ROS 2 — BOOTCAMP

Jour 2 — Navigation

LeKiwi · slam_toolbox · Nav2

Etienne Schmitz

Au programme

1. Pourquoi **naviguer** : se localiser, planifier, avancer sans rien heurter
2. Les **capteurs de localisation** : IMU, odométrie, LiDAR, GPS-RTK, UWB
3. Le **SLAM** & l'**AMCL** : cartographier la carte, puis s'y localiser
4. L'**architecture de Nav2** : behavior tree, planner, controller, costmaps

PARTIE 01 · POURQUOI NAVIGUER ?

Accomplir une mission dans le monde réel

Se localiser, planifier, avancer sans rien heurter.

Pourquoi un robot doit-il naviguer ?

Un robot mobile navigue pour **accomplir une mission** dans un environnement réel :

Pourquoi un robot doit-il naviguer ?

Un robot mobile navigue pour **accomplir une mission** dans un environnement réel :

🧺 Livrer

Transporter un colis d'un point à un autre.

Pourquoi un robot doit-il naviguer ?

Un robot mobile navigue pour **accomplir une mission** dans un environnement réel :

🧺 Livrer

Transporter un colis d'un point à un autre.

🧹 Nettoyer

Couvrir toute la surface d'une pièce.

Pourquoi un robot doit-il naviguer ?

Un robot mobile navigue pour **accomplir une mission** dans un environnement réel :

🧺 Livrer

Transporter un colis d'un point à un autre.

🧹 Nettoyer

Couvrir toute la surface d'une pièce.

🗺️ Explorer

Cartographier un lieu inconnu.

Pourquoi un robot doit-il naviguer ?

Un robot mobile navigue pour **accomplir une mission** dans un environnement réel :

🧺 Livrer

Transporter un colis d'un point à un autre.

🧹 Nettoyer

Couvrir toute la surface d'une pièce.

🗺️ Explorer

Cartographier un lieu inconnu.

👣 Suivre

Accompagner une personne en mouvement.

Pourquoi un robot doit-il naviguer ?

Un robot mobile navigue pour **accomplir une mission** dans un environnement réel :

🧺 Livrer

Transporter un colis d'un point à un autre.

🧹 Nettoyer

Couvrir toute la surface d'une pièce.

🗺️ Explorer

Cartographier un lieu inconnu.

👣 Suivre

Accompagner une personne en mouvement.

> Dans tous les cas, le robot doit **se déplacer seul**, et **en sécurité**.

Naviguer, c’est répondre à 3 questions

Naviguer, c’est répondre à 3 questions

📍 Où suis-je ?

Localisation — estimer sa pose dans l'environnement.

Naviguer, c’est répondre à 3 questions

📍 Où suis-je ?

Localisation — estimer sa pose dans l'environnement.

🎯 Où aller ?

Planification — calculer un chemin vers l'objectif.

Naviguer, c’est répondre à 3 questions

📍 Où suis-je ?

Localisation — estimer sa pose dans l'environnement.

🎯 Où aller ?

Planification — calculer un chemin vers l'objectif.

🚧 Comment, sans heurter ?

Perception & contrôle — suivre le chemin en évitant les obstacles.

Naviguer, c’est répondre à 3 questions

📍 Où suis-je ?

Localisation — estimer sa pose dans l'environnement.

🎯 Où aller ?

Planification — calculer un chemin vers l'objectif.

🚧 Comment, sans heurter ?

Perception & contrôle — suivre le chemin en évitant les obstacles.

> Trois questions que la stack **Nav2** orchestre de bout en bout.

ROS 2 — Bootcamp — Jour 2 Navigation

Nav2 est la stack de navigation de ROS 2. Elle combine tout ce qu'il faut pour naviguer **de façon autonome** :

ROS 2 — Bootcamp — Jour 2 Navigation

Nav2 est la stack de navigation de ROS 2. Elle combine tout ce qu'il faut pour naviguer **de façon autonome** :

- 🧠 **se localiser** — SLAM / AMCL ;

ROS 2 — Bootcamp — Jour 2 Navigation

Nav2 est la stack de navigation de ROS 2. Elle combine tout ce qu'il faut pour naviguer **de façon autonome** :

- 🧠 **se localiser** — SLAM / AMCL ;
- 🗺️ **créer ou utiliser une carte** ;

ROS 2 — Bootcamp — Jour 2 Navigation

Nav2 est la stack de navigation de ROS 2. Elle combine tout ce qu'il faut pour naviguer **de façon autonome** :

- 🧠 **se localiser** — SLAM / AMCL ;
- 🗺️ **créer ou utiliser une carte** ;
- 📦 **planifier** un chemin — global *et* local ;

ROS 2 — Bootcamp — Jour 2 Navigation

Nav2 est la stack de navigation de ROS 2. Elle combine tout ce qu'il faut pour naviguer **de façon autonome** :

- 🧠 **se localiser** — SLAM / AMCL ;
- 🗺️ **créer ou utiliser une carte** ;
- 📦 **planifier** un chemin — global *et* local ;
- 📡 **fusionner les capteurs** — LiDAR, IMU, odométrie ;

ROS 2 — Bootcamp — Jour 2 Navigation

Nav2 est la stack de navigation de ROS 2. Elle combine tout ce qu'il faut pour naviguer **de façon autonome** :

- 🧠 **se localiser** — SLAM / AMCL ;
- 🗺️ **créer ou utiliser une carte** ;
- 📦 **planifier** un chemin — global *et* local ;
- 📡 **fusionner les capteurs** — LiDAR, IMU, odométrie ;
- ⚙️ **exécuter** les mouvements avec feedback.

ROS 2 — Bootcamp — Jour 2 Navigation

Nav2 est la stack de navigation de ROS 2. Elle combine tout ce qu'il faut pour naviguer **de façon autonome** :

- 🧠 **se localiser** — SLAM / AMCL ;
- 🗺️ **créer ou utiliser une carte** ;
- 📦 **planifier** un chemin — global *et* local ;
- 📡 **fusionner les capteurs** — LiDAR, IMU, odométrie ;
- ⚙️ **exécuter** les mouvements avec feedback.

> Nav2 orchestre ces briques pour un comportement **intelligent et adaptable**.

PARTIE 02 · SE LOCALISER

Capteurs & principes de localisation

« Où suis-je ? » — aucun capteur n’est parfait, on les combine.

Les capteurs de localisation

Avant le tour d’horizon, **quatre familles** répondent à « où suis-je ? » — chacune avec ses forces :

Les capteurs de localisation

Avant le tour d'horizon, **quatre familles** répondent à « où suis-je ? » — chacune avec ses forces :

🌀 IMU

Mouvement propre (gyroscope + accéléromètre).
Haute fréquence, mais **dérive**.

Les capteurs de localisation

Avant le tour d'horizon, **quatre familles** répondent à « où suis-je ? » — chacune avec ses forces :

🌀 IMU

Mouvement propre (gyroscope + accéléromètre). Haute fréquence, mais **dérive**.

🛞 Odométrie

Encodeurs + IMU fusionnés → pose **continue**, mais cumulative.

Les capteurs de localisation

Avant le tour d'horizon, **quatre familles** répondent à « où suis-je ? » — chacune avec ses forces :

🌀 IMU

Mouvement propre (gyroscope + accéléromètre). Haute fréquence, mais **dérive**.

🛞 Odométrie

Encodeurs + IMU fusionnés → pose **continue**, mais cumulative.

📡 Multilatération

Distances à des balises fixes — **UWB** (intérieur), **GPS-RTK** (extérieur).

