE**, **LBTRRT**… ;
- la plupart sont **probabilistes** : ils tirent des configurations au hasard ;
- conséquence : résultats **non déterministes** (deux runs ≠ même chemin).

ROS 2 — Bootcamp — Jour 3 Manipulation

OMPL **cherche une trajectoire faisable** entre deux configurations en respectant les contraintes et en évitant les obstacles.

- algorithmes par **échantillonnage** : **RRT**, **PRM**, **KPIECE**, **LBTRRT**… ;
- la plupart sont **probabilistes** : ils tirent des configurations au hasard ;
- conséquence : résultats **non déterministes** (deux runs ≠ même chemin).

Échantillonner, pas calculer

Plutôt que résoudre une équation, OMPL **tire des configurations** et relie celles qui sont sans collision jusqu'à atteindre le but. Robuste sur les bras complexes.

ROS 2 — Bootcamp — Jour 3 Manipulation

OMPL **cherche une trajectoire faisable** entre deux configurations en respectant les contraintes et en évitant les obstacles.

- algorithmes par **échantillonnage** : **RRT**, **PRM**, **KPIECE**, **LBTRRT**… ;
- la plupart sont **probabilistes** : ils tirent des configurations au hasard ;
- conséquence : résultats **non déterministes** (deux runs ≠ même chemin).

Échantillonner, pas calculer

Plutôt que résoudre une équation, OMPL **tire des configurations** et relie celles qui sont sans collision jusqu'à atteindre le but. Robuste sur les bras complexes.

Alternative orientée **optimisation** : **STOMP** (lisse une trajectoire bruitée).

ROS 2 — Bootcamp — Jour 3 Manipulation

MoveIt 2 est la boîte à outils ROS 2 de la manipulation. Elle **centralise** tout ce qu'on a vu :

ROS 2 — Bootcamp — Jour 3 Manipulation

MoveIt 2 est la boîte à outils ROS 2 de la manipulation. Elle **centralise** tout ce qu'on a vu :

🧮 Cinématique

FK et IK (KDL / TracIK).

ROS 2 — Bootcamp — Jour 3 Manipulation

MoveIt 2 est la boîte à outils ROS 2 de la manipulation. Elle **centralise** tout ce qu'on a vu :

🧮 Cinématique

FK et IK (KDL / TracIK).

🗺️ Planification

Via OMPL / STOMP.

ROS 2 — Bootcamp — Jour 3 Manipulation

MoveIt 2 est la boîte à outils ROS 2 de la manipulation. Elle **centralise** tout ce qu'on a vu :

🧮 Cinématique

FK et IK (KDL / TracIK).

🗺️ Planification

Via OMPL / STOMP.

🛡️ Collisions

Vérif. robot ↔ scène en continu.

ROS 2 — Bootcamp — Jour 3 Manipulation

MoveIt 2 est la boîte à outils ROS 2 de la manipulation. Elle **centralise** tout ce qu'on a vu :

🧮 Cinématique

FK et IK (KDL / TracIK).

🗺️ Planification

Via OMPL / STOMP.

🛡️ Collisions

Vérif. robot ↔ scène en continu.

👁️ Capteurs

Intègre la perception dans la scène.

ROS 2 — Bootcamp — Jour 3
Manipulation

MoveIt 2 est la boîte à outils ROS 2 de la manipulation. Elle **centralise** tout ce qu'on a vu :

S'appuie sur **URDF + SRDF**, raisonne via **TF**, et s'intègre à **RViz** et **Gazebo**.

🧮 Cinématique

FK et IK (KDL / TracIK).

🗺️ Planification

Via OMPL / STOMP.

🛡️ Collisions

Vérif. robot ↔ scène en continu.

👁️ Capteurs

Intègre la perception dans la scène.

ROS 2 — Bootcamp — Jour 3 Manipulation

MoveIt 2 est la boîte à outils ROS 2 de la manipulation. Elle **centralise** tout ce qu'on a vu :

🧮 Cinématique

FK et IK (KDL / TracIK).

🗺️ Planification

Via OMPL / STOMP.

🛡️ Collisions

Vérif. robot ↔ scène en continu.

