Université Mohamed Khider de Biskra
Faculté des Sciences et de la Technologie
Département de génie électrique et automatique

MÉMOIRE DE MASTER

Sciences et Technologies
Electronique
Electronique des Systémes Embarqués

Réf. : Entrez la référence du document

Présenté et soutenu par :
Hamzaoui Aya /Chamakhi Latifa

Le : mardi 3 juin 2025

Simulation dans ROS 2 et Gazebo d'un bras robotique

Jury :

Dr.	Khaled Bekhouche	MCA	Université Mohamed Khider	Rapporteur
Dr.	Okba benelmir	MCB	Université Mohamed Khider	Président
Pr.	Rahmani nacer eddine	Pr	Université Mohamed Khider	Examinateur

Année universitaire : 2025

Université Mohamed Khider de Biskra
Faculté des Sciences et de la Technologie
Département de génie électrique et automatique

MÉMOIRE DE MASTER

Sciences et Technologies
Electronique
Electronique des Systémes Embarqués

Réf. : Entrez la référence du document

Simulation dans ROS 2 et Gazebo d'un bras robotique

Le : mardi 3 juin 2025

Présenté par :Avis favorable de l'encadreur :

Signature Avis favorable du Président du Jury

Cachet et signature

REMERCIEMENTS

Louange à Dieu, par Sa grâce les bonnes œuvres s'accomplissent, et grâce à Son soutien, ce modeste travail a pu être mené à bien.

Je tiens à exprimer ma sincère gratitude et ma profonde reconnaissance à toutes les personnes qui m'ont soutenu(e) tout au long de mon parcours universitaire, en particulier durant l'élaboration de ce mémoire.

Je remercie tout spécialement Monsieur Khaled Bekhouch, mon encadrant, pour ses conseils précieux, ses orientations judicieuses, sa disponibilité constante et son accompagnement tout au long de ce travail. Je lui adresse mes plus vifs remerciements et ma haute considération.

Je remercie également les membres du jury pour avoir accepté d'évaluer ce mémoire, ainsi que l'ensemble des enseignants de l'Université Mohamed Khider – Faculté des Sciences et Technologies, pour la qualité de leur enseignement, leur engagement et leur dévouement tout au long de ma formation.

Mes remerciements vont également à toutes les personnes qui, de près ou de loin, ont contribué à la réalisation de ce travail.

Enfin, je tiens à exprimer ma reconnaissance à ma famille et à mes ami(e)s pour leur soutien moral, leur patience et leurs encouragements constants tout au long de ce parcours.

Merci à vous tous, du fond du cœur.

Nom de l'étudiant(e) :

-Chamakhi Latifa

-Hamzaoui Aya

DÉDICASE

Louange à Allah au commencement et à la fin, car nul chemin ne s'achève, nul effort ne s'accomplit, nul but n'est atteint sans Sa faveur.

À celui qui a dit : "Je suis fait pour cela, et j'y parviendrai, même si cela me résiste. Malgré tout, je l'ai réalisé."

Me voici aujourd'hui couronnant les derniers instants de ce parcours, après des années de labeur et de difficultés sur le chemin du savoir, des années remplies de rêves nocturnes qui sont aujourd'hui devenus réalité. Me voici, debout à l'aube de mon obtention du diplôme de MASTER, récoltant les fruits de mes efforts, et levant fièrement ma casquette. Ô Seigneur, à Toi la louange si Tu es satisfait, et à Toi la louange après la satisfaction, car c'est Toi qui m'as permis de réussir et de réaliser ce rêve.

Avec tout mon amour et toute ma fierté, je dédie le fruit de ma réussite et de mon diplôme :

À moi-même, ambitieuse, qui ai commencé avec un rêve et terminé avec un succès.

À celui dont je porte le nom avec fierté, à celui qui a écarté les épines de mon chemin pour l'éclairer de lumière, à celui qui fut mon pilier après Dieu, à celui qui m'a appris que la volonté crée des miracles, à celui qui a semé dans mon cœur l'amour du savoir et qui m'a accompagnée à chaque pas : mon cher père.

À celle dont Dieu m'a recommandé de prendre soin, à celle sous les pieds de laquelle se trouve le paradis, à celle qui m'a nourrie comme l'oiseau nourrit ses petits, à celle dont les prières m'ont accompagnée vers la réussite, à celle qui apaise mes blessures, à mon immense patrie sans frontières : ma chère mère.

Et à ces étoiles qui illuminent mon chemin sans jamais s'éteindre, à ceux qui m'inspirent la réussite, à ceux qui sont ma force, et les joyaux de mes jours, qui ont été les partenaires de chaque pas : mes chers frères et sœurs

Aya hamzaoui

DÉDÉCASE

À ceux qui nous avons indiqué la bonne voie en nous rappelant que lavolonté fait toujours les grands hommes et qui ont attendus avec patienceles fruits de leur bonne éducation...

« Mes Parents »

À ceux qui nous avons été toujours la garantie d'une existence paisible etd'un avenir radieux…

« Notre belle famille »

À Ceux qui nous avons soutenus, encouragés, appréciés nos efforts etcrées le milieu favorable, l'ambiance joyeuse et l'atmosphère joviale pour procurer ce travail.

« Mes chers amis »

À toutes ces personnes que nous sentons redoutables de leur dédier ce modeste travail avec nos vifs remerciements et expressions respectueuses de notre profonde gratitud

Chamakhi Latifa

Résumé :

Ce mémoire présente en détail les aspects fondamentaux de la création d'un environnement de simulation réaliste pour un bras robotique. L'étude se concentre sur trois axes principaux :

ROS 2 est présenté comme cadre central, avec une analyse de son architecture, de ses principales fonctionnalités et de la manière dont il permet de gérer et de coordonner les composants du robot simulé.

la conception et la modélisation du bras robotique sont approfondies, notamment sa représentation au format URDF/XACRO et son intégration dans l'environnement de simulation Gazebo, en mettant l'accent sur la dynamique du mouvement et la modélisation physique.

l'étude analyse les capteurs et actionneurs couramment utilisés dans les bras robotiques, ainsi que leur simulation précise dans Gazebo. Elle détaille également les mécanismes de contrôle de ces composants via ROS 2, en particulier à travers l'utilisation du paquet ros2_control pour la commande des articulations du bras.

L'étude conclut que l'intégration de ROS 2 et de Gazebo offre une plateforme puissante et flexible pour développer, tester et visualiser le comportement des bras robotiques de manière efficace avant toute mise en œuvre physique. Les résultats mettent en évidence la capacité de ces outils open source à simuler des interactions complexes entre le bras robotique et son environnement, contribuant ainsi à accélérer le cycle de développement en robotique et à réduire les coûts.

الملخص :

تتناول هذه المذكرة بالتفصيل الجوانب الأساسية لإنشاء بيئة محاكاة واقعية لذراع روبوتية .تم التركيز على ثلاثة محاور رئيسية ، ROS 2 كإطار عمل مركزي، حيث تم استعراض بنيته المعمارية، ميزاته الرئيسية، وكيفية استخدامه لإدارة وتنسيق مكونات الروبوت المحاكى .

تم التعمق في تصميم و نمذجة الذراع الروبوتية، بما في ذلك تمثيلها في تنسيق URDF/XACRO وتضمينها في بيئة Gazebo، مع التركيز على ديناميكيات الحركة والفيزياء .تم تحليل المستشعرات والمشغلات (Actuators) المستخدمة عادة في الأذرع الروبوتية، وكيفية محاكاتها بدقة داخل Gazebo ، بالإضافة إلى توضيح آليات التحكم في هذه المكونات عبر ROS 2 ، بما في ذلك استخدام حزمة ros2_control للتحكم في مفاصل الذراع .

خلصت الدراسة إلى أن التكامل بين ROS 2 و Gazebo يوف رمنصة قوية ومرنة لـ تطوير واختبار و تصوير سلوك الأذرع الروبوتية بكفاءة عالية قبل أي تطبيق فيزيائي .

تبرز النتائج قدرة هذه الأدوات مفتوحة المصدر على محاكاة التفاعلات المعقدة بين الذراع الروبوتية و بيئتها، مما يساهم بشكل كبير في تسريع دورة تطوير الروبوتات و تقليل التكاليف.

LISTE DES FIGURES

TABLE DE MATIÈRES

Introduction générale :

Dans le cadre de la quatrième révolution industrielle, les robots sont devenus un élément essentiel dans le développement des systèmes intelligents et de l'automatisation industrielle. Des rapports récents montrent que l'adoption des robots dans les domaines industriel, médical et éducatif connaît une croissance rapide dépassant 30 % par an, ce qui souligne l'importance des recherches contribuant au développement de cette technologie vitale.

Parmi les outils les plus importants pour le développement des systèmes robotiques figure la simulation, qui permet de tester de nouvelles technologies sans risquer d'endommager les composants physiques ou d'engendrer des coûts élevés.

Dans ce contexte, ROS 2 (Robot Operating System 2) et Gazebo se distinguent comme des plateformes open source puissantes et complètes pour la simulation réaliste et efficace des robots, notamment des bras robotiques.

La problématique de cette recherche réside dans la difficulté de simuler avec précision un bras robotique dans un environnement basé sur ROS 2 et Gazebo, notamment en ce qui concerne l'intégration des capteurs et des actionneurs, ainsi que l'interaction fluide entre le logiciel et le modèle physique. Ce manque de précision et de synchronisation représente un défi pour les chercheurs et les développeurs souhaitant concevoir des systèmes fiables et applicables en conditions réelles.

Cette recherche vise à répondre aux questions suivantes :

Comment exploiter l'environnement ROS 2 pour simuler et contrôler efficacement un bras robotique ?

Quelle est la structure mécanique du bras robotique étudié ?

Quels sont les types de capteurs et d'actionneurs utilisés dans ces systèmes ? Et comment sont-ils intégrés au niveau logiciel et physique ?

L'importance de cette étude réside dans sa capacité à offrir une approche méthodologique combinant l'aspect logiciel (ROS 2), l'aspect mécanique (structure du bras robotique) et l'aspect technique (capteurs et actionneurs), ce qui en fait une référence utile pour les chercheurs et ingénieurs souhaitant développer ou améliorer des systèmes robotiques similaires.

La portée de ce travail se limite à la simulation d'un bras robotique à l'aide de ROS 2 et Gazebo, en se concentrant sur sa structure et ses mouvements, ainsi que sur les capteurs et actionneurs utilisés. Ce mémoire n'abordera pas la conception des circuits de commande électroniques ni l'intégration avec les systèmes de vision artificielle.

Ce mémoire est structuré en trois chapitres principaux :

Chapitre 1 : une vue d'ensemble de ROS 2, ses caractéristiques et ses avantages par rapport à la version précédente, ainsi que son rôle dans le contrôle des robots.

Chapitre 2 : une étude de la structure du bras robotique, ses composants et ses degrés de liberté.

Chapitre 3 : une analyse détaillée des capteurs et actionneurs utilisés dans le bras robotique, et leur intégration dans l'environnement de simulation.

Chapitre I
ROS2

I.1.Introduction :

ROS, ou Robot Operative System, est un ensemble d'outils et de techniques open source conçus pour faciliter le développement d'applications robotiques. Sa fonction principale est de fournir aux développeurs diverses bibliothèques de logiciels et outils nécessaires à la création de ces applications.

Initialement créé en 2007, ROS est depuis devenu le premier système de création d'applications robotiques dans le monde entier. Son caractère open source et sa communauté qualifiée ont contribué de manière significative à son progrès au fil des années. En conséquence, ROS a connu un développement majeur ces derniers temps.

La première itération du système d'exploitation du robot était ROS1. Il possédait un grand nombre de caractéristiques qui lui ont permis de gagner en popularité dans l'industrie de la robotique, mais il présentait également certaines limites. L'un des principaux problèmes était son manque de capacités en temps réel, ce qui rendait son utilisation difficile dans des applications critiques pour la sécurité. De plus, le système de communication de ROS1 était basé sur un nœud maître centralisé, ce qui le rendait vulnérable aux points de défaillance uniques.

En réponse à ces défis, ROS2 a été publié en 2015, qui a résolu bon nombre de ces problèmes tout en conservant les points forts de ROS1. ROS2 a introduit un nouveau protocole de communication, DDS (Data Distribution Service), qui est plus robuste et fiable que le système centralisé de ROS1. Ce nouveau protocole permet également une communication en temps réel et prend en charge les systèmes distribués, ce qui rend ROS2 adapté aux applications robotiques plus grandes et plus complexes.

ROS2 a également ajouté plusieurs nouvelles fonctionnalités, telles qu'une meilleure prise en charge des systèmes multirobots, des outils de débogage améliorés et une modularité accrue. De plus, ROS2 a conservé les fonctionnalités principales qui ont rendu ROS1 si populaire, telles que sa nature open source et sa vaste communauté de développeurs.

Actuellement, il existe 8 distributions ROS différentes, mais seules deux d'entre elles sont actuellement prises en charge, ROS2 Foxy et ROS2 Humble. Ce projet est développé en utilisant la distribution Humble, ce qui signifie qu'il aura des fonctionnalités ou des compatibilités légèrement différentes avec d'autres composants logiciels ou matériels que Foxy, mais les principales procédures et idées seront similaires. La raison pour laquelle nous avons choisi Humble plutôt que le reste des distributions est qu'il s'agit de la dernière version

de ROS2 à support à long terme, ce qui signifie qu'elle sera idéalement la prochaine distribution ROS la plus utilisée.**[1]**

I..2.Architecture ROS2 :

1 . Nœuds (Nodes) :

Les nœuds sont les unités de base dans ROS2. Chaque nœud représente une tâche ou une fonction spécifique, par exemple : lire un capteur, commander un moteur, traiter des images, etc.Plusieurs nœuds peuvent s'exécuter en parallèle.Les nœuds communiquent entre eux via des topics, services ou actions.