Les capteurs de localisation

Avant le tour d'horizon, **quatre familles** répondent à « où suis-je ? » — chacune avec ses forces :

🌀 IMU

Mouvement propre (gyroscope + accéléromètre). Haute fréquence, mais **dérive**.

🛞 Odométrie

Encodeurs + IMU fusionnés → pose **continue**, mais cumulative.

📡 Multilatération

Distances à des balises fixes — **UWB** (intérieur), **GPS-RTK** (extérieur).

🔦 LiDAR

Profondeur par laser → obstacles et **cartographie (SLAM)**.

Les capteurs de localisation

Avant le tour d'horizon, **quatre familles** répondent à « où suis-je ? » — chacune avec ses forces :

🌀 IMU

Mouvement propre (gyroscope + accéléromètre). Haute fréquence, mais **dérive**.

🛞 Odométrie

Encodeurs + IMU fusionnés → pose **continue**, mais cumulative.

📡 Multilatération

Distances à des balises fixes — **UWB** (intérieur), **GPS-RTK** (extérieur).

🔦 LiDAR

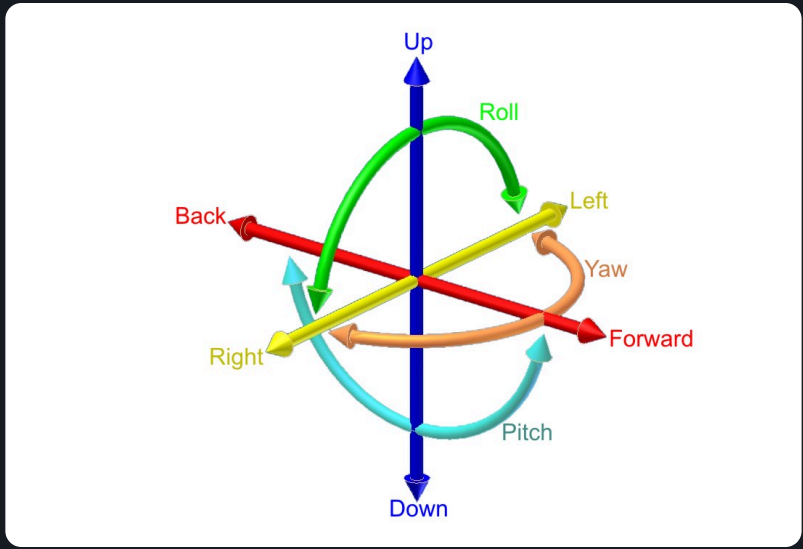
Profondeur par laser → obstacles et **cartographie (SLAM)**.

> On les **fusionne** : aucun capteur seul ne donne une pose fiable et absolue.

ROS 2 — Bootcamp — Jour 2 Navigation

Mesure le **mouvement propre** du robot :

- **vitesses angulaires** (gyroscope) ;
- **accélérations linéaires** (accéléromètre) ;
- parfois le **champ magnétique** (magnétomètre).

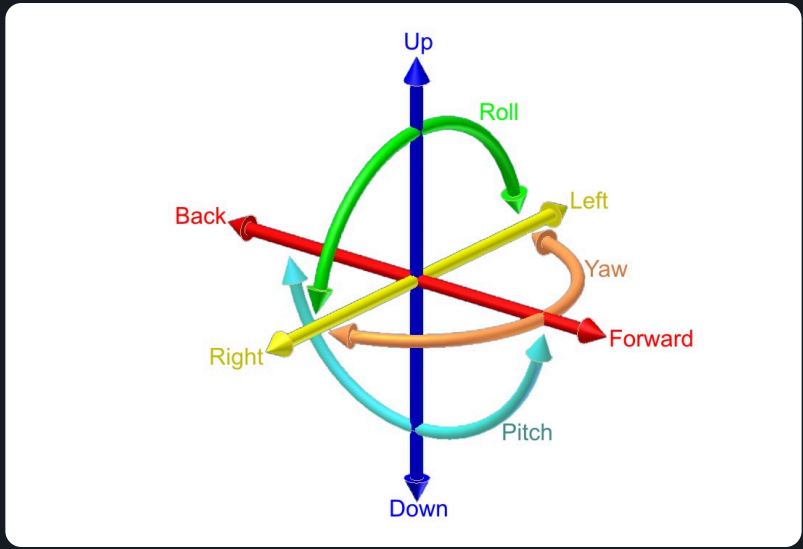


ROS 2 — Bootcamp — Jour 2
Navigation

Mesure le **mouvement propre** du robot :

- **vitesses angulaires** (gyroscope) ;
- **accélérations linéaires** (accéléromètre) ;
- parfois le **champ magnétique** (magnétomètre).

> ⚠️ **Dérive rapide**
>
> Intégrer une accélération cumule l'erreur. Utile en **fusion** et sur des mouvements **courts**, pas seule.

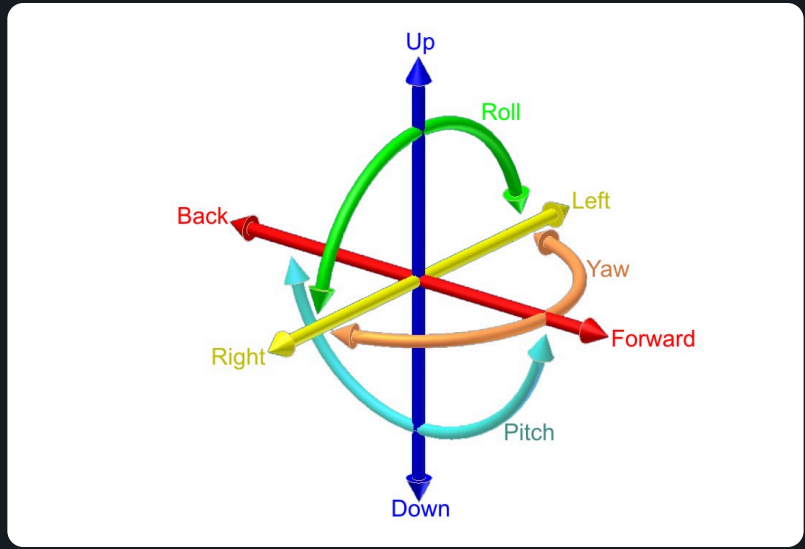


ROS 2 — Bootcamp — Jour 2 Navigation

Mesure le **mouvement propre** du robot :

- **vitesses angulaires** (gyroscope) ;
- **accélérations linéaires** (accéléromètre) ;
- parfois le **champ magnétique** (magnétomètre).

> ⚠️ **Dérive rapide**
>
> Intégrer une accélération cumule l'erreur. Utile en **fusion** et sur des mouvements **courts**, pas seule.



6 axes : 3 **rotations** (gyro) + 3 **accélérations** (accel).