👁️ Capteurs

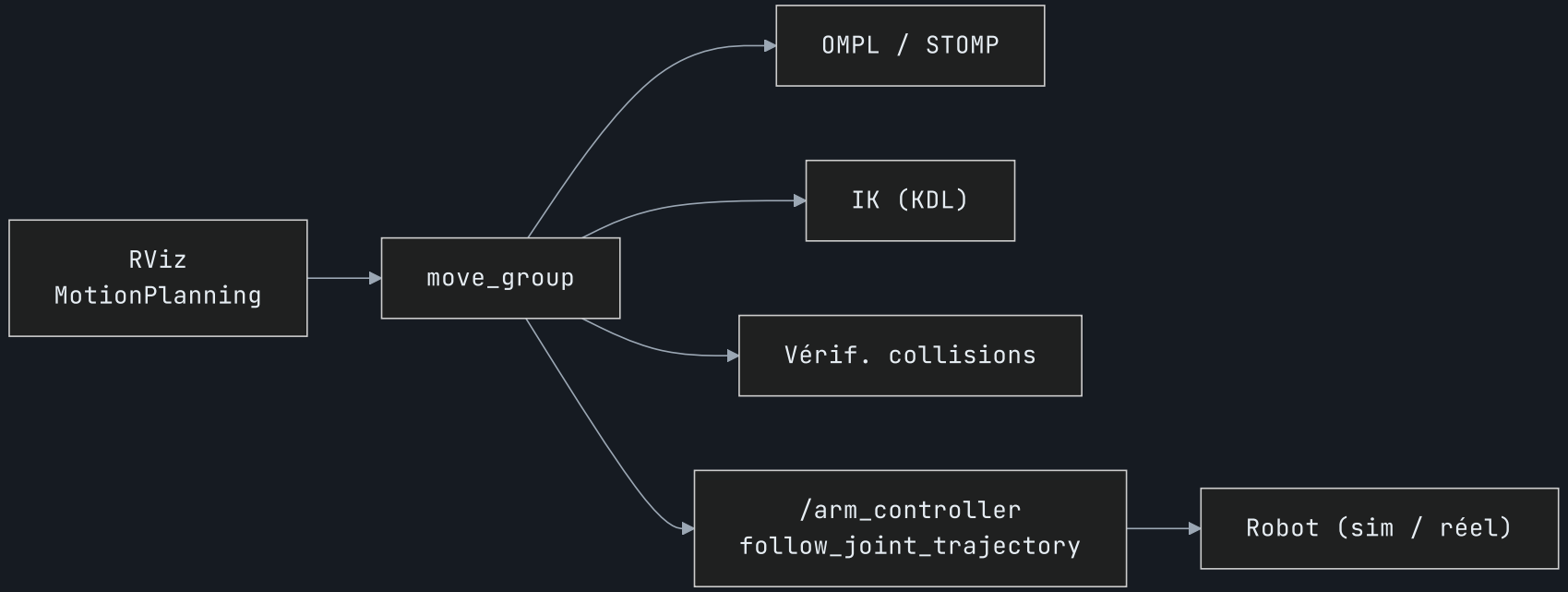
Intègre la perception dans la scène.

S'appuie sur **URDF + SRDF**, raisonne via **TF**, et s'intègre à **RViz** et **Gazebo**.

> Plutôt que recoder solveurs et détection de collisions, on **assemble des briques éprouvées**.

Le pipeline MoveIt 2

Au cœur du pipeline, `move_group` reçoit une cible et distribue le travail aux briques spécialisées :



Le pipeline MoveIt 2 — déroulé

Le pipeline MoveIt 2 — déroulé

1 · Cible

Une **cible** arrive (RViz ou code) → `move_group` orchestre.

Le pipeline MoveIt 2 — déroulé

1 · Cible

Une **cible** arrive (RViz ou code) → `move_group` orchestre.

2 · Calcul

OMPL cherche un chemin, l'**IK** convertit les poses, les **collisions** sont vérifiées.

Le pipeline MoveIt 2 — déroulé

1 · Cible

Une **cible** arrive (RViz ou code) → `move_group` orchestre.

2 · Calcul

OMPL cherche un chemin, l'**IK** convertit les poses, les **collisions** sont vérifiées.

3 · Exécution

La trajectoire validée part au **contrôleur** ; la pince est pilotée à part.

Le pipeline MoveIt 2 — déroulé

1 · Cible

Une **cible** arrive (RViz ou code) → `move_group` orchestre.

2 · Calcul

OMPL cherche un chemin, l'**IK** convertit les poses, les **collisions** sont vérifiées.

3 · Exécution

La trajectoire validée part au **contrôleur** ; la pince est pilotée à part.

> Une **cible** entre, une **trajectoire validée** sort — le reste est délégué aux briques éprouvées.