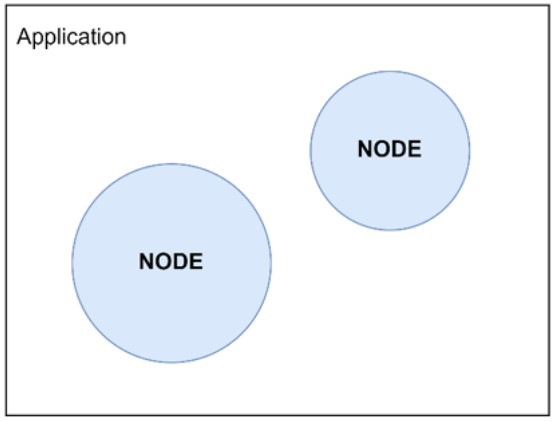


Figure 1. Représentation des nœuds dans une application.

2 .Topics (Sujets) :

Les topics permettent la communication asynchrone entre nœuds à l'aide d'un modèle éditeur/abonné (publisher/subscriber).

- Un nœud publie des messages sur un topic, d'autres nœuds peuvent s'y abonner pour recevoir les messages.
- Exemple : Une caméra publie des images, et un autre nœud s'abonne pour les traiter.

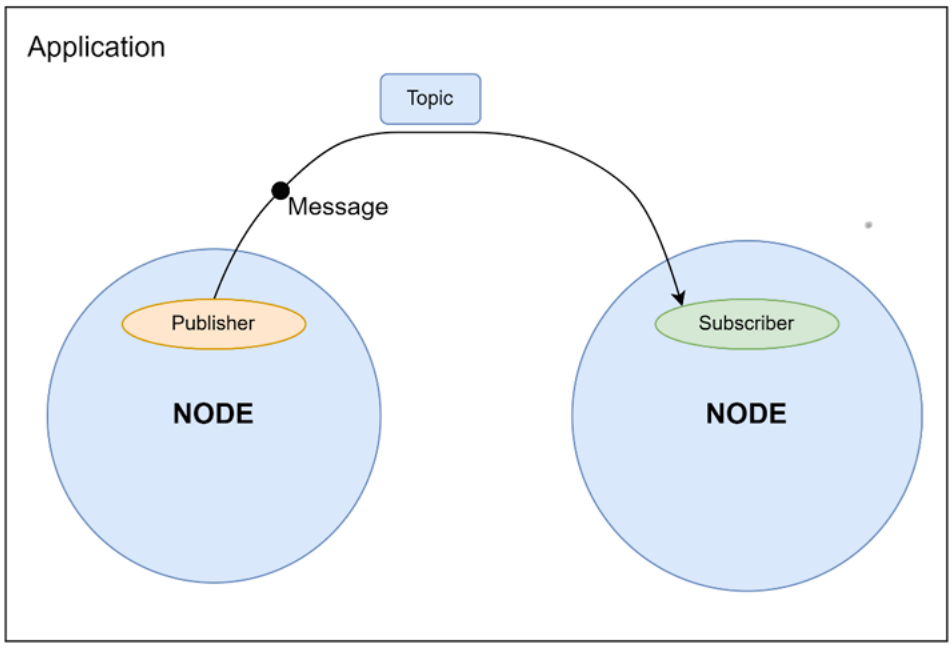


Figure 2. Communication par le biais d'un sujet entre deux nœuds.

3. Services (Services) :

Les services permettent une communication synchrone, type requête/réponse.

- Un client envoie une requête, et un serveur de service y répond.
- Exemple : demander au robot d'aller à une position précise.

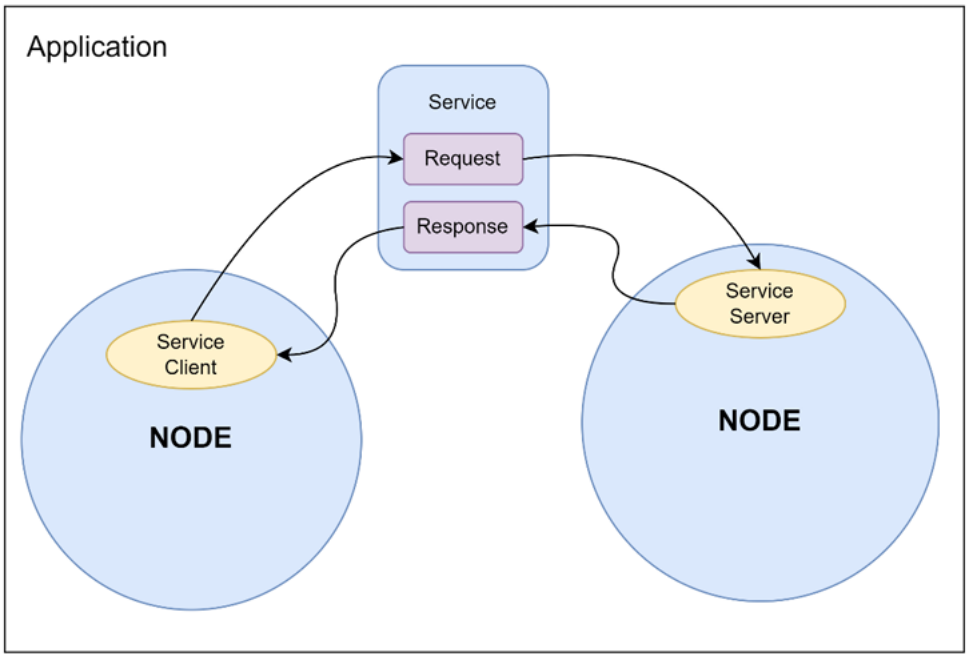


Figure 3. Communication par le biais d'un service entre deux nœuds..

4 . Actions :

Les actions sont utilisées pour des tâches longues ou continues, avec possibilité de feedback pendant l'exécution.

- Exemple : demander au robot de se déplacer vers un point tout en recevant son avancement.

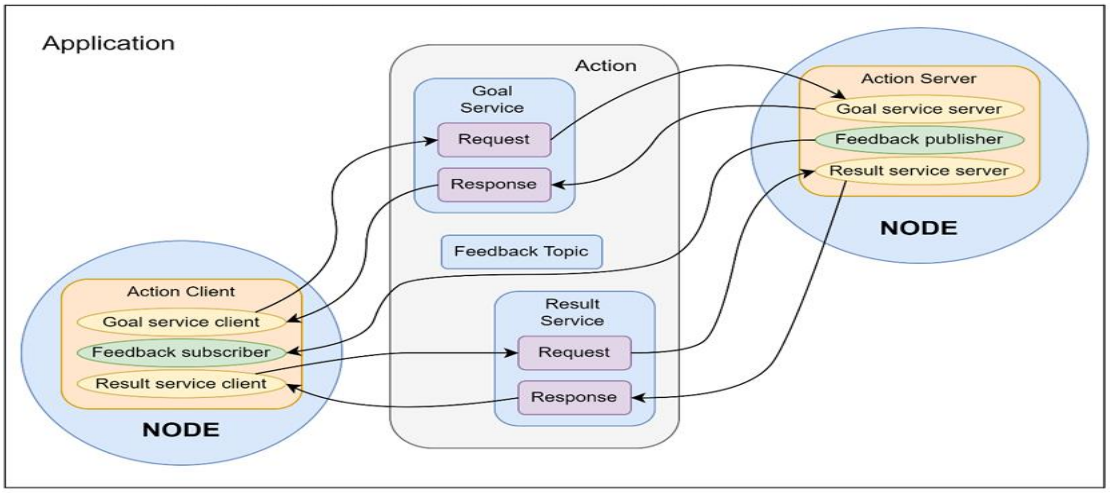


Figure 4. . Communication par le biais d'une action entre deux nœuds

5. Messages et Interfaces :

Les messages sont les structures de données échangées entre les nœuds (via topics, services ou actions).

Ils peuvent contenir des types simples (entiers, flottants) ou complexes (positions, images, etc.).

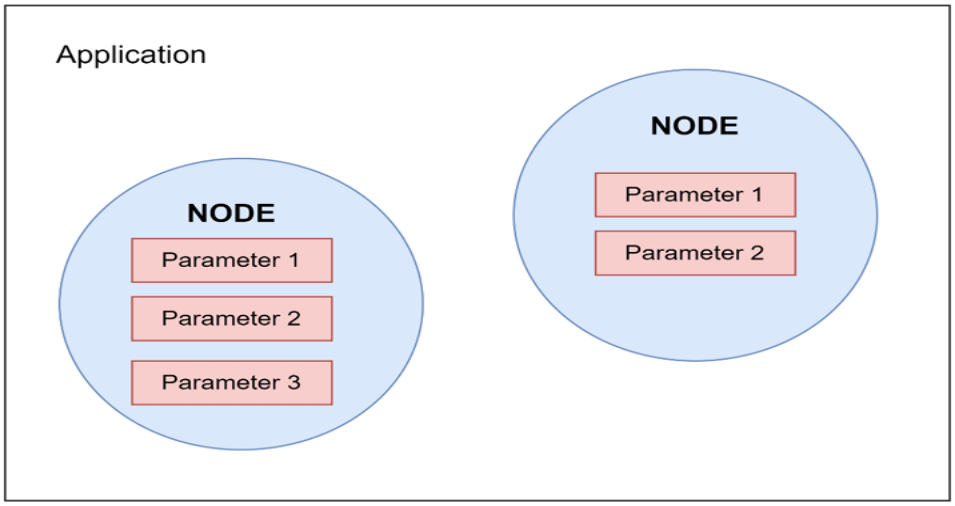


Figure 5. Paramètres tels que paramètres ou caractéristiques des nœuds.

6. Middleware DDS (Data Distribution Service):

ROS2 repose sur DDS pour la communication :

- Découverte automatique des nœuds.
- Qualité de service configurable.
- Échange de données rapide, sécurisé et fiable.
- Permet une architecture distribuée sans configuration manuelle.

7. Système de lancement (Launch System) :

Les fichiers de lancement permettent de démarrer plusieurs nœuds et de configurer leurs paramètres.

- Ces fichiers sont écrits en Python ou XML.

8. RMW (ROS Middleware Interface) :

RMW est une couche d'abstraction qui permet à ROS2 d'être compatible avec différents fournisseurs DDS (Fast DDS, Cyclone DDS, RTI Connext, etc.).

9. Contrôle et Planification (MoveIt2, Nav2) :

MoveIt2 – Planification du Mouvement :

MoveIt2 est un framework utilisé pour la planification du mouvement des bras robotiques. Il permet de calculer un trajet optimal pour atteindre une position cible, tout en évitant les collisions avec l'environnement.

Composants principaux de MoveIt2 :

1. Motion Planning (Planification de mouvement)
    - Utilise des bibliothèques comme OMPL pour générer des trajectoires entre un point de départ et une cible.
2. Vérification des collisions
    - Évite que le robot entre en collision avec lui-même ou avec l'environnement (scène 3D simulée).
3. Cinématique inverse (Inverse Kinematics - IK)
    - Transforme une position dans l'espace (x, y, z) en angles de rotation des articulations.

4. Exécution de trajectoire
    - Envoie les commandes nécessaires au contrôleur du robot pour effectuer les mouvements planifiés.
5. Interface MoveGroup (C++ / Python)
    - Interface de haut niveau pour contrôler facilement le bras robotique.

Cas d'utilisation :

- Manipulation d'objets (Pick and Place)
- Robots industriels collaboratifs
- Robotique chirurgicale, recherche, etc.

Nav2 – Navigation autonome :

Nav2 (Navigation 2) est un système de navigation pour les robots mobiles autonomes. Il permet au robot de se déplacer intelligemment d'un point A à un point B dans un environnement 2D.

Composants principaux de Nav2 :

1. SLAM (Simultaneous Localization and Mapping)
    - Le robot construit une carte de l'environnement tout en se localisant dessus.
2. Localisation (ex. AMCL)
    - Permet au robot de savoir où il se trouve sur une carte existante.
3. Global Planner (Planificateur global)
    - Génère un chemin optimal à travers la carte (algorithmes comme Dijkstra, A*).
4. Contrôleur Local (Local Planner)
    - Permet d'éviter les obstacles et suivre la trajectoire globale en temps réel.
5. Comportements de récupération (Recoverybehaviors)
    - Gèrent les situations bloquées (ex : faire marche arrière, tourner sur place).
6. Arbres de comportement (BehaviorTrees - BT Navigator)
    - Orchestrent les différentes actions du robot de manière modulaire et intelligente.

Cas d'utilisation :

- Robots de livraison ou d'assistance
- Robots industriels autonomes
- Surveillance, sécurité, hôpitaux, entrepôts, etc.

I.3 .Installation de ROS2 :

1. Installation de ROS 2 Humble sur Ubuntu 22.04 :

Étapes d'installation :

Configurer les sources du dépôt ROS 2 :

```
sudo apt update &&sudo apt install curl gnupglsb-release
```

```
sudo curl -sSL https://raw.githubusercontent.com/ros/rosdistro/master/ros.key | sudo tee /usr/share/keyrings/ros-archive-keyring.gpg> /dev/null
```

```
echo "deb [arch=$(dpkg --print-architecture) signed-by=/usr/share/keyrings/ros-archive-keyring.gpg] http://packages.ros.org/ros2/ubuntu $(lsb_release -cs) main" | \
```

```
sudo tee /etc/apt/sources.list.d/ros2.list > /dev/null
```

Mettre à jour l'index des paquets :

```
sudoapt update
```

Installer ROS 2 Humble (version Desktop complète) :

```
sudoaptinstall ros-humble-desktop
```

Configurer l'environnement (pour chaque nouveau terminal) :

```
echo "source /opt/ros/humble/setup.bash" >> ~/.bashrc
```

```
source ~/.bashrc
```

Installer les dépendances pour la création de packages ROS 2 :

```
sudoaptinstall -y python3-colcon-common-extensions python3-rosdep python3-argcomplete
```

Initialiser rosdep :

```
sudorosdep init
```

```
rosdep update
```

2. Synchronisation NTP avec chrony (recommandée pour ROS 2 multi-machines) :

Étapes recommandées :

Installer chrony :

sudoaptinstallchrony

Synchroniser une première fois avec un serveur NTP :

sudontpdate ntp.ubuntu.com

S'assurer que le service chrony fonctionne :

sudosystemctl enable chrony

sudosystemctl start chrony

Vérifier l'état de la synchronisation :

Chronyctracking **[2]**

I.4/Gazebo :

Gazebo est le simulateur historique sur ROS.La dernière version est Gazebo 11 est d'après http: //gazebosim.org/#features la fin de vie de Gazebo est prévue pour le 29 Janvier 2025. Le successeur de Gazebo est Ignition.