ROS 2 — Bootcamp — Jour 2 Navigation

ROS 2 — Bootcamp — Jour 2 Navigation

ROS 2 — Bootcamp — Jour 2 Navigation

ROS 2 — Bootcamp — Jour 2 Navigation

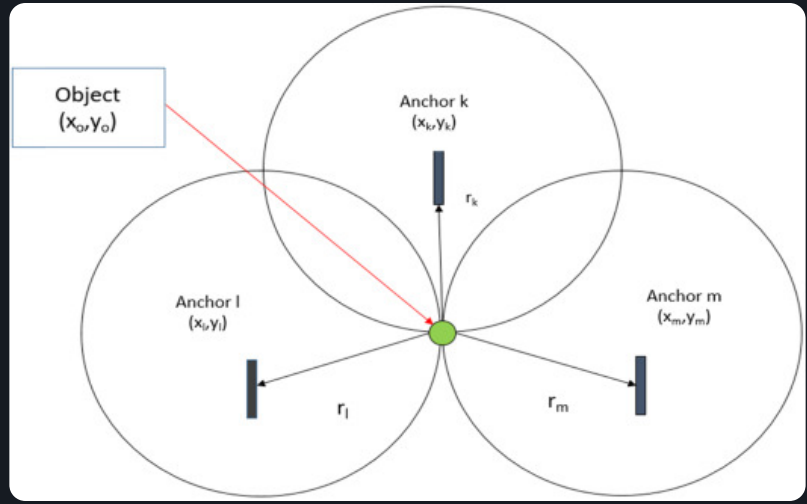
ROS 2 — Bootcamp — Jour 2 Navigation

ROS 2 — Bootcamp — Jour 2 Navigation

ROS 2 — Bootcamp — Jour 2 Navigation

Estime la position en mesurant les **distances** à plusieurs **stations fixes** :

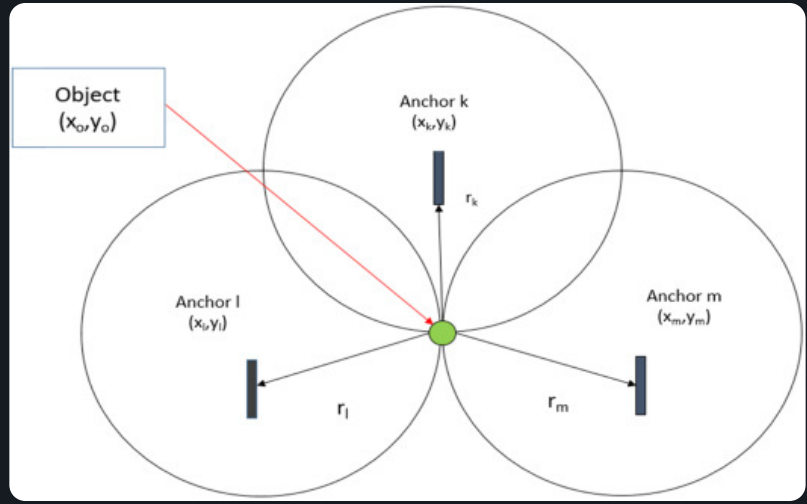
- **3** stations → localisation 2D ;
- **4** stations → localisation 3D.



ROS 2 — Bootcamp — Jour 2 Navigation

Estime la position en mesurant les **distances** à plusieurs **stations fixes** :

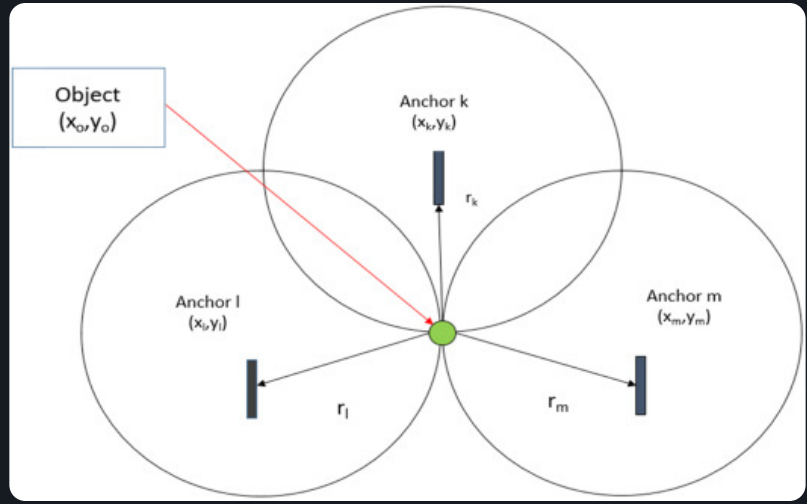
- **3** stations → localisation 2D ;
- **4** stations → localisation 3D.
- ⚠️ à ne pas confondre avec la **triangulation** (qui utilise des **angles**) ;



ROS 2 — Bootcamp — Jour 2 Navigation

Estime la position en mesurant les **distances** à plusieurs **stations fixes** :

- **3** stations → localisation 2D ;
- **4** stations → localisation 3D.
- ⚠️ à ne pas confondre avec la **triangulation** (qui utilise des **angles**) ;
- ⚠️ sensible aux **réflexions** de signal (rebonds, interférences).



GPS-RTK & UWB — un repère absolu

Deux technologies, **même principe** (distances à des points fixes connus), selon l'environnement :

GPS-RTK & UWB — un repère absolu

Deux technologies, **même principe** (distances à des points fixes connus), selon l'environnement :

🛰️ GPS-RTK — extérieur

Real Time Kinematic. GPS + **station de référence au sol**. Précision **centimétrique** temps réel. Exige une zone **dégagée** (agriculture, topo, véhicules autonomes).

GPS-RTK & UWB — un repère absolu

Deux technologies, **même principe** (distances à des points fixes connus), selon l'environnement :

🛰️ GPS-RTK — extérieur

Real Time Kinematic. GPS + **station de référence au sol**. Précision **centimétrique** temps réel. Exige une zone **dégagée** (agriculture, topo, véhicules autonomes).

📶 UWB — intérieur

Ultra Wide Band. **Ancres fixes** dans le bâtiment, mesure des **temps de vol** du signal. Précision **±10 à 30 cm** (usines, entrepôts).

GPS-RTK & UWB — un repère absolu

Deux technologies, **même principe** (distances à des points fixes connus), selon l'environnement :

🛰️ GPS-RTK — extérieur

Real Time Kinematic. GPS + **station de référence au sol**. Précision **centimétrique** temps réel. Exige une zone **dégagée** (agriculture, topo, véhicules autonomes).

📶 UWB — intérieur

Ultra Wide Band. **Ancres fixes** dans le bâtiment, mesure des **temps de vol** du signal. Précision **±10 à 30 cm** (usines, entrepôts).

🧭 **Le point commun**

Tous deux fournissent un **repère absolu** — ce qui manque à l'odométrie seule.

LiDAR — Light Detection and Ranging

Un **laser** mesure les **distances** à l'environnement → une carte de **profondeur**. Trois grandes familles :

LiDAR — Light Detection and Ranging

Un **laser** mesure les **distances** à l'environnement → une carte de **profondeur**. Trois grandes familles :

📍 **Fixe**

un faisceau orienté, distance sur un seul axe.

LiDAR — Light Detection and Ranging

Un **laser** mesure les **distances** à l'environnement → une carte de **profondeur**. Trois grandes familles :

📍 Fixe

un faisceau orienté, distance sur un seul axe.

🔁 Rotatif 360° (2D)

mono-faisceau qui balaie un plan — le **capteur du cours**.

LiDAR — Light Detection and Ranging

Un **laser** mesure les **distances** à l'environnement → une carte de **profondeur**. Trois grandes familles :

📍 Fixe

un faisceau orienté, distance sur un seul axe.

🔁 Rotatif 360° (2D)

mono-faisceau qui balaie un plan — le **capteur du cours**.

🌐 Multi-beam (3D)

plusieurs faisceaux empilés → perception volumique.

LiDAR — Light Detection and Ranging

Un **laser** mesure les **distances** à l'environnement → une carte de **profondeur**. Trois grandes familles :

📍 Fixe

un faisceau orienté, distance sur un seul axe.

🔁 Rotatif 360° (2D)

mono-faisceau qui balaie un plan — le **capteur du cours**.