ROS 2 — Bootcamp — Jour 3 Manipulation

Le nœud central qui **assemble** toutes les briques pour répondre à une demande de planification :

🧠 **move_group**
orchestre toute la planification

📄 **URDF + SRDF**
structure & sémantique

🌳 **TF**
pose courante des repères

🗺️ **OMPL / IK**
chemin & cinématique

ROS 2 — Bootcamp — Jour 3 Manipulation

Le nœud central qui **assemble** toutes les briques pour répondre à une demande de planification :

- charge **URDF + SRDF** (structure + sémantique) ;

🧠 **move_group**
orchestre toute la planification

📄 URDF + SRDF
structure & sémantique

🌳 TF
pose courante des repères

🗺️ OMPL / IK
chemin & cinématique

ROS 2 — Bootcamp — Jour 3 Manipulation

Le nœud central qui **assemble** toutes les briques pour répondre à une demande de planification :

- charge **URDF + SRDF** (structure + sémantique) ;
- interroge **TF** pour la pose courante ;

🧠 **move_group**
orchestre toute la planification

📄 URDF + SRDF
structure & sémantique

🌳 TF
pose courante des repères

🗺️ OMPL / IK
chemin & cinématique

ROS 2 — Bootcamp — Jour 3 Manipulation

Le nœud central qui **assemble** toutes les briques pour répondre à une demande de planification :

- charge **URDF + SRDF** (structure + sémantique) ;
- interroge **TF** pour la pose courante ;
- lance **OMPL** (chemin) + **l'IK** (poses cartésiennes) ;

ROS 2 — Bootcamp — Jour 3 Manipulation

Le nœud central qui **assemble** toutes les briques pour répondre à une demande de planification :

- charge **URDF + SRDF** (structure + sémantique) ;
- interroge **TF** pour la pose courante ;
- lance **OMPL** (chemin) + **l'IK** (poses cartésiennes) ;
- **valide** chaque état (collisions, limites), puis **transmet** au contrôleur.

Les interfaces exposées

`move_group` communique avec le reste du système via les **trois mécanismes ROS 2** :

Les interfaces exposées

`move_group` communique avec le reste du système via les **trois mécanismes ROS 2** :

📨 Topics

Flux continus : `/joint_states`, l'arbre TF, la **planning scene** (objets de l'environnement).

Les interfaces exposées

`move_group` communique avec le reste du système via les **trois mécanismes ROS 2** :

📨 Topics

Flux continus : `/joint_states`, l'arbre TF, la **planning scene** (objets de l'environnement).

🛎️ Services

Requêtes ponctuelles : calcul d'**IK/FK**, vérification de collision, état du robot.

Les interfaces exposées

`move_group` communique avec le reste du système via les **trois mécanismes ROS 2** :

📨 Topics

Flux continus : `/joint_states`, l'arbre TF, la **planning scene** (objets de l'environnement).

🛎️ Services

Requêtes ponctuelles : calcul d'**IK/FK**, vérification de collision, état du robot.

🎬 Actions

Tâches longues avec feedback : **planifier & exécuter** une trajectoire (annulable).

Les interfaces exposées

`move_group` communique avec le reste du système via les **trois mécanismes ROS 2** :

📨 Topics

Flux continus : `/joint_states`, l'arbre TF, la **planning scene** (objets de l'environnement).

🛎️ Services

Requêtes ponctuelles : calcul d'**IK/FK**, vérification de collision, état du robot.

🎬 Actions

Tâches longues avec feedback : **planifier & exécuter** une trajectoire (annulable).

> Mêmes primitives qu'au **Jour 1** (topics / services / actions) : MoveIt n'invente rien, il les **compose** pour la manipulation.

Saisir un objet (pick & place)

Planifier un mouvement ne suffit pas à **saisir**. Quatre points à part :

Saisir un objet (pick & place)

Planifier un mouvement ne suffit pas à **saisir**. Quatre points à part :

🎯 Le bon repère (TCP)

On vise le **point de préhension** (`gripper_frame_link`), pas le corps de la pince — sinon les doigts tombent à côté.

Saisir un objet (pick & place)

Planifier un mouvement ne suffit pas à **saisir**. Quatre points à part :

🎯 Le bon repère (TCP)

On vise le **point de préhension** (`gripper_frame_link`), pas le corps de la pince — sinon les doigts tombent à côté.

⬇️ Approche / retrait

On descend **par le haut** (pré-prise au-dessus), on saisit, on **remonte** avant de partir — pour ne pas heurter l'objet.

Saisir un objet (pick & place)

Planifier un mouvement ne suffit pas à **saisir**. Quatre points à part :

🎯 Le bon repère (TCP)

On vise le **point de préhension** (`gripper_frame_link`), pas le corps de la pince — sinon les doigts tombent à côté.