Gazebo est développé sous licence Apache 2.0.

Gazebo est un simulateur système.

Il permet d'interfacer les logiciels développés pour le robot avec le simulateur.

Il est possible d'étendre les capacités du simulateur grâce à un système de plugins.

Pour la suite on peut cloner le package suivant dans un workspacegazebo_ws :

```
mkdir -p $HOME/gazebo_ws/src
cd $HOME/gazebo_ws/src
git clone https://github.com/olivier-stasse/gazebo_ros_demos.git
```

Pour le construire :

```
cd $HOME/gazebo_ws/
colcon build --packages-select
```

SDF format

Le format SDF est formalisé ici : http://sdformat.org/, il permet de décrire les différents éléments nécessaires à la simulation : le modèle des éléments du monde, le robot et les modèles de contact à utiliser.

Un exemple extrêmement simple est donné ici

```
<?xml version="1.0" ?>
<sdf version="1.4">
  <world name="default">
    <include>
      <uri>model://ground_plane</uri>
    </include>

    <include>
      <uri>model://sun</uri>
    </include>

    <!-- reference to your plugin -->
    <plugin name="gazebo_tutorials" filename="libgazebo_tutorials.so"/>
  </world>
</sdf>
```

Ce fichier XML est un exemple très simple. On peut spécifier le format grâce à la ligne :

```
<sdf version="1.4">
```

On décrit le monde entre les balises world.

On peut ensuite mettre des modèles :

```
    <include>
      <uri>model://ground_plane</uri>
    </include>

    <include>
      <uri>model://sun</uri>
    </include>
```

Ici le sol avec ground_plane et le soleil comme lumière (sun).

Enfinonpeutspécifierunpluginnommégazebo_tutorialsinclusdanslabibliothèque libgazebo_tutorials.so grâce à :

```
    <plugin name="gazebo_tutorials" filename="libgazebo_tutorials.so"/>
```

Quand utiliser le format SDF ou le format URDF?

Lorsque l'on utilise ROS dans une application robotique une bonne pratique est de ne pas faire de fichier SDF spécifique pour son robot. En effet, gazebo est capable de lire un fichier URDF et d'en faire un fichier SDF. Les plugins de gazebo à utiliser peuvent être indiqués par des balises gazebo.

Celles-ci seront ignorées par les autres logiciels, MoveIt par exemple, qui ne les utilise pas. Afin de pouvoir contrôler le robot à la fois en simulation et en contrôle il est généralement conseillé d'utiliser ros-control.Ce projet offre une abstraction du robot qui permet de tester le même logiciel en simulation et sur le vrai robot. Cependant il nécessite un démarrage

asynchrone qui rend l'utilisation plus compliquée et faire perdre du temps. Enfin si le format URDF permet de modéliser un robot il ne permet pas de modéliser un monde et de fournir les informations nécessaires à sa simulation. Donc pour modéliser un environnement il n'est pas possible d'utiliser le format SDF. Si votre robot n'utilise pas ros-control, il peut-être également plus avantageux d'utiliser directement le format SDF.**[3]**

I.5.RViz 2 :

RViz est toujours un outil important et vraiment utile dans ROS 2, comme il l'était dans ROS 1. La fonction principale reste la même : donner un endroit pour voir en 3D votre robot et ses infos.

1.Installation:

Si la version complète du paquet ROS 2 Humble Desktop a été installée, RViz 2 aura déjà été inclus. Une commande comme celle-ci aura probablement été utilisée :
sudoaptinstall ros-humble-desktop
Dans le cas d'une installation légère, RViz 2 devra être installé séparément à l'aide de la commande suivante :
sudoaptinstall rviz2

2.Lancement de RViz 2 :

Pour démarrer RViz 2, un nouveau terminal doit être ouvert, l'environnement ROS 2 Humble doit être sourcé, puis la commande suivante doit être exécutée :
ros2 run rviz2 rviz2

3.Principales différences et considérations dans ROS 2 :

Même si les concepts de base sont maintenus, quelques différences par rapport à RViz dans ROS 1 peuvent être observées :

- Une architecture basée sur les nœuds est utilisée dans ROS 2 : RViz est exécuté comme un nœud ROS 2, conformément à l'architecture fondamentale du système.
- La communication est assurée via les rubriques, services et actions ROS 2, avec un middleware (tel que Cyclone DDS ou Fast DDS) responsable du transport des données.

- Un système de plugins puissant continue d'être utilisé. Des affichages tels que *RobotModel*, *LaserScan*, *PointCloud2* et *TF* sont fournis sous forme de plugins. De nouveaux plugins continuent d'être développés par la communauté ROS 2.
- Des fichiers de configuration .rviz sont utilisés pour enregistrer et charger les paramètres d'affichage. Une compatibilité entre les versions de RViz 2 au sein d'une même distribution ROS 2 est généralement assurée.
- Une API Python étendue (via rclpy) est disponible, et de nombreux outils exploitant Python pour interagir avec RViz peuvent être trouvés.

4.Tâches courantes dans RViz 2 :

Des affichages peuvent être ajoutés en cliquant sur « Ajouter » dans le panneau des affichages. Des types comme *RobotModel*, *LaserScan*, *PointCloud2* ou *TF* peuvent être sélectionnés.

Une configuration des affichages est possible via le panneau « Affichages » : le sujet écouté, les couleurs, tailles et autres propriétés visuelles peuvent être modifiés.

Des outils sont proposés dans la barre d'outils supérieure :

- *Interagir* : pour sélectionner et interagir avec les objets.
- *Déplacer la caméra* : pour changer le point de vue.
- *Sélectionner* : pour choisir des éléments affichés.
- *Focus Camera* : pour centrer la vue sur un point donné.
- *Mesurer* : pour mesurer des distances ou des angles.
- *Définir la pose initiale* : pour fixer manuellement la position initiale du robot.
- *Définir un objectif* : pour spécifier un but de navigation (dans le cadre d'une configuration de navigation).

Une compréhension du système TF est nécessaire pour une visualisation correcte. Les relations spatiales entre les référentiels sont gérées par TF.

Le bon fonctionnement des éditeurs TF doit être vérifié.

Une sauvegarde de la configuration peut être effectuée dans un fichier .

rviz (via *Fichier > Enregistrer la configuration*). Le fichier peut être chargé plus tard (via *Fichier > Ouvrir la configuration*) ou directement au lancement :

ros2 run rviz2 rviz2 -d your_config.rviz

5.Conseils pour ROS 2 Humble sur Ubuntu 22.04 (Rétrospective) :

- Un sourcing approprié de l’environnement devait être effectué dans chaque terminal avant toute commande ROS 2, y compris pour RViz. En général, la commande suivante était utilisée :

source /opt/ros/humble/setup.bash

Ajouter cette commande au fichier .bashrc était une pratique courante.

- Une vérification des noms des sujets ROS 2 publiés par les capteurs ou le robot devait être réalisée. Pour consulter les sujets disponibles, la commande suivante pouvait être utilisée :

ros2 topic list

En fonction des capteurs et des messages personnalisés utilisés, des plugins RVizsupplémentaires pouvaient être nécessaires. Par exemple, pour les messages vision_msgs, des plugins spécifiques à RViz étaient disponibles et pouvaient être installés depuis les dépôts ROS 2.**[4]**

I.6.Conclution :

ROS 2 Humble est une version Long-Term Support (LTS) majeure, offrant stabilité,performances et sécurité pour les développeurs et les projets industriels. Avec un support jusqu'en 2027, elle constitue un choix idéal pour :

Les applications critiques nécessitant une plateforme fiable et maintenue.
Les améliorations de performances, notamment en latence et gestion mémoire.
La sécurité renforcée avec SROS2 pour les systèmes robotiques sensibles.
L’écosystème moderne, compatible avec Ubuntu 22.04 et intégrant les dernières évolutions de ROS 2.

Comparée à Galactic (non-LTS) et Iron (version ultérieure non-LTS), Humble se distingue par sa longévité et sa robustesse, en faisant une référence pour les projets à moyen et long terme.**[5]**

Chapitre II

Robotic arm

II.1.Introduction :

Les bras robotiques représentent l'une des applications les plus emblématiques de la robotique moderne. Inspirés du bras humain, ces dispositifs mécaniques sont conçus pour accomplir des tâches répétitives, précises ou dangereuses avec une grande efficacité. Utilisés principalement dans l'industrie, mais aussi dans des domaines comme la médecine, l'aérospatiale ou la recherche scientifique, les bras robotiques jouent un rôle crucial dans l'automatisation et l'amélioration des performances humaines.

II.2.Historique des bras robotiques :

L'histoire des bras robotiques remonte au XXe siècle, avec les premières tentatives de mécanisation de tâches industrielles. Le premier bras robotique industriel, appelé "Unimate", a été développé au début des années 1960 par George Devol et JosephEngelberger. Il a été utilisé pour la première fois en 1961 dans une usine de General Motors pour manipuler des pièces métalliques chaudes sur une chaîne de montage.

Depuis cette première application, les bras robotiques n'ont cessé d'évoluer, gagnant en précision, en vitesse et en intelligence. Aujourd'hui, ils sont capables de réaliser des tâches extrêmement complexes, souvent en collaboration avec des opérateurs humains, dans des environnements variés et parfois hostiles.

II.3.Principaux développements technologiques :

Plusieurs avancées technologiques majeures ont permis l'évolution rapide des bras robotiques :

1. L'apparition des ordinateurs et des microprocesseurs :
   Le développement de l'informatique a permis la création de systèmes de contrôle sophistiqués capables de gérer des mouvements complexes en temps réel.
2. L'amélioration des moteurs et des actionneurs :
   Les progrès dans la conception des moteurs électriques, hydrauliques et pneumatiques ont permis d'augmenter la puissance, la précision et la fiabilité des bras robotiques.
3. Les systèmes de contrôle avancés :
   L'introduction de l'intelligence artificielle, du contrôle adaptatif et des algorithmes d'apprentissage a permis aux bras robotiques de s'ajuster automatiquement à leur environnement.
4. Les robots collaboratifs (ou cobots) :

Ces nouveaux types de bras robotiques sont conçus pour interagir directement avec les humains, en toute sécurité, ouvrant ainsi de nouvelles possibilités dans les environnements non industriels**.[1]**

Figure6. Bras robotique

II.4.Types de bras robotiques :

Les bras robotiques se distinguent principalement par **leur configuration mécanique**et la manière dont ils se déplacent dans l'espace. Chaque type possède des **caractéristiques spécifiques** qui le rendent adapté à des **applications précises**. Voici les types les plus courants:

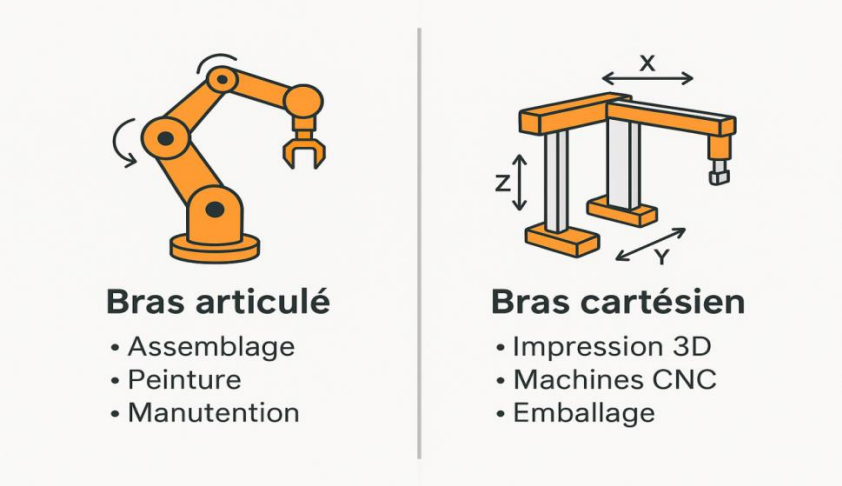


Figure7. Types de bras robotiques

1. Bras robotique articulé (Articulated Robot Arm) :

Caractéristiques :

- Inspiré du bras humain, avec plusieurs articulations rotatives.
- Généralement doté de 4 à 6 degrés de liberté, parfois plus.
- Offre une grande flexibilité et une amplitude de mouvement étendue.

Domaines d'utilisation :

- Très utilisé dans l'industrie automobile.
- Convient aux tâches comme la soudure, l'assemblage, la peinture et la manutention.

2. Bras robotique cylindrique (Cylindrical Robot Arm) :

Caractéristiques :

- Se déplace selon un système de coordonnées cylindriques.
- Comprend une rotation de la base, un mouvement linéaire vertical et un mouvement linéaire horizontal.
- Couvre un volume de travail en forme de cylindre.