🌐 Multi-beam (3D)

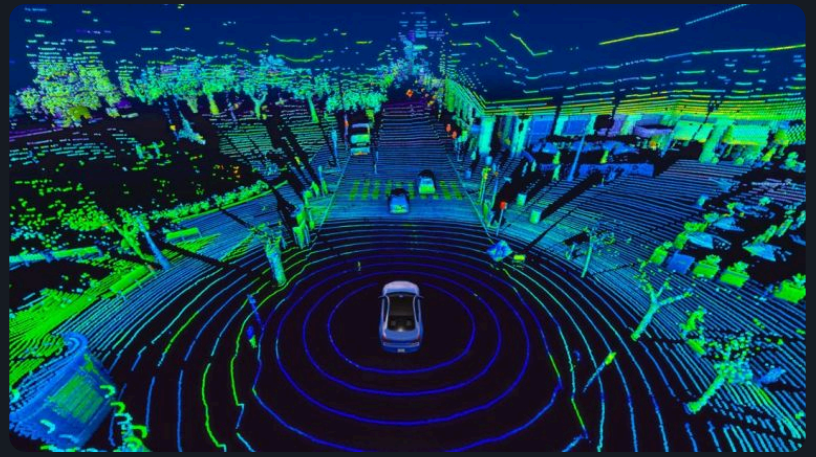
plusieurs faisceaux empilés → perception volumique.

Le capteur clé du cours

Le LeKiwi embarque un **LiDAR 360° 2D** (`/scan`) — détection d'obstacles, suivi de murs, et surtout **cartographie (SLAM)**.

ROS 2 — Bootcamp — Jour 2 Navigation

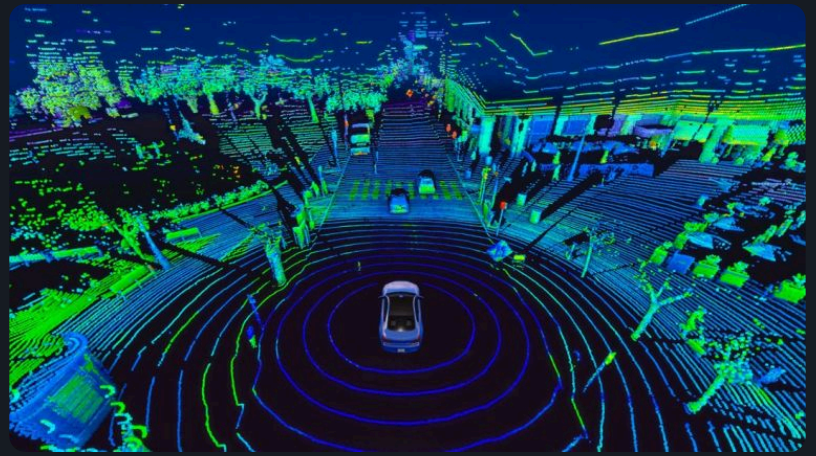
Superpose plusieurs **faisceaux** verticaux et horizontaux → une **perception 3D dense**.



ROS 2 — Bootcamp — Jour 2 Navigation

Superpose plusieurs **faisceaux** verticaux et horizontaux → une **perception 3D dense**.

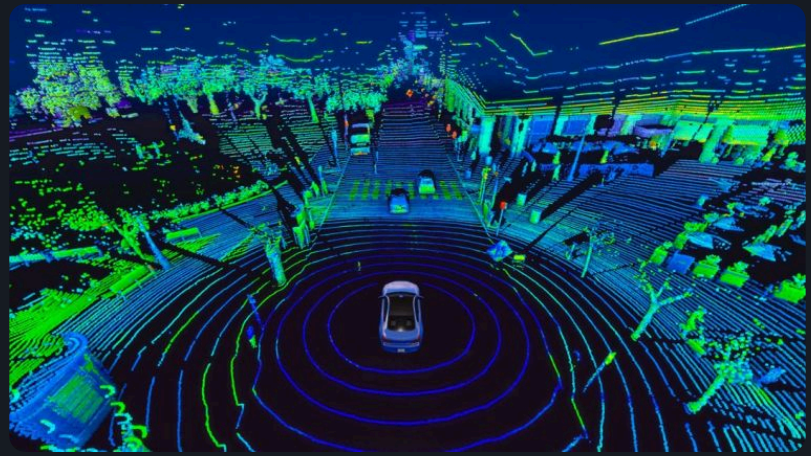
- très précis pour l'**évitement d'obstacles 3D** ;



ROS 2 — Bootcamp — Jour 2 Navigation

Superpose plusieurs **faisceaux** verticaux et horizontaux → une **perception 3D dense**.

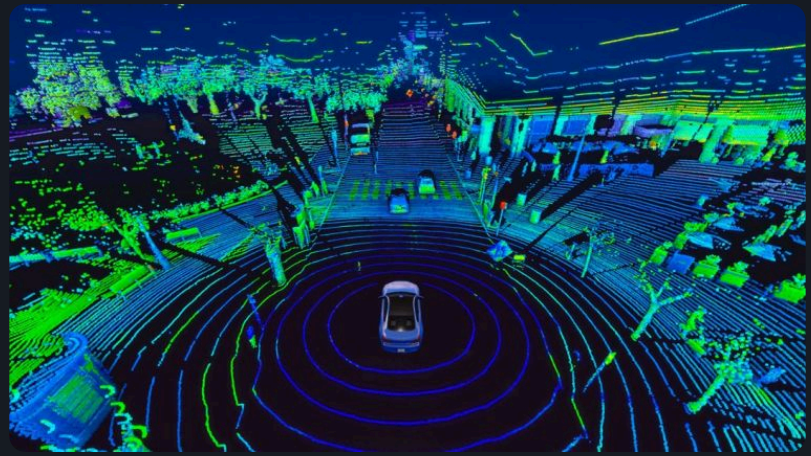
- très précis pour l'**évitement d'obstacles 3D** ;
- compréhension fine de la **scène** autour du robot ;



ROS 2 — Bootcamp — Jour 2 Navigation

Superpose plusieurs **faisceaux** verticaux et horizontaux → une **perception 3D dense**.

- très précis pour l'**évitement d'obstacles 3D** ;
- compréhension fine de la **scène** autour du robot ;
- au prix d'un volume de données bien plus lourd.



PARTIE 03 · CARTOGRAPHIER & SE LOCALISER

SLAM & AMCL

Construire la carte… puis s’y retrouver.

Qu’est-ce que le SLAM ?

SLAM = *Simultaneous Localization And Mapping* — un problème de **l’œuf et la poule** : pour se localiser il faut une carte, pour cartographier il faut savoir où l’on est. Le SLAM résout **les deux à la fois**.

Qu’est-ce que le SLAM ?

SLAM = *Simultaneous Localization And Mapping* — un problème de **l’œuf et la poule** : pour se localiser il faut une carte, pour cartographier il faut savoir où l’on est. Le SLAM résout **les deux à la fois**.

🎯 Le problème

L'odométrie **dérive** : sans repère absolu, l'erreur s'accumule au fil du trajet.

Qu’est-ce que le SLAM ?

SLAM = *Simultaneous Localization And Mapping* — un problème de **l’œuf et la poule** : pour se localiser il faut une carte, pour cartographier il faut savoir où l’on est. Le SLAM résout **les deux à la fois**.

🎯 Le problème

L'odométrie **dérive** : sans repère absolu, l'erreur s'accumule au fil du trajet.

🧩 L’idée

Recaler en continu les **scans laser** sur la carte en construction pour estimer la vraie pose.