⬇️ Approche / retrait

On descend **par le haut** (pré-prise au-dessus), on saisit, on **remonte** avant de partir — pour ne pas heurter l'objet.

🧲 Attacher l'objet

À la fermeture, on **attache** l'objet à la pince dans la *planning scene* : il suit le bras et MoveIt gère ses collisions.

Saisir un objet (pick & place)

Planifier un mouvement ne suffit pas à **saisir**. Quatre points à part :

🎯 Le bon repère (TCP)

On vise le **point de préhension** (`gripper_frame_link`), pas le corps de la pince — sinon les doigts tombent à côté.

⬇️ Approche / retrait

On descend **par le haut** (pré-prise au-dessus), on saisit, on **remonte** avant de partir — pour ne pas heurter l'objet.

🧲 Attacher l'objet

À la fermeture, on **attache** l'objet à la pince dans la *planning scene* : il suit le bras et MoveIt gère ses collisions.

🔓 Relâcher

À l'ouverture, on le **détache** ; il est déposé.

Saisir un objet — sim ≠ réel

La saisie diffère selon l’environnement d’exécution :

Saisir un objet — sim ≠ réel

La saisie diffère selon l'environnement d'exécution :

🖥️ En simulation

La saisie tient par **friction** (fragile) ou par **attache** dans la *planning scene*.

Saisir un objet — sim ≠ réel

La saisie diffère selon l'environnement d'exécution :

🖥️ En simulation

La saisie tient par **friction** (fragile) ou par **attache** dans la *planning scene*.

🤖 Sur le vrai SO-101

C'est la **force des moteurs** qui maintient l'objet — pas d'attache magique.

Saisir un objet — sim ≠ réel

La saisie diffère selon l'environnement d'exécution :

🖥️ En simulation

La saisie tient par **friction** (fragile) ou par **attache** dans la *planning scene*.

🤖 Sur le vrai SO-101

C'est la **force des moteurs** qui maintient l'objet — pas d'attache magique.

🧪 **Contrainte des 5 DOF**

Sur 5 DOF on saisit **par le haut** (orientation imposée), pas sous tous les angles.

Récap Jour 3

Récap Jour 3

- **Anatomie** : un bras = chaîne série de joints ; 1 joint = 1 DDL ; le SO-101 = **5 DOF** + pince.

Récap Jour 3

- **Anatomie** : un bras = chaîne série de joints ; 1 joint = 1 DDL ; le SO-101 = **5 DOF** + pince.
- **Cinématique** : FK (unique) vs IK (0 / 1 / ∞, KDL) ; sur 5 DOF on **planifie en espace des joints**.

Récap Jour 3

- **Anatomie** : un bras = chaîne série de joints ; 1 joint = 1 DDL ; le SO-101 = **5 DOF** + pince.
- **Cinématique** : FK (unique) vs IK (0 / 1 / ∞, KDL) ; sur 5 DOF on **planifie en espace des joints**.
- **Modélisation** : URDF (géométrie), Xacro (paramétrage), SRDF (sémantique), TF (repères dans le temps).

Récap Jour 3

- **Anatomie** : un bras = chaîne série de joints ; 1 joint = 1 DDL ; le SO-101 = **5 DOF** + pince.
- **Cinématique** : FK (unique) vs IK (0 / 1 / ∞, KDL) ; sur 5 DOF on **planifie en espace des joints**.
- **Modélisation** : URDF (géométrie), Xacro (paramétrage), SRDF (sémantique), TF (repères dans le temps).
- **Exécution** : `ros2_control` (JTC + pince) pilote le matériel ; MoveIt lui envoie ses trajectoires.

Récap Jour 3

- **Anatomie** : un bras = chaîne série de joints ; 1 joint = 1 DDL ; le SO-101 = **5 DOF** + pince.
- **Cinématique** : FK (unique) vs IK (0 / 1 / ∞, KDL) ; sur 5 DOF on **planifie en espace des joints**.
- **Modélisation** : URDF (géométrie), Xacro (paramétrage), SRDF (sémantique), TF (repères dans le temps).
- **Exécution** : `ros2_control` (JTC + pince) pilote le matériel ; MoveIt lui envoie ses trajectoires.
- **Planification** : OMPL cherche le chemin, **MoveIt 2** orchestre via `move_group` ; saisie = TCP + approche + attache.

ROS 2 — BOOTCAMP

Questions ?

ROS 2 — Bootcamp · Perpignan 2026

ros2.etienne-schmitz.com