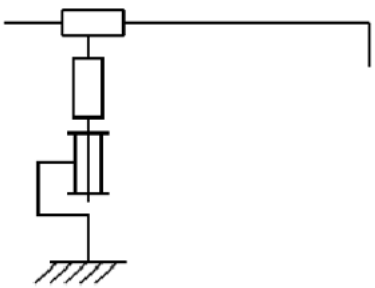


Figure8.le robot cylindrique

Figure9.Robot Seliko

Domaines d'utilisation :

- Utilisé pour le chargement/déchargement de machines.
- Adapté aux applications nécessitant des mouvements verticaux et horizontaux simples.

3. Bras robotique cartésien (Cartesian Robot Arm) :

Caractéristiques :

- Se déplace selon les axes X, Y et Z (coordonnées cartésiennes).
- Mouvement linéaire sur trois axes, parfois combiné avec des rotations.
- Facile à contrôler et très précis.

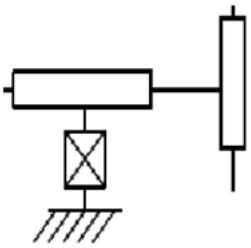


Figure10.robot Cartésien

Figure11.robot Toshiba

Domaines d'utilisation :

- Utilisé dans l'impression 3D, la découpe laser et l'usinage CNC.
- Idéal pour les tâches d'assemblage et de manipulation en ligne droite.

4. Bras robotique polaire (Polar Robot Arm) :

Caractéristiques :

- Fonctionne selon un système de coordonnées polaires (distance radiale et deux angles).
- Capacité de couvrir un volume hémisphérique.
- Structure moins rigide mais avec une bonne portée.

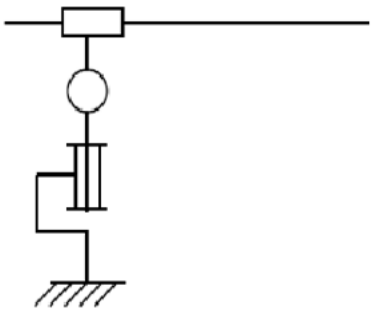


Figure12.robot sphérique

Figure13.ROBOT FANUC

Domaines d'utilisation :

- Utilisé pour le moulage, la peinture ou la pulvérisation dans des zones larges.

5. Bras SCARA (SelectiveComplianceAssembly Robot Arm) :

Caractéristiques :

- Conçu pour être flexible dans le plan horizontal mais rigide verticalement.
- Très rapide et précis pour les mouvements sur un plan.

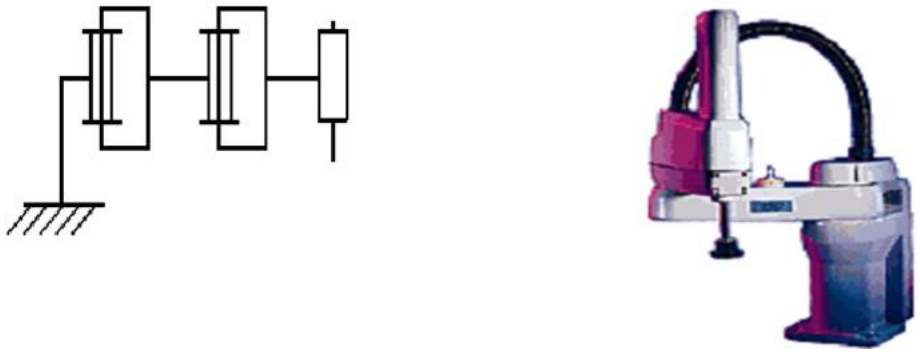


Figure14.schéma de les robots SCARA

Domaines d'utilisation :

- Idéal pour l'assemblage électronique, le placement de composants sur les circuits imprimés, et les tâches de pick-and-place.**[2]**

II.5. Les composants d'un bras robotique et leurs fonctions :

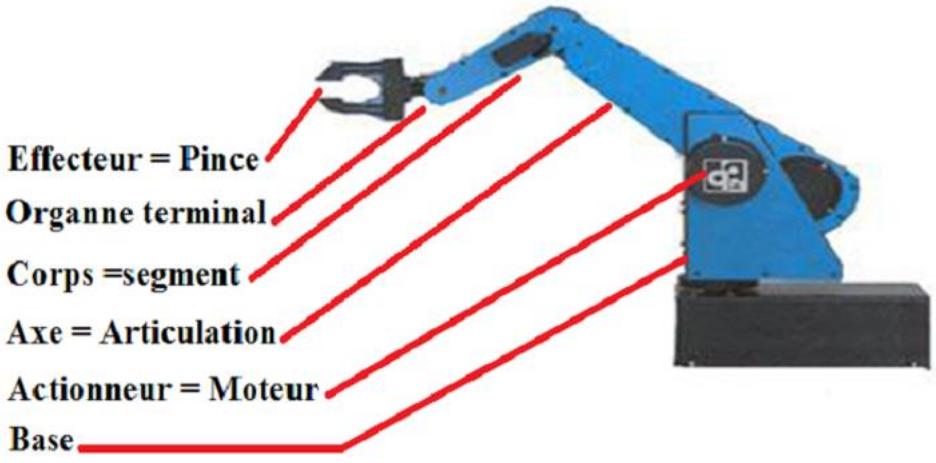


Figure15. Vocabulaires du bras de robot

Voici les **principaux éléments** qui composent un bras robotique :

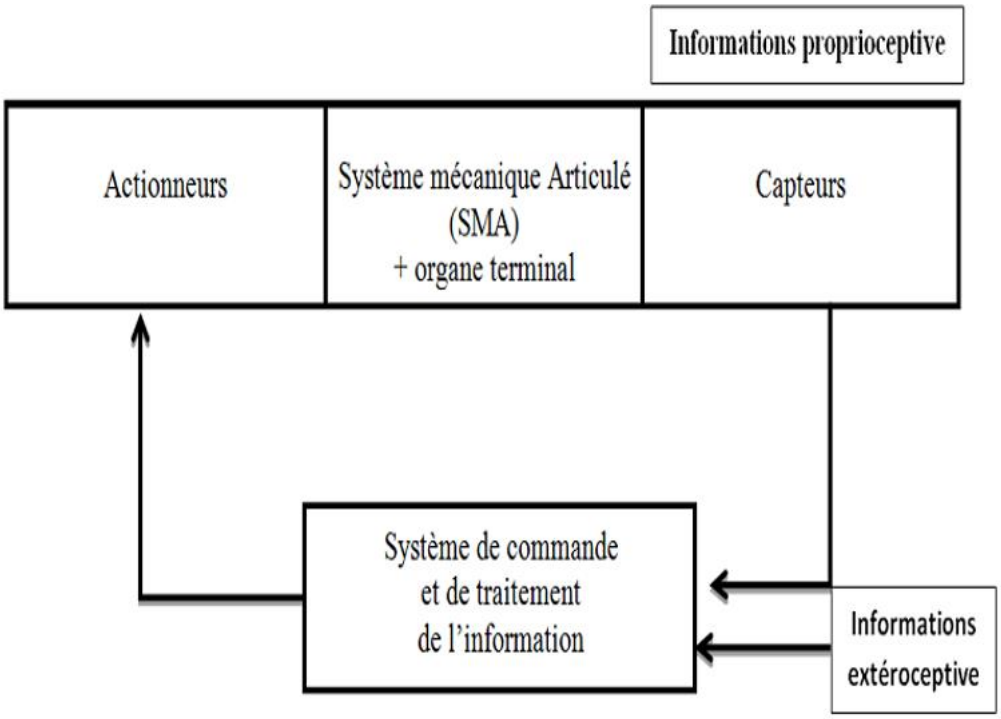


Figure16. Parties principales dans un robot manipulateur

1. **Les liaisons :**

- **Description :** Ce sont les parties rigides du bras, comparables aux os dans un bras humain.
- **Fonction :** Relient les articulations entre elles et assurent la structure du bras.
- **Interaction :** Transmettent le mouvement généré par les moteurs aux extrémités (ex: pince, outil).

2. **Les articulations (ou "joints") :**

- **Description :** Points de connexion entre deux segments permettant le mouvement (rotatif ou linéaire).
- **Fonction :** Donnent au bras sa mobilité (par ex., tourner, se plier, s'étendre).
- **Interaction :** Contrôlées par des moteurs, elles définissent les degrés de liberté du bras.

3. Les moteurs (ou actionneurs) :

- **Description :** Dispositifs mécaniques (souvent des moteurs électriques, hydrauliques ou pneumatiques).
- **Fonction :** Génèrent le mouvement au niveau des articulations.
- **Interaction :** Reçoivent des signaux de l'unité de commande pour déplacer les articulations.

4. Les capteurs (ou "senseurs") :

- **Description :** Dispositifs électroniques qui recueillent des informations sur l'environnement ou l'état du bras.
- **Types courants :**
  - Capteurs de position (encodeurs)
  - Capteurs de force/pression
  - Capteurs tactiles ou de vision
- **Fonction :** Fournissent un retour (feedback) pour ajuster les mouvements.
- **Interaction :** Envoient des données à l'unité de contrôle pour permettre un comportement adaptatif.

5. L'unité de commande (ou contrôleur) :

- **Description :** Cerveau du bras robotique (souvent un microcontrôleur ou un ordinateur embarqué).
- **Fonction :** Interprète les instructions, planifie les trajectoires, contrôle les moteurs.
- **Interaction :**
  - Reçoit des signaux des capteurs
  - Donne des ordres aux moteurs
  - Coordonne tous les composants pour effectuer des tâches précises
- Comment ces composants interagissent-ils ?

1. **L'unité de commande** reçoit une instruction (ex: déplacer un objet).
2. Elle calcule la trajectoire nécessaire et envoie des signaux aux **moteurs**.
3. Les **moteurs** déplacent les **articulations**, ce qui bouge les **liens**.
4. Les **capteurs** surveillent la position et l'état du bras et renvoient l'information à l'unité de commande.
5. Le système ajuste les mouvements en temps réel selon les retours des capteurs. **[3]**

II.6.Cinématique des bras robotiques :

II.6.1.Cinématique directe (Direct Kinematics – DK) :

La cinématique directe consiste à déterminer la position et l'orientation del'extrémité du bras (l'effecteur) à partir des valeurs connues des angles d'articulations ou desdéplacements linéaires

Comment ?

On utilise une chaîne de matrices de transformation(souvent basées sur les paramètres de Denavit-Hartenberg) pour calculer la position finale dans l'espace (X, Y, Z) et l'orientation (roll, pitch, yaw).

Unité :

- Simuler le mouvement du bras
- Visualiser la trajectoire
- Contrôler la main du robot (end-effector)

Exemple Pratique :

Un robot assembleur connaît les angles de ses moteurs :
→ il utilise la cinématique directe pour calculeroù se trouve sa pince dans l'espace3D afin d'atteindre un point précis.

II.6.2 :Cinématique inverse (Inverse Kinematics – IK) :

La cinématique inverse est l'opération inverse : on connaît la position etl'orientation souhaitées de l'extrémité du bras, et on cherche à déterminer les angles ou positionsdes articulations nécessaires pour atteindre cette cible.

Comment ?

- C'est souvent plus complexe mathématiquement, car il peut y avoir plusieurs solutions, ou parfois aucune.
- Utilisation d'équations trigonométriques, algèbre linéaire, ou algorithmes numériques (comme les Jacobiennes ou Gradient Descent).

Unité :

- Contrôle de trajectoire vers une cible

- Interaction avec des objets détectés par vision artificielle
- Planification de mouvements

Exemple Pratique :

Une caméra détecte une pièce sur une table à (X=40cm, Y=20cm, Z=10cm) → le robot utilise la cinématique inverse pour calculer les angles nécessaires de sesarticulations pour atteindre cette position et saisir la pièce.

II.7.Dynamique des bras robotiques :

La dynamique étudie le mouvement du brasen tenant compte des forces et descouples (torques) qui causent ce mouvement.

Contrairement à la cinématique, qui ne s'intéresse qu'aux trajectoires et aux positions, la dynamique prend en compte :

- La masse des éléments du bras
- Les inerties des segments
- Les accélérations et vitesses
- Les forces extérieures, comme la gravité ou les frottements

Deux formes de dynamique :

II.7.1 Dynamique directe (Forward Dynamics)

On connaît les forces et couples appliqués, et on cherche les accélérations et mouvements du bras.

II.7.2 Dynamique inverse (Inverse Dynamics)

On connaît la trajectoire souhaitée (vitesses, accélérations) et on veut calculer les forces et couples nécessaires pour la suivre.

Formules classiques utilisées :

Les équations de mouvement sont souvent dérivées à l'aide :

- Du principe de Lagrange
- ou des lois de Newton-Euler

Exemple d'équation simplifiée (1 articulation) :

$$\tau = I \cdot \alpha + C + G + F \backslash tau \ = \ I \ \backslash cdot \ \backslash alpha \ + \ C \ + \ G \ + \ F \tau = I \cdot \alpha + C + G + F$$

où :

- τ\tauτ : couple requis
- I: moment d'inertie

- α\alphaα : accélération angulaire
- C : force centrifuge/corriolis
- G : effet gravitationnel
- F : frottements

Exemples d'application de la dynamique :

1.Conception de moteurs

- Pour choisir le bon **type de moteur** (électrique, hydraulique…), il faut connaître le **couple nécessaire** pour déplacer chaque articulation.
- Exemple : Un bras qui soulève une charge de 5 kg à une distance de 50 cm devra générer un couple suffisant au niveau de l'épaule.

2 .Commandeadaptative / Contrôle en boucle fermée

- Dans un système dynamique, les conditions changent (poids déplacé, accélération, vitesse).
- La dynamique permet de construire des contrôleurs intelligents (PID, modèles prédictifs, etc.) qui ajustent les efforts automatiquement selon les conditions.