Qu’est-ce que le SLAM ?

SLAM = *Simultaneous Localization And Mapping* — un problème de **l’œuf et la poule** : pour se localiser il faut une carte, pour cartographier il faut savoir où l’on est. Le SLAM résout **les deux à la fois**.

🎯 Le problème

L'odométrie **dérive** : sans repère absolu, l'erreur s'accumule au fil du trajet.

🧩 L’idée

Recaler en continu les **scans laser** sur la carte en construction pour estimer la vraie pose.

🔗 Le résultat

Une carte `/map` **et** la transformation `map → odom` qui corrige la dérive.

Qu’est-ce que le SLAM ?

SLAM = *Simultaneous Localization And Mapping* — un problème de **l’œuf et la poule** : pour se localiser il faut une carte, pour cartographier il faut savoir où l’on est. Le SLAM résout **les deux à la fois**.

🎯 Le problème

L'odométrie **dérive** : sans repère absolu, l'erreur s'accumule au fil du trajet.

🧩 L’idée

Recaler en continu les **scans laser** sur la carte en construction pour estimer la vraie pose.

🔗 Le résultat

Une carte `/map` **et** la transformation `map → odom` qui corrige la dérive.

> `slam_toolbox` complète enfin la chaîne `map → odom → base_footprint`.

Comment le SLAM se recale

L’odométrie seule **dérive** : le SLAM la recale en continu sur ce que voit le LiDAR.

Comment le SLAM se recale

L’odométrie seule **dérive** : le SLAM la recale en continu sur ce que voit le LiDAR.

🌀 Dérive odométrique

Les roues **glissent**, les encodeurs **arrondissent** : l'erreur s'accumule à chaque pas.

Comment le SLAM se recale

L'odométrie seule **dérive** : le SLAM la recale en continu sur ce que voit le LiDAR.

🌀 Dérive odométrique

Les roues **glissent**, les encodeurs **arrondissent** : l'erreur s'accumule à chaque pas.

🧩 Scan matching

Aligner chaque `/scan` sur la carte déjà construite → le **déplacement réel**, la dérive corrigée.

Comment le SLAM se recale

L'odométrie seule **dérive** : le SLAM la recale en continu sur ce que voit le LiDAR.

🌀 Dérive odométrique

Les roues **glissent**, les encodeurs **arrondissent** : l'erreur s'accumule à chaque pas.

🧩 Scan matching

Aligner chaque `/scan` sur la carte déjà construite → le **déplacement réel**, la dérive corrigée.

🔁 Fermeture de boucle

Un **lieu reconnu** ferme la boucle → l'erreur est **redistribuée** sur tout le trajet.

Comment le SLAM se recale

L'odométrie seule **dérive** : le SLAM la recale en continu sur ce que voit le LiDAR.

🌀 Dérive odométrique

Les roues **glissent**, les encodeurs **arrondissent** : l'erreur s'accumule à chaque pas.

🧩 Scan matching

Aligner chaque `/scan` sur la carte déjà construite → le **déplacement réel**, la dérive corrigée.

🔁 Fermeture de boucle

Un **lieu reconnu** ferme la boucle → l'erreur est **redistribuée** sur tout le trajet.

> `slam_toolbox` publie ainsi `map → odom`, la transformation qui **corrige la dérive** en recalant les scans.

AMCL — l'algorithme

Une fois la **carte connue**, plus besoin de la reconstruire : on s'y **localise** avec **AMCL** (*Adaptive Monte-Carlo Localization*).

AMCL — l’algorithme

Une fois la **carte connue**, plus besoin de la reconstruire : on s’y **localise** avec **AMCL** (*Adaptive Monte-Carlo Localization*).

- repose sur un **filtre à particules** : des centaines d’**hypothèses** de pose réparties sur la carte ;

AMCL — l'algorithme

Une fois la **carte connue**, plus besoin de la reconstruire : on s'y **localise** avec **AMCL** (*Adaptive Monte-Carlo Localization*).

- repose sur un **filtre à particules** : des centaines d'**hypothèses** de pose réparties sur la carte ;
- chaque scan **renforce** les bonnes hypothèses, **élimine** les mauvaises ;

AMCL — l’algorithme

Une fois la **carte connue**, plus besoin de la reconstruire : on s’y **localise** avec **AMCL** (*Adaptive Monte-Carlo Localization*).

- repose sur un **filtre à particules** : des centaines d’**hypothèses** de pose réparties sur la carte ;
- chaque scan **renforce** les bonnes hypothèses, **élimine** les mauvaises ;
- combine **LiDAR** + **odométrie** + **IMU**.

AMCL — l'algorithme

Une fois la **carte connue**, plus besoin de la reconstruire : on s'y **localise** avec **AMCL** (*Adaptive Monte-Carlo Localization*).

- repose sur un **filtre à particules** : des centaines d'**hypothèses** de pose réparties sur la carte ;
- chaque scan **renforce** les bonnes hypothèses, **élimine** les mauvaises ;
- combine **LiDAR** + **odométrie** + **IMU**.

> 🧭 **SLAM ≠ AMCL**
>
> **SLAM** *construit* la carte tout en se localisant (aucune carte au départ, il modifie `/map`). **AMCL** *se localise seulement* dans une carte **déjà connue** — il ne touche jamais à la carte.

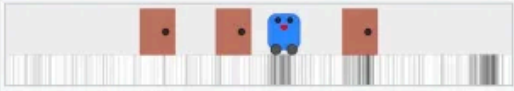
$t = 0$

The algorithm initializes with a uniform distribution of particles. The robot considers itself equally likely to be at any point in space along the corridor, even though it is physically at the first door.

Sensor update: the robot detects **a door**. It assigns a weight to each of the particles. The particles which are likely to give this sensor reading receive a higher weight.

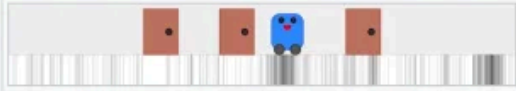
Resampling: the robot generates a set of new particles, with most of them generated around the previous particles with more weight. It now believes it is at one of the three doors.

$t = 1$



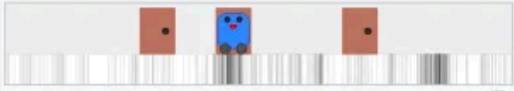
Motion update: the robot moves some distance to the right. All particles also move right, and some noise is applied. The robot is physically between the second and third doors.

Sensor update: the robot detects **no door**. It assigns a weight to each of the particles. The particles likely to give this sensor reading receive a higher weight.



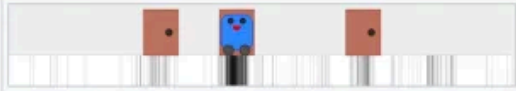
Resampling: the robot generates a set of new particles, with most of them generated around the previous particles with more weight. It now believes it is at one of two locations.