3 .Simulation réaliste

- Avant de construire un bras réel, les ingénieurs utilisent des simulateurs dynamiques (comme Gazebo, ROS, MATLAB Simscape) pour tester les mouvements en prenant en compte les forces réelles**.**

II.8.Commande des bras robotiques

Les bras robotiques doivent être **pilotés avec précision** pour exécuter des mouvements complexes, stables et sûrs. Cela se fait grâce à

II.8.1.La commande PID (Proportionnelle, Intégrale, Dérivée) :

Un des systèmes de contrôle les plus utilisés. Il ajuste la sortie (ex: moteur) en fonction

Formule du contrôleur PID :

de l'erreur entre la position réelle et la position désirée.

$$u(t) = Kpe(t) + Ki\int e(t)dt + Kdde(t)dtu(t)$$
$$= K\_p\, e(t) + K\_i \backslash int\, e(t)\, dt + K\_d \backslash frac\{de(t)\}\{dt\}u(t)$$
$$= Kpe(t) + Ki\int e(t)dt + Kddtde(t)$$

- Kp : Gain proportionnel → corrige rapidement
- Ki : Gain intégral → élimine l'erreur persistante
- K_d : Gain dérivé → stabilise le système

Avantages **:**

- Facile à implémenter
- Réactif et robuste pour des systèmes linéaires simples

Utilisation **:**

- Contrôle de position / vitesse dans les bras industriels
- Robots mobiles ou petits bras éducatifs

II.8.2. La commande adaptative :

Cette commande s'adapte dynamiquement aux changements dans le système ou l'environnement (ex: changement de charge ou friction).

Comment ça marche ?

Le système ajuste ses propres paramètres de contrôle en temps réel grâce à des algorithmes d'adaptation (comme MIT Rule, MRAC, etc.).

Avantages :

- Très utile pour les environnements non constants
- Résiste bien aux incertitudes du modèle

Utilisation :

- Robots manipulateurs qui changent de tâche
- Systèmes de production flexibles

- Bras collaboratifs (cobots)

II.8.3.La commande intelligente (IA) :

Utilise des algorithmes d'intelligence artificielle (comme les réseaux de neurones, la logique floue, les systèmes experts) pour apprendre et prendre des décisions demanière autonome.

Exemples :

- **Contrôle flou (fuzzylogic)** : imite le raisonnement humain pour gérer des incertitudes
- **Réseaux neuronaux** : apprennent des données pour améliorer la précision
- **Algorithmes génétiques / apprentissage par renforcement**

Avantages :

- Peut gérer des systèmes complexes et non linéaires
- Très efficace en robotique adaptative

Utilisation :

- Robots de service et médicaux
- Bras robotiques assistés par vision
- Systèmes interactifs ou collaboratifs **[4]**

II. 9.Applications des bras robotiques : Les applications industrielles :

II.9.1.Utilisation des bras robotiques dans l'industrie :

Les bras robotiques industriels sont devenus des éléments clés dans les chaînes de production modernes. Ils permettent :

- Une automatisation précise des tâches,
- Une augmentation de la productivité,
- Une réduction des coûts de main-d'œuvre,
- Une amélioration de la sécurité (en évitant aux humains les tâches dangereuses ou répétitives).

Figure 17: Une série de bras robotiques modernes dans un environnement industriel futuriste

Principaux domaines d'application :

1. Le soudage (le "welding")

- Les bras robotiques effectuent des soudures **à l'arc, par points ou au laser**, avec une précision constante.
- Utilisé dans :
    - l'industrie automobile (carrosseries),
    - la fabrication d'équipements lourds.

Avantage : qualité uniforme, travail continu.

Exemple : **KUKA** et **FANUC** proposent des robots de soudage pour Volkswagen ou Toyota.

2. L'assemblage (assembly) :

- Les bras robotiques placent, fixent et assemblent des composants avec une rapidité inégalée.
- Utilisé dans :
    - l'électronique (smartphones, PC),
    - les biens de consommation.

Avantage : grande précision, cadence élevée.

Exemple : ABB équipe les lignes d'assemblage chez Foxconn (fabricant d'Apple).

3. La peinture (painting) :

- Bras spécialisés dans le revêtement de surfaces par pulvérisation contrôlée.
- Utilisé dans :
    - l'automobile (peinture de carrosseries),
    - la fabrication de meubles.

Avantage : couverture homogène, réduction des déchets.

Exemple :Yaskawa et Staubli équipent des usines de BMW, Peugeot, etc.

4. L'emballage et la palettisation :

- Les bras robotiques saisissent, déplacent et empilent les produits pour l'expédition.
- Utilisé dans :
    - l'agroalimentaire,
    - la logistique.

Avantage : rapidité, flexibilité, endurance.

Exemple : Universal Robots est très utilisé dans les PME de production.

5. Le contrôle qualité (inspection visuelle) :

- Bras équipés de capteurs ou de caméras qui détectent les défauts ou mesures non conformes.

Avantage : élimine les erreurs humaines.

Exemple : Bosch, Siemens, et Tesla emploient cette technologie.

II.9.2 Applications des bras robotiques :

Le domaine médical :

Les bras robotiques sont utilisés en médecine pour effectuer des chirurgies de haute précision et assister les professionnels de santé.

Figure 18: Applications des bras robotiques (Le domaine médical)

1. **Chirurgie assistée par robot (chirurgie robotique) :**

La chirurgie robotique utilise des bras robotiques pilotés par un chirurgien pour réaliser des opérations avec une précision extrême et une invasion minimale.

Comment ça fonctionne ?

- Le chirurgien contrôle les bras robotiques via une console informatique.
- Les bras exécutent des gestes microscopiques impossibles à réaliser à la main.
- Une caméra 3D hautedéfinition permet une vision détaillée de la zone opérée.

Avantages :

- Incisions plus petites → moins de douleurs et de cicatrices.
- Moins de saignement.
- Temps de récupération plus court.
- Moins de risque d'infection.

Exemple de système :

- Da Vinci Surgical System : l'un des systèmes chirurgicaux robotiques les plus répandus au monde.

2. Rééducation robotique (robotique de réhabilitation) :

Les bras robotiques sont également utilisés pour **assister** les patients dans leur rééducation suite à un AVC, une blessure orthopédique ou des troubles moteurs.

Rôle des robots de rééducation :

- Accompagner les mouvements du patient pour retrouver mobilité **e**t forcemusculaire.
- Fournir un feedback en temps réel au patient et au thérapeute.
- Adapter les exercices au niveau de progrès du patient.

Exemples d'équipements :

- Armeo® Power (Hocoma) : pour la rééducation du bras et de la main.
- ReoGo™ (Motorika) : pour les membres supérieurs.

Autres applications médicales émergentes :

- **Robotique en imagerie médicale** : pour positionner les sondes et capteurs avec précision.
- **Assistance à la biopsie ou au placement d'implants**.

- **Télémédecine robotisée** : les bras robotiques permettent d'examiner un patient à distance.**[5]**

II.9.3.Applications des bras robotiques :

Autres domaines d'utilisation :

II.9.3.1 L'exploration spatiale (exploration de l'espace) :

Les bras robotiques jouent un rôle essentiel dans les missions spatiales. Ils sont conçus pour fonctionner dans des **environnements extrêmes**, sans gravité, et avec un **contrôle à distance précis**.

Fonctions principales :

- Manipuler des objets en orbite (satellites, panneaux solaires).
- Réparer ou entretenir des équipements spatiaux.
- Prélever des échantillons sur des planètes ou astéroïdes.

Exemples :

- **Canadarm** et **Canadarm2** : bras robotique utilisé sur la navette spatiale américaine et la Station spatiale internationale (ISS).
- **Bras de Perseverance (NASA)** : utilisé sur Mars pour analyser des roches.

Avantages : autonomie, précision, robustesse dans des milieux hostiles.

II.9.3.2 . Agriculture intelligente (agriculture robotisée) :

L'agriculture moderne adopte de plus en plus la robotique pour améliorer laproductivité et la durabilité**.**

Fonctions des bras robotiques :

- Récolte automatisée des fruits/légumes avec détection par caméra.
- Taille, désherbage ou pulvérisation ciblée.
- Manipulation délicate des plantes en serre.

Exemples :

- **Agrobot** : robot récolteur de fraises utilisant un bras robotique.
- **FARM-NG** : plateforme agricole modulaire avec bras pour diverses tâches.

Avantages : gain de temps, réduction de l'usage des produits chimiques, efficacité en main-d'œuvre.

II.9.3.3 Logistique et entreposage (supplychain& entrepôts) :

Les bras robotiques sont largement utilisés pour automatiser la manutention et la gestion des stocks.

Fonctions principales :

- Prélèvement (picking) d'objets dans des entrepôts (comme Amazon).
- Tri des colis et palettisation.
- Chargement/déchargement des camions.

Exemples *:*

- **Amazon Robotics** : utilise des bras robotiques pour le tri et l'emballage.
- **Boston Dynamics – Stretch** : robot logistique avec bras flexible pour déplacer des boîtes.

Avantages : rapidité, réduction des erreurs, fonctionnement 24/7. **[6]**

II.10.Conclusion :

Le bras robotique constitue une composante essentielle dans le domaine de la robotique moderne. À travers ce chapitre, nous avons exploré sa structure, ses degrés de liberté, ses mécanismes de mouvement, ainsi que les principes de commande qui permettent son fonctionnement. Qu'il s'agisse de bras industriels, médicaux ou éducatifs, ces systèmes mécaniques intelligents jouent un rôle crucial dans l'automatisation des tâches complexes, répétitives ou dangereuses.

Grâce à l'intégration de technologies avancées telles que les capteurs, les actionneurs, et les systèmes de contrôle, le bras robotique est devenu de plus en plus précis, rapide et adaptable. Il illustre parfaitement la convergence entre la mécanique, l'électronique et l'informatique.

Ainsi, le bras robotique n'est pas seulement un outil technologique : il est le reflet des progrès de l'ingénierie et une porte ouverte vers des applications toujours plus innovantes dans l'industrie, la santé, la recherche et l'exploration.

Chapitre III
Les capteurs et les actionneurs utilisé dans les bras robotiques

III.1 Introduction :

Les capteurs et les actionneurs représentent des éléments fondamentaux dans la conception et le fonctionnement des bras robotiques. Ils jouent un rôle central en permettant au robot d'exécuter des tâches avec précision et adaptabilité. Les capteurs fournissent au bras robotique des informations sur son environnement, telles que la position, la vitesse, la direction ou encore la force appliquée. De leur côté, les actionneurs transforment les signaux électriques en mouvements mécaniques, assurant ainsi la mobilité des articulations du bras.

Ces composants travaillent en synergie pour accomplir les fonctions du bras robotique. Les capteurs transmettent des données en temps réel à l'unité de contrôle, qui les analyse et décide des actions à entreprendre, comme activer une articulation ou ajuster la force de préhension. Cette boucle continue de perception et d'action constitue le fondement du comportement intelligent et adaptatif du bras robotisé, permettant son utilisation dans des domaines variés tels que l'assemblage industriel, la chirurgie ou l'interaction homme-machine.

Pour faciliter le développement et la validation de tels systèmes avant leur mise en œuvre réelle, des environnements de simulation comme Gazebo sont largement utilisés. Gazebo permet de simuler de manière réaliste les capteurs (tels que les capteurs de distance, de couple ou les caméras) ainsi que le comportement des actionneurs. Grâce à cette plateforme, il est possible de créer des modèles virtuels de bras robotiques, de tester leurs réactions face à divers scénarios, et d'optimiser les algorithmes de contrôle sans risquer d'endommager des équipements physiques, réduisant ainsi les coûts et les erreurs de développement**.[1]**

III.2 Les capteurs utilisés dans les bras robotiques :

Les capteurs jouent un rôle essentiel dans le contrôle précis du mouvement des bras robotiques. Ils permettent au système de connaître la position et la vitesse de chaque articulation, ce qui facilite la prise de décisions précises pour réaliser les mouvements souhaités. Parmi les capteurs les plus importants, on trouve :

III.2.1 Capteurs de position et de vitesse :

1. Codeurs :

Figure19 : codeurs

Les codeurs sont des dispositifs utilisés pour mesurer la position et la vitesse de rotation des articulations. Il en existe plusieurs types, mais en général, ils transforment un mouvement mécanique en un signal électrique utilisé pour déterminer la position et la vitesse.

Codeurs absolus (AbsoluteEncoders) :

- Fournissent une valeur unique pour chaque angle ou position.
- Conservent la position même après une coupure de courant.
- Utilisés dans les applications nécessitant une connaissance précise de la position sans réinitialisation lors du démarrage.

Codeurs incrémentaux :

- Mesurent les variations de position par rapport à un point de référence.
- Nécessitent une position de départ appelée "Home" au démarrage.
- Très répandus en raison de leur simplicité et de leur faible coût.

Utilisation :

- Montés généralement sur les articulations du bras pour mesurer le nombre de rotations ou les angles.
- Utilisés pour contrôler la position et la vitesse en suivant les signaux produits.

2. Potentiomètres :

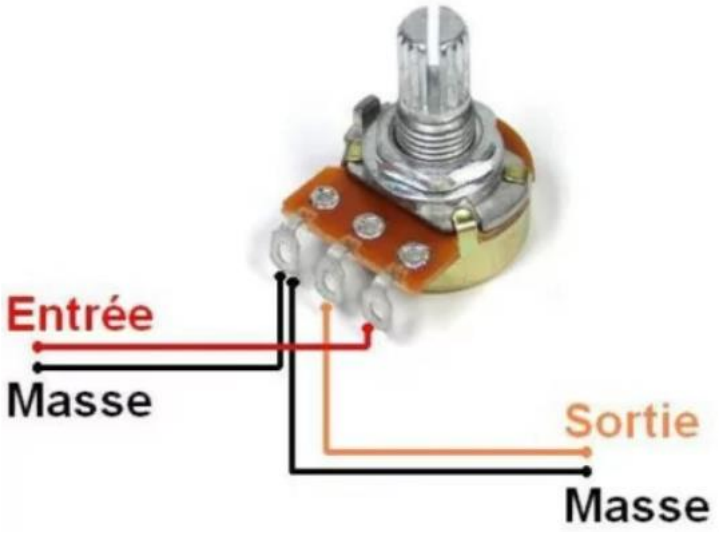


Figure20. Potentiomètres

Il s'agit d'un capteur analogique qui mesure un angle à travers une résistance électrique variable en fonction de la rotation de l'axe de l'articulation.

- Composé d'une résistance variable et d'un axe rotatif.
- Lorsque l'axe tourne, la résistance change, générant un signal électrique proportionnel à l'angle.

Utilisation :

- Utilisé pour mesurer directement l'angle d'une articulation.
- Fréquent dans les applications où la précision requise est faible à moyenne.
- Facile à installer et à utiliser.

3. Gyroscopes (Capteurs de vitesse angulaire) :

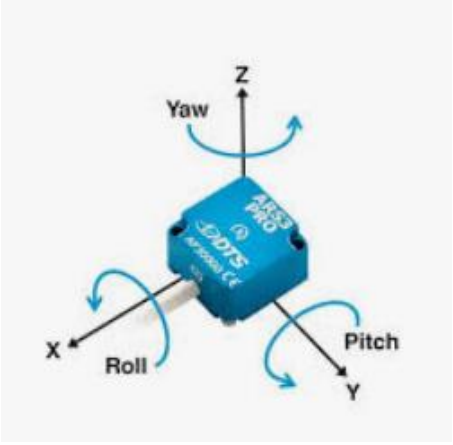


Figure21. Gyroscopes (Capteurs de vitesse angulaire)

Le gyroscope mesure la vitesse de rotation autour d'un axe donné, c'est-à-dire la vitesse angulaire.

- Fonctionne selon le principe de conservation du moment angulaire ou à partir d'effets gyroscopiques intégrés dans les systèmes MEMS (Micro-Electro-MechanicalSystems).
- Génère des signaux représentant la variation de l'angle au cours du temps.

Utilisation :

- Utilisé pour déterminer la vitesse angulaire des articulations du bras.
- Complète les données des codeurs pour une meilleure précision du mouvement.
- Utile pour compenser les vibrations ou pour des applications de stabilisation et de contrôle dynamique.**[2]**

1. Capteurs de force/couple :

Figure22. Capteurs de force/couple

Principe :

- Les **ca**pteurs de force et de couple sont des dispositifs utilisés pour mesurer les forces exercées sur le bras robotique dans différentes directions, ainsi que le couple (ou le moment de rotation) appliqué sur les articulations.
- Ces capteurs utilisent des matériaux sensibles comme des **c**ristaux piézoélectriques ou des résistances variables pour mesurer les variations de pression ou de force appliquées au bras.

Types de capteurs :

- **Capteurs de force uniaxiaux (Single-axis force sensors) :** Mesurent la force dans une seule direction.
- **Capteurs de couple (Torque sensors) :** Mesurent le couple (rotation) autour d'un axe.
- **Capteurs de force et de couple multiaxiaux (Multi-axis force/torque sensors) :** Mesurent les forces et les couples sur plusieurs axes (X, Y, Z).

Principe de fonctionnement :

- Les **c**apteurs de force et de couple fonctionnent en mesurant les changements de forme du matériau sensible lorsqu'il est soumis à une force. Ces changements sont ensuite convertis en signaux électriques utilisés pour déterminer la quantité de force ou de couple.
- **Exemple :** Lors de l'assemblage d'une pièce ou du soudage, les capteurs de force/couple sont utilisés pour ajuster la force appropriée appliquée entre le bras et la pièce.

Utilisation :

- Utilisés dans des applications nécessitant des mesures précises des forces appliquées, telles que :
- Le soudage, l'assemblage et des tâches de précision nécessitant un contrôle de la force.
- Surveillance du couple pour s'assurer que la force appliquée ne dépasse pas les limites mécaniques des composants.
- Interactions sensibles, comme les mouvements des bras robotiques dans les robots humanoïdes où ces capteurs sont utilisés pour simuler le toucher humain.

2. Capteurs de toucher :

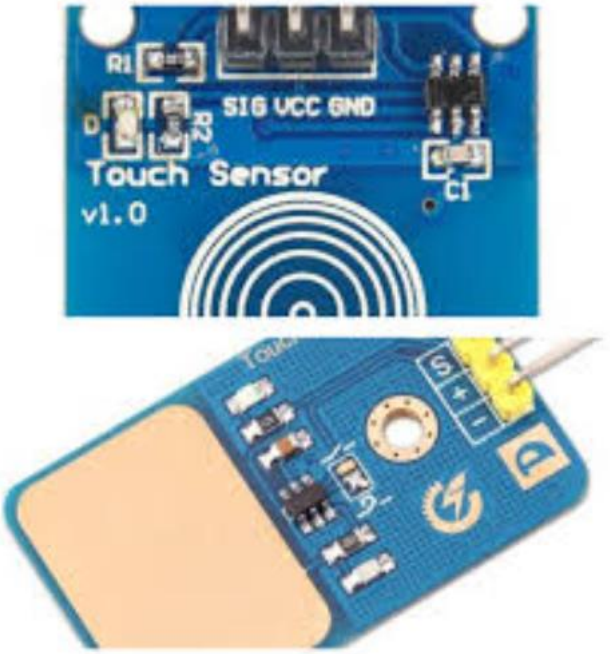


Figure23. Capteurs de toucher

Principe :

- Les **c**apteurs de toucher sont des dispositifs sensibles utilisés pour détecter le contact et les petites forces qui agissent sur une surface spécifique.
- Ces capteurs fonctionnent en utilisant diverses technologies, telles que la piézoélectricité ou la résistance variable, pour détecter le contact avec des objets ou des surfaces.

Principe de fonctionnement :

- Les **c**apteurs de toucher mesurent la pression exercée sur une surface sensible, ce qui permet de déterminer l'emplacement précis du contact.
- **Exemple :** Les capteurs de toucher sur les doigts d'un robot humanoïde peuvent détecter la forme d'une surface ou des variations subtiles de pression ou de force.

Utilisation :

- Utilisés dans des applications nécessitant la détection du contact ou des forces légères, telles que :
- Robots humanoïdes qui imitent les mouvements de la main.
- Chirurgie robotique où une détection précise de la pression est requise sur les tissus.
- Robots industriels manipulant des matériaux fragiles, où une précision dans la mesure de la pression sur ces matériaux est nécessaire.**[3]**

III.2.2 Capteurs de proximité :

Les capteurs de proximité sont utilisés dans les bras robotiques pour détecter les objets proches, ce qui permet de mesurer les distances et d'interagir de manière sécurisée et efficace avec l'environnement. Ces capteurs sont essentiels dans de nombreuses applications, telles que l'évitement de collisions et la navigation dans des environnements non équipés.

1.Capteurs à infrarouge :

Figure24. Capteurs à infrarouge

Principe de fonctionnement :

- Les capteurs à infrarouge fonctionnent en émettant un faisceau d'infrarouges vers un objet. Lorsque ce faisceau rencontre un objet, il est réfléchi et revient vers le capteur.
- Le temps qu'il faut à la lumière pour revenir est mesuré, ce qui permet de déterminer la distance entre le capteur et l'objet.
- Certains capteurs utilisent également des **capteurs de sécurité** pour mesurer l'intensité du rayonnement réfléchi et la convertir en un signal électrique représentant la distance.

Applications :

- **Détection des objets proches** : Ils sont utilisés dans les robots pour détecter des obstacles ou des objets proches qui peuvent affecter le mouvement du robot.
- **Protection contre les collisions** : Dans les robots industriels ou mobiles, les capteurs IR aident à éviter les collisions avec des objets proches.

- **Mesure dans des environnements fermés** : Ces capteurs sont couramment utilisés dans des environnements où la précision des mesures à courte distance est plus importante que les longues distances.

3. **Capteurs à ultrasons :**

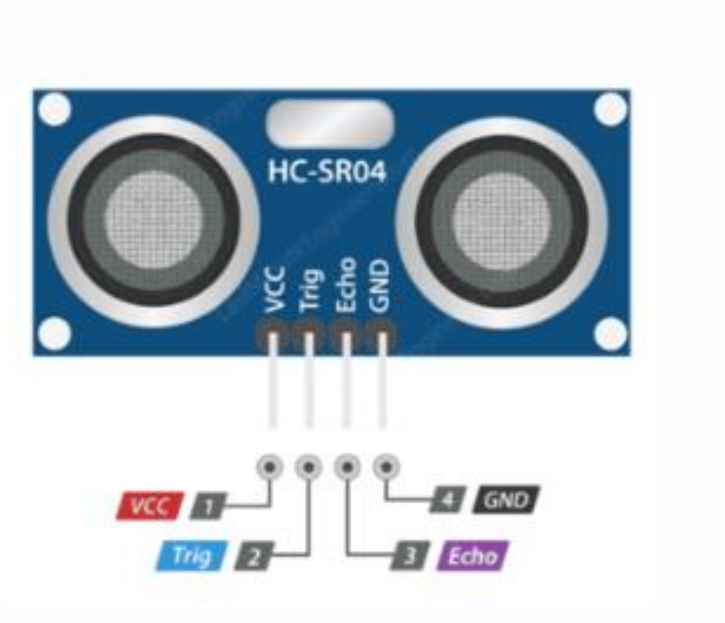


Figure25. Capteurs à ultrasons

Principe de fonctionnement :

- Les capteurs à ultrasons utilisent des ondes sonores à haute fréquence (au-delà de la portée de l'oreille humaine) pour mesurer les distances.
- Le capteur émet des impulsions sonores à haute fréquence vers un objet et mesure le temps qu'il faut pour que le son revienne après avoir été réfléchi par l'objet.
- Comme les capteurs IR, ces informations sont converties en distance entre le capteur et l'objet.

Applications :

- **Mesure des distances** : Ces capteurs peuvent être utilisés pour déterminer la distance entre le robot et les obstacles ou objets dans son environnement.
- **Évitement des collisions** : Ils sont utilisés dans les robots mobiles pour les aider à éviter les collisions avec des objets proches.
- **Navigation dans des environnements complexes** : Ces capteurs sont utilisés dans des robots de navigation, tels que les robots industriels ou les drones (UAV), pour déterminer les distances et les altitudes.**[2]**

III.3 Moteurs utilisés dans les bras robotiques :

Les moteurs sont des composants essentiels qui permettent aux bras robotiques de se déplacer et de contrôler les articulations. Il existe plusieurs types de moteurs utilisés dans les bras robotiques, et le choix du moteur dépend de l'application spécifique. Dans cette section, nous aborderons les moteurs électriques utilisés dans les bras robotiques, notamment les **servo-**moteurs, les moteurs pas à pas (stepper motors)**,** et les moteurs à courant continu (DC)**.**

III.3.1 Moteurs électriques :

1. Servo-moteurs :

Figure26. Servo-moteurs

Principe de fonctionnement :

- Les servo-moteurs sont des moteurs électriques conçus pour contrôler précisément **l'angle** ou la **position** d'un membre robotique. Ils sont utilisés principalement dans les applications nécessitant un contrôle précis des articulations du robot.
  - Un servo-moteur est constitué d'un moteur électrique de petite taille, d'un **système de retour d'information (feedback)**, comme des **encodeurs** ou des **potentiomètres**, qui lui permet de connaître la position actuelle de l'axe.
  - Le moteur est contrôlé par une **modulation de largeur d'impulsion (PWM)** qui ajuste l'angle à atteindre.

Applications :

- Utilisés principalement pour **contrôler les articulations** des bras robotiques, fournissant une haute précision dans la détermination de la position.
- **Applications de précision**, comme l'ouverture et la fermeture des **pinces** ou le mouvement d'outils avec une grande exactitude.

Avantages :

- **Haute précision** dans le contrôle de **la position**.
- **Maintien stable de la position**.

Inconvénients :

- **Plage de mouvement limitée**, généralement entre 0 et 180 degrés.
- Peut être**plus lent** que d'autres types de moteurs.

2. Moteurs pas à pas :

Figure27. Moteurs pas à pas

Principe de fonctionnement :

- Les **moteurs pas à pas** sont des moteurs électriques qui effectuent des **mouvements en étapes fixes**, contrairement à un moteur conventionnel qui tourne en continu.
- Le moteur fonctionne en déplaçant les **bobines magnétiques** de manière ordonnée, ce qui permet de réaliser des mouvements dans des **angles définis**.
- Le contrôle du moteur pas à pas se fait en envoyant des **signaux séquencés** pour contrôler le nombre d'étapes ou l'angle à atteindre.

Applications :

- Utilisés dans des applications nécessitant un **mouvement précis** et **répétitif**.
- Idéals pour les **bras robotiques** qui doivent réaliser des mouvements linéaires ou de positionnement exacts.

Avantages :

- **Haute précision** dans le **contrôle du mouvement**.
- Ne nécessite pas de **système de rétroaction** comme les servo-moteurs.

Inconvénients :

- **Plus lent** que les servo-moteurs dans certains cas.
- Moins **efficace énergétiquement**.

4. Moteurs à courant continu :

Figure28. Moteurs à courant continu

Principe de fonctionnement :

- Les **moteurs à courant continu (DC)** sont des moteurs électriques qui fonctionnent avec un **courant continu**. Ils sont contrôlés en ajustant le **courant traversant le moteur**, ce qui influence la **vitesse** et le **couple** du moteur.
- Les moteurs à courant continu ne fournissent pas un contrôle aussi précis de la position que les moteurs à pas ou les servo-moteurs, mais ils sont très efficaces dans les applications nécessitant **de grandes vitesses**.