$t = 2$

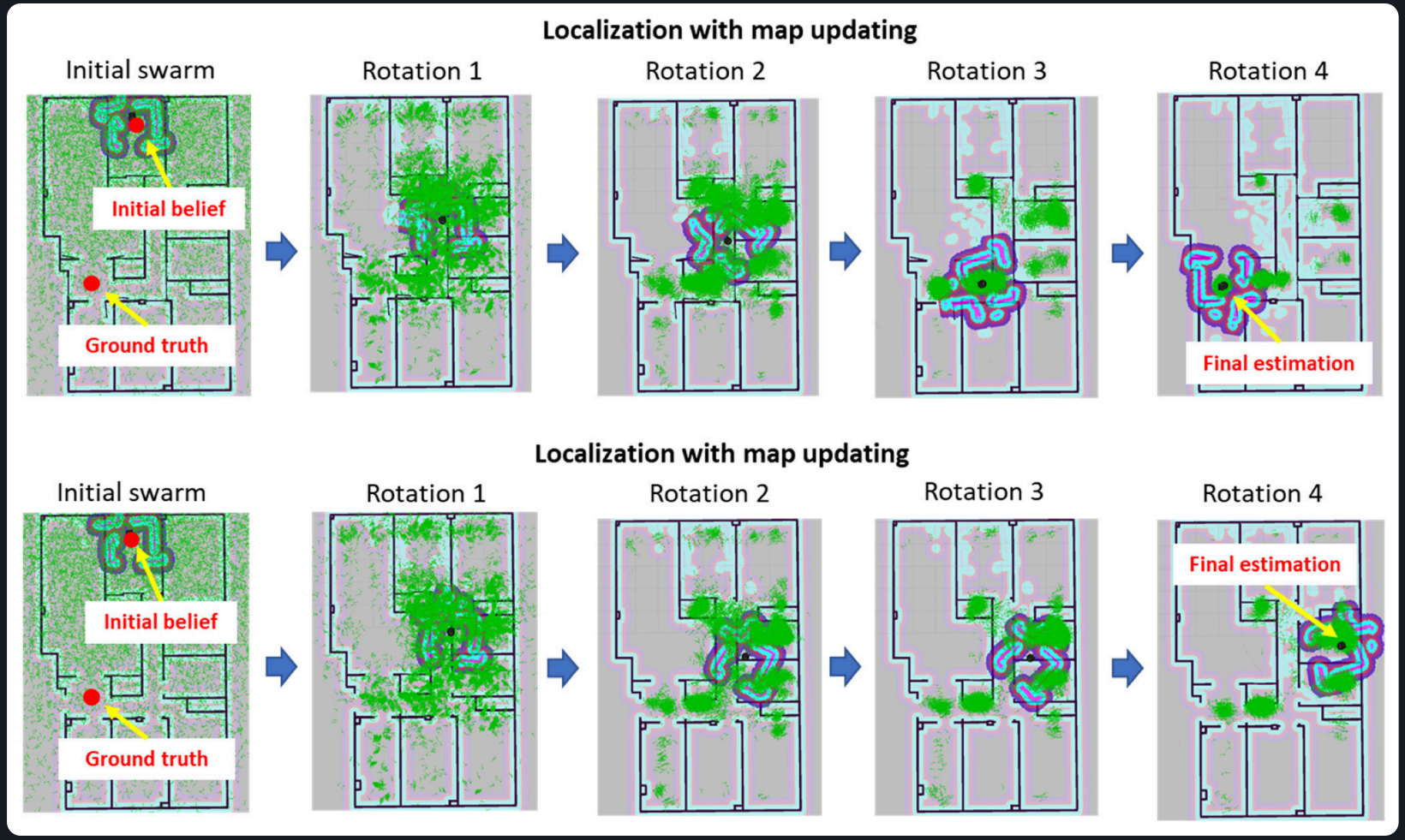


Motion update: the robot moves some distance to the left. All particles also move left, and some noise is applied. The robot is physically at the second door.

Sensor update: the robot detects **a door**. It assigns a weight to each of the particles. The particles likely to give this sensor reading receive a higher weight.



Resampling: the robot generates a set of new particles, with most of them generated around the previous particles with more weight. The robot has successfully localized itself.


Localization with map updating
Initial swarm
Rotation 1
Rotation 2
Rotation 3
Rotation 4
Initial belief
Ground truth
Final estimation
Localization with map updating
Initial swarm
Rotation 1
Rotation 2
Rotation 3
Rotation 4
Initial belief
Ground truth
Final estimation

PARTIE 04 · NAVIGUER AVEC NAV2

L’architecture de Nav2

Planifier globalement, réagir localement.

La stack Nav2

Une **boîte à outils complète** pour naviguer de façon autonome dans un environnement **inconnu ou non structuré** :

La stack Nav2

Une **boîte à outils complète** pour naviguer de façon autonome dans un environnement **inconnu ou non structuré** :

🗺️ Planifier

Chemin **global** (vers le but) et **local** (suivi temps réel).

La stack Nav2

Une **boîte à outils complète** pour naviguer de façon autonome dans un environnement **inconnu ou non structuré** :

🗺️ Planifier

Chemin **global** (vers le but) et **local** (suivi temps réel).

🛑 Éviter

Obstacles **statiques et dynamiques**.

La stack Nav2

Une **boîte à outils complète** pour naviguer de façon autonome dans un environnement **inconnu ou non structuré** :

🗺️ Planifier

Chemin **global** (vers le but) et **local** (suivi temps réel).

🛑 Éviter

Obstacles **statiques et dynamiques**.

📡 Percevoir

Fusionner LiDAR, odométrie, IMU.

La stack Nav2

Une **boîte à outils complète** pour naviguer de façon autonome dans un environnement **inconnu ou non structuré** :

🗺️ Planifier

Chemin **global** (vers le but) et **local** (suivi temps réel).

🛑 Éviter

Obstacles **statiques et dynamiques**.

📡 Percevoir

Fusionner LiDAR, odométrie, IMU.

🧭 Se localiser

SLAM ou AMCL.

La stack Nav2

Une **boîte à outils complète** pour naviguer de façon autonome dans un environnement **inconnu ou non structuré** :

🗺️ Planifier

Chemin **global** (vers le but) et **local** (suivi temps réel).

🛑 Éviter

Obstacles **statiques et dynamiques**.

📡 Percevoir

Fusionner LiDAR, odométrie, IMU.

🧭 Se localiser

SLAM ou AMCL.

⚙️ Exécuter

Mouvements avec **feedback**.

La stack Nav2

Une **boîte à outils complète** pour naviguer de façon autonome dans un environnement **inconnu ou non structuré** :

🗺️ Planifier

Chemin **global** (vers le but) et **local** (suivi temps réel).

🛑 Éviter

Obstacles **statiques et dynamiques**.

📡 Percevoir

Fusionner LiDAR, odométrie, IMU.

🧭 Se localiser

SLAM ou AMCL.

⚙️ Exécuter

Mouvements avec **feedback**.

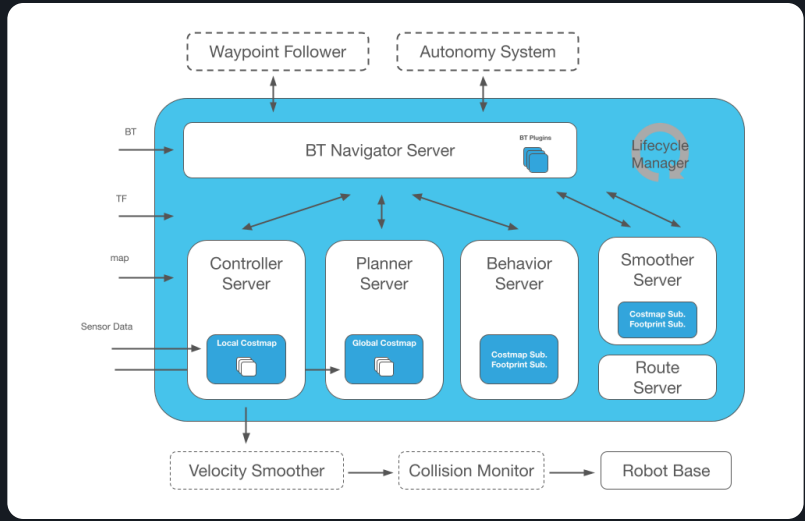
🌲 Orchestrer

Un **behavior tree** coordonne le tout.

ROS 2 — Bootcamp — Jour 2 Navigation

Nav2 n'est pas un nœud unique mais un **ensemble de serveurs** spécialisés, démarrés et supervisés par un `lifecycle_manager` :

- `bt_navigator`, `planner_server`, `controller_server` ;
- `behavior_server`, `velocity_smoother`, `collision_monitor`.

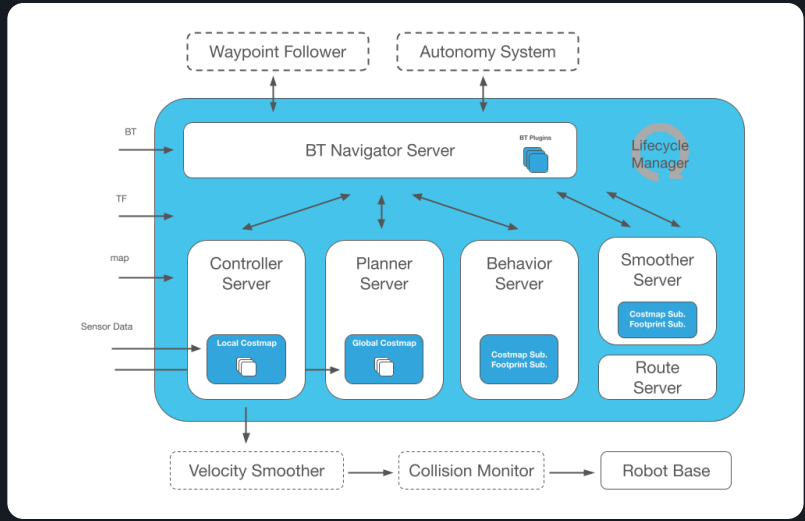


ROS 2 — Bootcamp — Jour 2 Navigation

Nav2 n'est pas un nœud unique mais un **ensemble de serveurs** spécialisés, démarrés et supervisés par un `lifecycle_manager` :

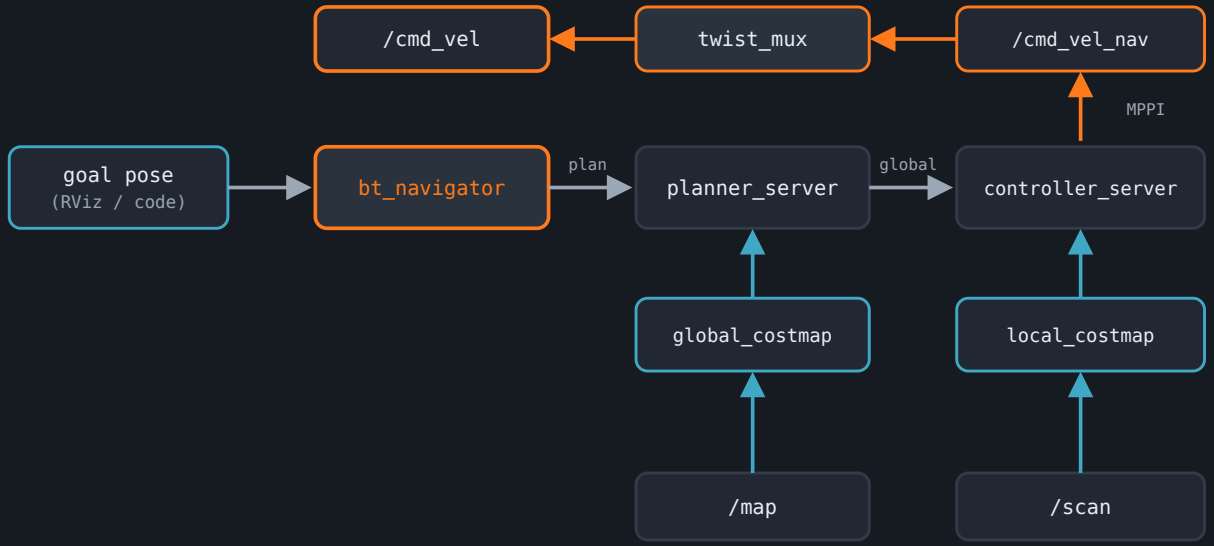
- `bt_navigator`, `planner_server`, `controller_server` ;
- `behavior_server`, `velocity_smoother`, `collision_monitor`.

> Chacun a un **cycle de vie** (configure → activate) géré proprement par le manager. On les détaille un par un.



Le pipeline, vu de haut

Du **but** à la **commande moteur** : qui calcule quoi, et avec quelles données.



- **goal pose** (RViz ou code) → `bt_navigator` **orchestre** la mission ;
- `planner_server` trace le **chemin global** — sur le `global_costmap` alimenté par `/map` ;

ROS 2 — Bootcamp — Jour 2
Navigation

`bt_navigator` — le **cœur** de la stack :

- orchestre la mission via un **behavior tree** ;
- reçoit une cible → planifie, suit, **récupère** ;
- guide le robot du début à la fin de la mission.

🌲 **bt_navigator**
orchestre la mission (behavior tree)

🗺️ **planner_server**
chemin global vers le but

🎮 **controller_server**
suivi local temps réel (MPPI)

🛟 **behavior_server**
comportements de récupération

ROS 2 — Bootcamp — Jour 2
Navigation

`planner_server` — la **planification globale** :

- entrées : **pose actuelle** + **objectif** ;
- calcule un **itinéraire optimal** (court, sûr, sans obstacle) ;
- planners : **Smac**, **NavFn** ;
- sort un **chemin global** à suivre.

🌲 **bt_navigator**
orchestre la mission (behavior tree)

🗺️ **planner_server**
chemin global vers le but

🎮 **controller_server**
suivi local temps réel (MPPI)

🛟 **behavior_server**
comportements de récupération

ROS 2 — Bootcamp — Jour 2 Navigation

`controller_server` — le **suivi local** :

- transforme le chemin global en **commandes de vitesse** ;
- **réagit en temps réel** aux obstacles et glissements ;
- garde le robot sur la voie dans un monde **changeant**.

🌲 **bt_navigator**
orchestre la mission (behavior tree)

🗺️ **planner_server**
chemin global vers le but

🎮 **controller_server**
suivi local temps réel (MPPI)

🛟 **behavior_server**
comportements de récupération

ROS 2 — Bootcamp — Jour 2 Navigation

`controller_server` — le **suivi local** :

- transforme le chemin global en **commandes de vitesse** ;
- **réagit en temps réel** aux obstacles et glissements ;
- garde le robot sur la voie dans un monde **changeant**.

> Détail du contrôleur **MPPI Omni** (base holonome) sur la slide suivante.

🌲 **bt_navigator**
orchestre la mission (behavior tree)

🗺️ **planner_server**
chemin global vers le but

🎮 **controller_server**
suivi local temps réel (MPPI)

🛟 **behavior_server**
comportements de récupération

ROS 2 — Bootcamp — Jour 2 Navigation

Tous les contrôleurs ne gèrent pas la **translation latérale**. Pour une base holonome, il faut le bon plugin :

Controller	Holonome ?
DWB	Oui, si `vy_samples > 0`
MPPI	**Oui,** `motion_model: Omni`
Reg. Pure Pursuit	Non (différentielle)

Le LeKiwi est **holonome → MPPI Omni** : meilleur tracking, sortie sur `/cmd_vel_nav`.

```yaml
controller_server:
  ros__parameters:
    controller_plugins: ["FollowPath"]
    FollowPath:
      plugin: "nav2_mppi_controller::MPPIController"
      motion_model: "Omni"   # ← base holonome
      time_steps: 56
      model_dt: 0.05
      vx_max: 0.5
      vy_max: 0.5            # ← non nul = holonome
      wz_max: 1.5
```

ROS 2 — Bootcamp — Jour 2
Navigation

`behavior_server` — réagit aux imprévus :

- robot **bloqué** ? obstacle **soudain** ?
- lance des **comportements de récupération** : reculer, tourner, attendre, réessayer.