Applications :

- Utilisés dans des applications nécessitant des **vitesses élevées**, comme des mouvements rapides dans les bras robotiques.
- Parfois utilisés avec des **systèmes de rétroaction** (comme des encodeurs) pour obtenir un certain contrôle de position.

Avantages :

- **Vitesse élevée**.
- **Coût relativement faible** comparé aux autres moteurs.
- Fournissent un **couple élevé** à haute vitesse.

Inconvénients :

- **Contrôle de position faible** par rapport aux servo-moteurs et moteurs pas à pas.
- **Moins précis** dans le mouvement.

Choisir le moteur adapté

Le choix du moteur pour un bras robotique dépend des exigences de l'application :

- **Les servo-moteurs** sont idéaux pour un contrôle précis de l'**angle** des articulations.
- **Les moteurs pas à pas** sont parfaits pour les applications nécessitant des **mouvements linéaires** ou très **précis**.
- **Les moteurs DC** sont préférés lorsque des **vitesses élevées** et un **couple élevé** sont nécessaires.**[4]**

III.3.2 Moteurs hydrauliques et pneumatiques :

Les moteurs **hydrauliques** et **pneumatiques** sont utilisés dans les applications nécessitant **de fortes forces** ou **des capacités de charge élevées**. Ces moteurs sont idéaux pour des systèmes où les moteurs électriques ne sont pas suffisants, car ils permettent de générer des puissances importantes de manière compacte.

Moteurs hydrauliques :

Figure29. Moteurs hydrauliques

Principe de fonctionnement :

- Les **moteurs hydrauliques** fonctionnent en utilisant la **pression du fluide** (généralement de l'huile) pour produire un mouvement mécanique. Le fluide est comprimé dans des cylindres ou moteurs hydrauliques, ce qui génère une **grande force**.
- Ces moteurs sont contrôlés par des **vannes** régulant le débit du fluide, permettant ainsi un **contrôle précis** de la vitesse et du mouvement.
- Les moteurs hydrauliques génèrent des **couples élevés** même à faible vitesse.

Applications :

- Les moteurs hydrauliques sont utilisés dans des applications nécessitant une **force importante** avec un **volume relativement petit**, tels que les **bras robotiques lourds** ou dans des **systèmes hydrauliques industriels**.
- Ils sont également utilisés pour des **mouvements puissants**, comme le **levage d'objets lourds** ou la manipulation d'objets dans des environnements difficiles.

Avantages :

- Génération de **couples élevés**.
- **Contrôle précis** de la vitesse et de la force.
- **Haute puissance dans un format compact**.

Inconvénients :

- **Maintenance régulière** en raison des fuites de fluides et de l'usure des composants.
- Moins **économe en énergie** par rapport aux moteurs électriques dans certaines applications.
- Les **fuites de fluides** peuvent poser des problèmes environnementaux.

Moteurs pneumatiques :

Figure30. Moteurs pneumatiques

Principe de fonctionnement :

- Les **moteurs pneumatiques** fonctionnent en utilisant de **l'air comprimé** pour produire un mouvement mécanique. L'air comprimé est injecté dans des **cylindres pneumatiques**, générant un mouvement du piston qui génère la force.
- Bien que similaires aux moteurs hydrauliques, ils utilisent de **l'air comprimé** plutôt qu'un fluide.

Applications :

- Les moteurs pneumatiques sont utilisés dans des applications nécessitant des **mouvements rapides**, mais avec des **forces modérées**, comme dans les **robots légers** ou les outils **pneumatiques**.
- Ils sont également utilisés dans des environnements où des **moteurs électriques** ne peuvent pas fonctionner efficacement, comme dans des **zones à risque d'explosion**.

Avantages :

- **Réponse rapide** et **vitesse élevée**.
- **Facilité de maintenance** par rapport aux moteurs hydrauliques.
- Aucun problème de **fuite de fluide**, contrairement aux moteurs hydrauliques.

Inconvénients :

- **Couple faible** comparé aux moteurs hydrauliques.
- Dépendance à l'**air comprimé**, nécessitant des sources d'air spécifiques.
- **Moins précis** dans le contrôle des mouvements comparé aux moteurs électriques ou hydrauliques.

Choisir le moteur adapté

- **Les moteurs électriques** sont idéaux pour les applications nécessitant une **précision élevée** et des **vitesses variables**.
- **Les moteurs hydrauliques** sont préférés pour les applications nécessitant des **forces très élevées**, comme dans les **robots lourds** ou les systèmes nécessitant des charges lourdes.
- **Les moteurs pneumatiques** sont utilisés dans des applications nécessitant des **mouvements rapides** mais avec un **couple plus faible**, comme dans des **robots légers** ou des outils **pneumatiques**.**[5]**

III.4 Conclusion :

Dans ce chapitre, nous avons étudié le rôle fondamental des capteurs et des actionneurs dans le fonctionnement des bras robotiques. Les capteurs permettent au robot de percevoir son environnement et son propre état (position, vitesse, température, force, etc.), tandis que les actionneurs assurent le mouvement et l'exécution des commandes de contrôle.

Ces deux composants sont étroitement liés : les capteurs fournissent les données nécessaires à la prise de décision, et les actionneurs traduisent ces décisions en actions physiques. Leur bonne intégration garantit à la fois précision, efficacité et sécurité du bras robotique dans l'accomplissement de ses tâches.

L'évolution constante de ces technologies, notamment avec l'arrivée de capteurs intelligents et d'actionneurs plus performants, contribue à rendre les robots plus autonomes et réactifs. Maîtriser leur fonctionnement est donc indispensable pour concevoir des systèmes robotiques modernes, capables de s'adapter à des environnements dynamiques et complexes.

Chapitre IV
Simulation dans ROS 2 et Gazebo d'un bras robotique

IV. 1 Introduction :

La simulation joue un rôle essent eloppement de la robotique en général, et dans celui des bras robotiques en particulier. Elle permet aux chercheurs et aux ingénieurs de tester des idées et de concevoir des systèmes dans un environnement virtuel, sans les coûts ou les risques associés à une mise en œuvre physique.

Ce chapitre met en pratique les connaissances acquises dans les chapitres précédents — notamment sur ROS 2, les bras robotiques, les capteurs et les actionneurs — afin de construire un modèle de bras robotique dans un environnement de simulation.

Les objectifs principaux de ce chapitre sont les suivants :

Concevoir un modèle 3D d'un bras robotique dans un environnement simulé.

Configurer l'environnement ROS 2 pour lancer la simulation et contrôler le bras.

Exécuter des mouvements et contrôler les articulations à l'aide des données des capteurs.

Analyser les performances et le comportement du bras dans différents scénarios.

IV. 2Architecture générale du projet :

Ce projet vise à concevoir et simuler un bras robotique dans un environnement virtuel intégré, en s'appuyant sur deux des outils les plus puissants dans le domaine de la robotique : ROS 2 (Robot Operating System) et Gazebo, le simulateur 3D de robots. L'objectif principal est de créer un modèle numérique précis du bras, de l'activer dans un environnement de simulation réaliste, et d'en contrôler les mouvements avec une grande précision.

Le projet est un espace de travail (workspace) composé de packages suivants :

- arduinobot_bringup
- arduinobot_control
- arduinobot_description
- arduinobot_moveit2
- arduinobot_msgs
- arduinobot_remote

Objectifs techniques du projet :

Construction du modèle physique et cinématique du brasarduinobot.xacro :

Création d'un modèle détaillé du bras robotique en utilisant XACRO (une extension de URDF), en définissant les liens (links), articulations (joints), propriétés physiques (masse, inertie), ainsi que les aspects visuels et collisionnels.

Intégration des paramètres de ROS 2 Control directement dans le fichier XACRO pour définir les interfaces de commande et les types de contrôleurs utilisés.

Intégration de la simulation avec Gazebo avec gazebo.launch.py :

Chargement du modèle dans l'environnement Gazebo, qui offre une simulation physique précise permettant de tester le comportement du bras dans des conditions proches de la réalité (interaction avec des objets, application de forces, simulation des capteurs).

Utilisation des plugins Gazebo pour simuler les moteurs et publier les états des articulations vers ROS2.

Développement du système de contrôle avec ROS 2 à travers le fichier de configuration controller_manager_config.yaml:

Configuration de divers contrôleurs via ROS 2 Control, tels que le joint_trajectory_controller, pour contrôler la position et la vitesse des articulations.

Développement de nœuds ROS 2 pour envoyer des commandes logicielles, permettant de déplacer le bras vers des positions précises, exécuter des trajectoires complexes ou accomplir des tâches spécifiques.

Visualisation et analyse des données de simulation en appelant le fichier arduinobot_display.launch.py:

Utilisation de l'outil RViz pour visualiser le modèle du bras et ses mouvements en temps réel, ainsi que les données des capteurs simulés.

Analyse des données de simulation (angles et vitesses des articulations) pour évaluer la performance du système de contrôle et la précision des mouvements.Valeur ajoutée du projet :

Ce projet ne se limite pas à une approche théorique, mais fournit une application pratique montrant comment exploiter la puissance de ROS 2 et Gazebo pour :

Concevoir et développer des robots de manière efficace avant la mise en œuvre réelle.

Tester différentes stratégies de contrôle en toute sécurité.

Explorer le comportement robotique dans divers environnements sans les coûts et les risques liés au matériel physique.

IV. 3 Créer espace de travail et packages :

IV. 3.1 Créer espace de travail:

Un espace de travail ROS 2 est un répertoire où on développe les packages ROS 2. Pour créer un répertoiremy_ros2_ws/src del'espace de travail, on utilise la commande suivante :

```
mkdir -p ~/my_ros2_ws/src
cd ~/my_ros2_ws
```

Cela crée un répertoire ros2_ws dans le répertoire personnel, avec un sous-répertoire src à l'intérieur. Tous lespackages ROS 2 seront placés dans le répertoire src.

- Construisez l'espace de travail :

```
colconbuild --symlink-install
```

Cela construira l'espace de travail pour la première fois, créant des répertoires build, install et log.

- Sourcez l'espace de travail :

```
sourceinstall/setup.bash
```

Il est essentiel que ces packages soient trouvés par ROS 2. Cet espace de travail devra être sourcé à chaque ouverture d'un nouveau terminal. Pour que cela soit fait de manière permanente, cette ligne peut être ajoutée au fichier `~/.bashrc`.

```
source /opt/ros/humble/setup.bash
```

IV.3.2 Créer un Package ROS 2 :

Un package ROS 2 est un ensemble de fichiers et de répertoires qui contiennent un ou plusieurs nœuds ROS 2, ainsi que tous les fichiers de configuration, les données et autres éléments nécessaires.

Allez dans le répertoire src de votre espace de travail :

```
cd ~/ros2_ws/src
```

- Utilisez la commande ros2 pkgcreate pour créer un package :

```
ros2 pkg create --build-type ament_cmake --license Apache-2.0 my_package
```

ou, pour un package Python :

```
ros2 pkg create --build-type ament_python --license Apache-2.0 my_package
```

my_package doit être remplacé par le nom souhaité pour le package.

Explication de la commande ros2 pkgcreate :

- --build-type ament_cmake est utilisé pour les packages C++
- .--build-type ament_python est utilisé pour les packages Python
- .--license Apache-2.0 spécifie la licence du package. Il peut être choisi une autre licence si cela est souhaité.

- Retournez à la racine de votre espace de travail et construisez le package :

```
cd ~/ros2_ws
colconbuild
sourceinstall/setup.bash
```

- colconbuild: Construit les packages dans l'espace de travail. Colcon est le système de construction recommandé pour ROS 2.
- sourceinstall/setup.bash : Sourcez l'espace de travail. Ajoute les variables d'environnement nécessaires pour que ROS 2 puisse trouver et utiliser les

IV.3.2.1 Structure d'un Package C++ ROS 2 :

a. Package arduinobot_bringup :

On prend comme example le package arduinobot_bringup. Ce package est structuré comme suit.

```
arduinobot_bringup/
├──launch
│   └──arduinobot_sim.launch.py
├── src
    ├── CMakeLists.txt
└──package.xml
```

- launch : Scripts de lancement (souvent en Python dans ROS 2) pour démarrer les nœuds et paramétrer l'environnement.

- **arduinobot_sim.launch.py**

```
import os
from launch import LaunchDescription
fromlaunch_ros.actions import Node
fromlaunch.actions import IncludeLaunchDescription
fromlaunch.launch_description_sources import PythonLaunchDescriptionSource
fromament_index_python.packages import get_package_share_directory
defgenerate_launch_description():
gazebo = IncludeLaunchDescription(
PythonLaunchDescriptionSource(
os.path.join(get_package_share_directory("arduinobot_description"), "launch",
"arduinobot_gazebo.launch.py")
 ) )
controllers = IncludeLaunchDescription(
PythonLaunchDescriptionSource(
os.path.join(get_package_share_directory("arduinobot_control"), "launch",
"arduinobot_controllers.launch.py")
 )  )
moveit = IncludeLaunchDescription(
PythonLaunchDescriptionSource(
os.path.join(get_package_share_directory("arduinobot_moveit2"), "launch",
"arduinobot_moveit.launch.py")
  ))
returnLaunchDescription([
gazebo,
controllers,
moveit
 ])
```

- src : Contient les fichiers source (.cpp) avec l'implémentation des nœuds ROS 2.