🌲 **bt_navigator**
orchestre la mission (behavior tree)

🗺️ **planner_server**
chemin global vers le but

🎮 **controller_server**
suivi local temps réel (MPPI)

🛟 **behavior_server**
comportements de récupération

ROS 2 — Bootcamp — Jour 2 Navigation

`velocity_smoother` — lisse la trajectoire :

- **courbes plus douces**, vitesses réalistes ;
- déplacement **fluide**, moins d'à-coups.

🌲 bt_navigator

orchestre la mission (behavior tree)

🗺️ planner_server

chemin global vers le but

🎮 controller_server

suivi local temps réel (MPPI)

🛟 behavior_server

comportements de récupération

ROS 2 — Bootcamp — Jour 2
Navigation

`velocity_smoother` — lisse la trajectoire :

- **courbes plus douces**, vitesses réalistes ;
- déplacement **fluide**, moins d'à-coups.

> Avec `bt_navigator` et `behavior_server`, il rend la navigation **robuste** *et* **confortable**.

🌲 bt_navigator
orchestre la mission (behavior tree)

🗺️ planner_server
chemin global vers le but

🎮 controller_server
suivi local temps réel (MPPI)

🛟 behavior_server
comportements de récupération

Les costmaps

Nav2 raisonne sur **deux grilles de coût** complémentaires — il **planifie globalement** et **réagit localement** :

Les costmaps

Nav2 raisonne sur **deux grilles de coût** complémentaires — il **planifie globalement** et **réagit localement** :

🗺️ Global costmap

La carte statique `/map` + obstacles connus. Sert au **planner** pour le chemin global.

Les costmaps

Nav2 raisonne sur **deux grilles de coût** complémentaires — il **planifie globalement** et **réagit localement** :

🗺️ Global costmap

La carte statique `/map` + obstacles connus. Sert au **planner** pour le chemin global.

📡 Local costmap

Fenêtre glissante alimentée par le **LiDAR temps réel**. Sert au **controller** pour éviter les obstacles dynamiques.

Les costmaps

Nav2 raisonne sur **deux grilles de coût** complémentaires — il **planifie globalement** et **réagit localement** :

🗺️ Global costmap

La carte statique `/map` + obstacles connus. Sert au **planner** pour le chemin global.

📡 Local costmap

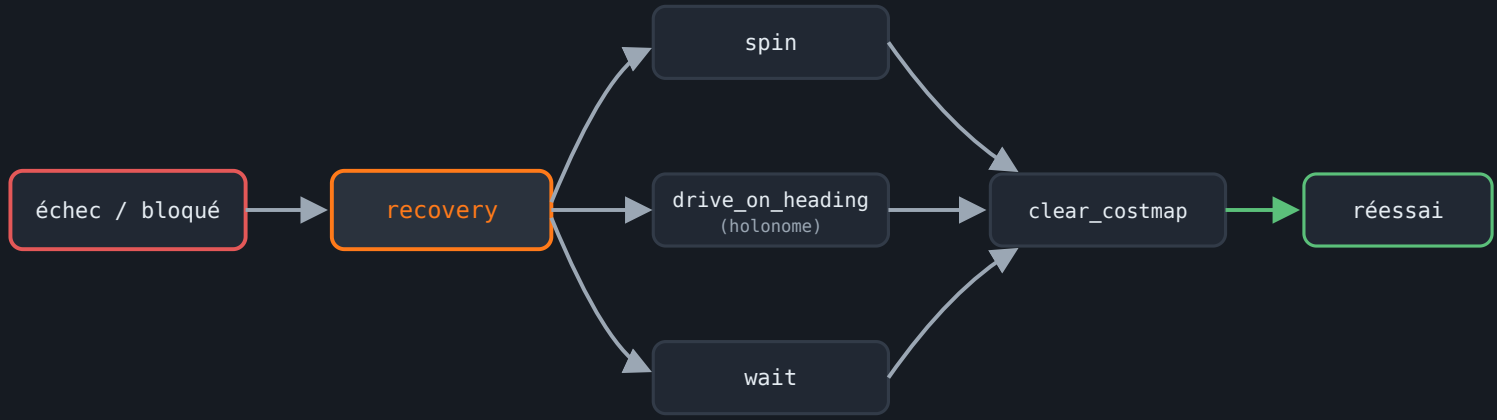
Fenêtre glissante alimentée par le **LiDAR temps réel**. Sert au **controller** pour éviter les obstacles dynamiques.

💡 **Inflation**

Une marge de coût est « gonflée » autour des obstacles (`inflation_radius`) pour garder le robot à distance des murs.

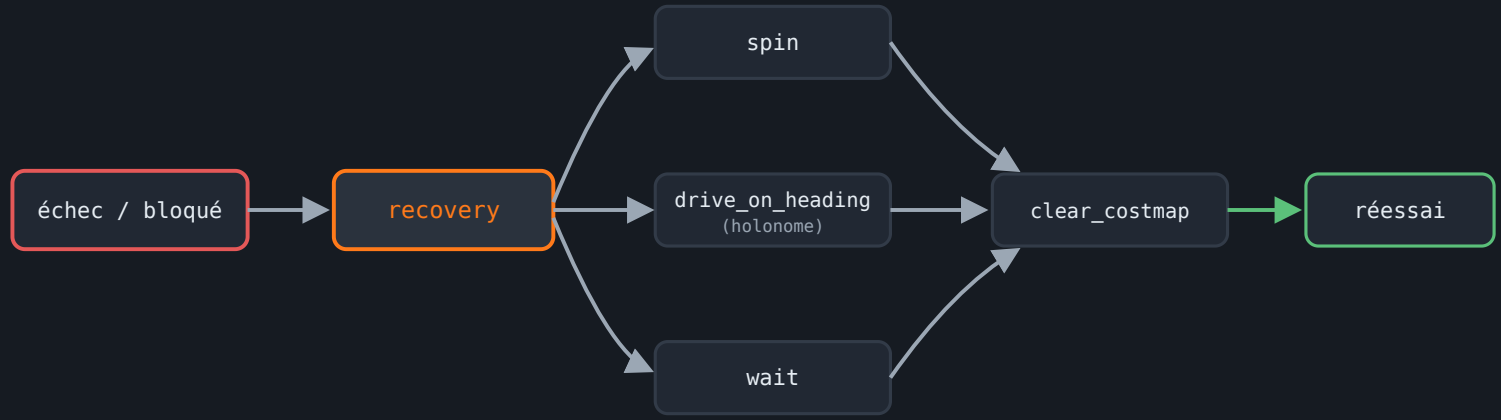
Recoveries

Quand le robot est bloqué, le behavior tree déclenche un **recovery** :



Recoveries

Quand le robot est bloqué, le behavior tree déclenche un **recovery** :



> Pour une base holonome, on retient `drive_on_heading` plutôt que `back_up` — manœuvres latérales possibles.

ROS 2 — BOOTCAMP

Questions ?

ROS 2 — Bootcamp · Perpignan 2026

ros2.etienne-schmitz.com