- CMakeLists.txt: Ce fichier contient les instructions de construction pour le package. Il spécifie comment compiler le code source, quels exécutables construire et comment installer le package. C'est un élément essentiel pour construire tout package C++ ROS2

```
cmake_minimum_required(VERSION 3.8)
```

```
project(arduinobot_bringup)
if(CMAKE_COMPILER_IS_GNUCXX OR CMAKE_CXX_COMPILER_ID MATCHES
"Clang")
add_compile_options(-Wall -Wextra -Wpedantic)
endif()
find_package(ament_cmake REQUIRED)
find_package(rclcpp REQUIRED)
  DIRECTORY
launch
  DESTINATION
share/${PROJECT_NAME}
)
if(BUILD_TESTING)
find_package(ament_lint_auto REQUIRED)
set(ament_cmake_copyright_FOUND TRUE)
ament_lint_auto_find_test_dependencies()
endif()
ament_package()
```

- package.xml:Ce fichier contient des métadonnées sur le package, telles que son nom, sa version, sa description, son mainteneur, sa licence et ses dépendances. Il est utilisé par les outils ROS 2 pour gérer les packages et leurs dépendances.

```
<?xml version="1.0"?>
<?xml-model href="http://download.ros.org/schema/package_format3.xsd"
schematypens="http://www.w3.org/2001/XMLSchema"?>
<package format="3">
<name>arduinobot_bringup</name>
<version>0.0.0</version>
<description>TODO: Package description</description>
<maintainer email="newtonkaris45@gmail.com">newton</maintainer>
<license>TODO: License declaration</license>
<buildtool_depend>ament_cmake</buildtool_depend>
<depend>rclcpp</depend>
<depend>ros2launch</depend>
<test_depend>ament_lint_auto</test_depend>
```

<test_depend>ament_lint_common</test_depend>
<export>
<build_type>ament_cmake</build_type>
</export>
</package>

b. Package arduinobot_description :

On prend comme example le package arduinobot_description Ce package est structuré comme suit.

```
arduinobot_description/
├── launch# Configuration des lancements
│   ├── arduinobot_display.launch.py
│   ├── arduinobot_display.launch.xml
│   └──arduinobot_gazebo.launch.py
├── meshes# Assets visuels 3D
│   ├── base_plate.STL
│   └──……
│   ├──rviz_config# Paramètres de visualisation
│   ├──rviz_sim.rviz
│   ├──urdf# Modélisation robotique
│   ├──arduinobot_camera.xacro
│   ├──arduinobot_control.xacro
│   ├──arduinobot_gazebo.xacro
│   └──arduinobot.xacro
├──worlds# Environnements simulés
│   ├──arduinobot.world
├──# arduinobot_description.md
├──CMakeLists.txt# Configuration de compilation
└──package.xml# Définition du package
```

- **Détail des Composants :**

a) Launch (Système d'exécution) :

Fichier	Type	Rôle Principal	Technologies Associées
arduinobot_display.launch.py	Python	Visualisation RViz	RViz2, Robot State Publisher
arduinobot_display.launch.xml	XML	Alternative historique	(Compatibilité ROS1)
arduinobot_gazebo.launch.py	Python	Intégration Gazebo	Gazebo ROS, Contrôleurs

b) Meshes (Représentation 3D) :

- **Format** : STL (Standard Triangle Language)
- **Utilisation** :
  - Visualisation réaliste
  - Collision en simulation
- **Exemple** :
  - base_plate.STL → Pièce maitresse du châssis

c) RViz Config (Visualisation) :

- rviz_sim.rviz :
  - Pré-configuration des vues
  - Topics par défaut
  - Styles visuels

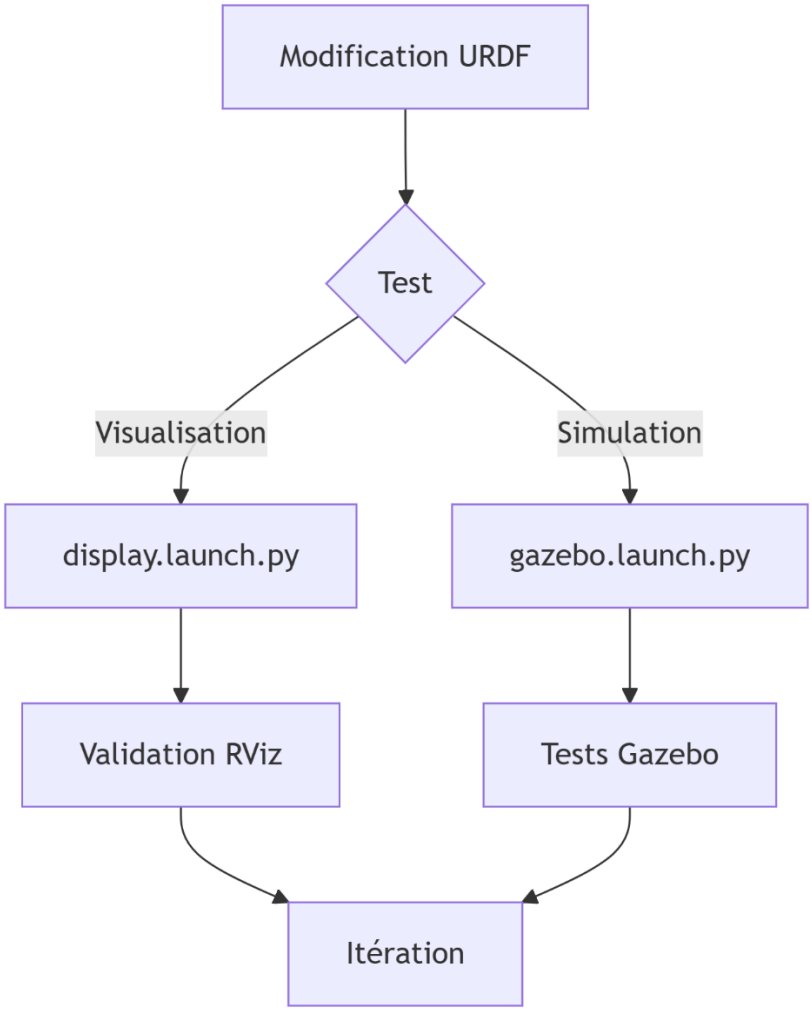
d) URDF/Xacro (Modélisation Robot) :

Fichier	Niveau	Description
arduinobot.xacro	Mécanique	Structure physique de base
arduinobot_gazebo.xacro	Simulation	Propriétés physiques (densité, friction)
arduinobot_control.xacro	Commande	Interfaces de contrôle
arduinobot_camera.xacro	Capteurs	Configuration des périphériques

e) Worlds (Environnements) :

- arduinobot.world :
    - Scénario de test
    - Objets interactifs
    - Conditions d'éclairage

- **Workflow de Développement :**



- **Fichiers Techniques :**

Fichier	Rôle
CMakeLists.txt	**- Déclaration des dépendances** **- Installation des ressources**
package.xml	**- Métadonnées du package** **- Dépendances système**

IV.3.2.2 Structure d'un Package Python ROS 2 :

my_python_package/

├── package.xml

├── setup.py

├── setup.cfg

├── resource/

│ └── my_python_package

├── my_python_package/

│ ├── __init__.py

│ └── my_node.py

├── launch/

│ └── my_launch_file.py

├── config/

│ └── params.yaml

├── test/

│ └── test_my_node.py

└── README.md

Détails des éléments :

- **package.xml** : Fichier de métadonnées du package (nom, version, dépendances, etc.).
- **setup.py** : Fichier principal pour l'installation du package avec setuptools (définit les scripts à installer comme nœuds ROS 2).
- **setup.cfg** : Configuration complémentaire pour setuptools.

- **resource/** : Doit contenir un fichier vide nommé comme le package (nécessaire pour l'indexation par ROS 2).
- **my_python_package/** : Le module Python contenant le code du nœud.
- **__init__.py** : Indique que ce dossier est un module Python.
- **my_node.py** : Fichier contenant le nœud ROS 2 (par exemple avec rclpy).
- **launch/** : Fichiers de lancement (.py) pour démarrer les nœuds et charger les paramètres.
- **config/** : Fichiers de configuration (comme params.yaml).
- **test/** : Tests unitaires, souvent avec pytest.
- **README.md** : Documentation du package.

IV.4 Créer les fichiers de type launch :

Les fichiers de lancement ROS 2 écrits en Python jouent un rôle fondamental dans la gestion des projets robotiques complexes.
Ils sont utilisés pour lancer plusieurs systèmes ou sous-systèmes simultanément, ce qui facilite la coordination entre différents composants logiciels du robot.

Grâce à ces fichiers, il est possible de démarrer automatiquement plusieurs nœuds ROS 2, de configurer leurs paramètres, et d'orchestrer leur fonctionnement sans intervention manuelle répétée.
Cela simplifie le développement, le test, et le déploiement des architectures robotiques modulaires.

IV.4.1 Explication de fichier : arduinobot_sim.launch.py :

Il lance trois parties principales pour le robot Arduinobot :
-Une simulation dans Gazebo : arduinobot_gazebo.launch.py
- Les contrôleurs du robot : arduinobot_controllers.launch.py
-Le système de planification de mouvement MoveIt2 : arduinobot_moveit.launch.py

```
import os
from launch import LaunchDescription
```

```
fromlaunch_ros.actions import Node
fromlaunch.actions import IncludeLaunchDescription
fromlaunch.launch_description_sources import PythonLaunchDescriptionSource
fromament_index_python.packages import get_package_share_directory
defgenerate_launch_description():
    gazebo = IncludeLaunchDescription(
        PythonLaunchDescriptionSource(
            os.path.join(get_package_share_directory("arduinobot_description"), "launch",
"arduinobot_gazebo.launch.py")
        )
    )
    controllers = IncludeLaunchDescription(
        PythonLaunchDescriptionSource(
            os.path.join(get_package_share_directory("arduinobot_control"), "launch",
"arduinobot_controllers.launch.py")
        )
    )
    moveit = IncludeLaunchDescription(
        PythonLaunchDescriptionSource(
            os.path.join(get_package_share_directory("arduinobot_moveit2"), "launch",
"arduinobot_moveit.launch.py")
        )
    )
    returnLaunchDescription([
        gazebo,
        controllers,
        moveit
    ])
```

- **Importation des bibliothèques**

import os : Utilisé pour manipuler les chemins de fichiers (comme os.path.join)

from launch import LaunchDescription :Représente la description complète du lancement, retournée par la fonction generate_launch_description()

- **Le cœur du programme :**

Cette fonction génère le fichier de lancement et retourne une description du lancement (une liste d'actions). Elle permet à ROS 2 de savoir quelles actions doivent être effectuées lorsque le fichier de lancement est exécuté.

```
defgenerate_launch_description():
```

Ajouter les fichiers de lancement avec IncludeLaunchDescription :

- **La fontionIncludeLaunchDescription**

Gazebo est un simulateur de robot utilisé pour tester des robots dans un environnement virtuel. Ici, ce bloc indique à ROS 2 d'inclure et d'exécuter le fichier de lancement arduinobot_gazebo.launch.py qui initialise la simulation Gazebo du robot.
get_package_share_directory("arduinobot_description") permet de récupérer le chemin vers le dossier du package arduinobot_description dans lequel se trouve le fichier de lancement.
os.path.join(...) permet de construire un chemin de fichier correct.

```
gazebo = IncludeLaunchDescription(
PythonLaunchDescriptionSource(
os.path.join(get_package_share_directory("arduinobot_description"),"launch",
"arduinobot_gazebo.launch.py")
) )
```

- **Lancer les contrôleurs du robot :**

Cette partie lance les contrôleurs du robot (probablement des contrôleurs de jointures ou de moteurs). Le fichier arduinobot_controllers.launch.py contient la configuration des contrôleurs, et il est inclus ici pour être lancé.
Comme pour Gazebo, on utilise get_package_share_directory pour trouver le fichier de lancement dans le package arduinobot_control.

```
controllers = IncludeLaunchDescription(
PythonLaunchDescriptionSource(
os.path.join(get_package_share_directory("arduinobot_control"),"launch",
"arduinobot_controllers.launch.py")
) )
```

- **Lancer MoveIt 2 (Planification de mouvement) :**

MoveIt 2 est utilisé pour la planification et l'exécution des mouvements du robot. Il génère des trajectoires pour le robot en fonction de son environnement et des commandes envoyées. Ce bloc lance le fichier arduinobot_moveit.launch.py.

Ce fichier est situé dans le package arduinobot_moveit2.

```
moveit = IncludeLaunchDescription(
PythonLaunchDescriptionSource(
os.path.join(get_package_share_directory("arduinobot_moveit2"),"launch",
"arduinobot_moveit.launch.py")
) )
```

- **Retour de la description du lancement :**

```
return LaunchDescription([ gazebo, controllers,moveit])
```

la fonction retourne une LaunchDescription qui contient la liste des processus à exécuter. Ici, il s'agit de trois processus :

- Lancer Gazebo.
- Lancer les contrôleurs du robot.
- Lancer MoveIt2.

IV.4.2 **Explication de fichier :** arduinobot_gazebo.launch.py

un fichier de lancement peut être utilisé pour lire un fichier XACRO, qui contient la description paramétrique du robot.
Ensuite, ce fichier est transformé en URDF pour être utilisé dans la simulation.
Le même fichier de lancement peut ensuite lancer le nœud Gazebo (spawnentity) afin d'insérer le robot dans l'environnement simulé.
Enfin, pour diffuser l'état des articulations du robot à travers le système ROS, on lance le nœud robot_state_publisher, ce qui permet aux autres composants (comme RViz ou MoveIt) de visualiser et utiliser la configuration actuelle du robot.

```
import os
from launch import LaunchDescription
fromlaunch_ros.actions import Node
fromlaunch.actions import DeclareLaunchArgument, IncludeLaunchDescription
fromlaunch.substitutions import Command, LaunchConfiguration
fromlaunch_ros.parameter_descriptions import ParameterValue
fromlaunch.launch_description_sources import PythonLaunchDescriptionSource
fromament_index_python.packages import get_package_share_directory, get_package_prefix
description_pkg = "arduinobot_description"
```

```
xacro_filename = "arduinobot.xacro"
defgenerate_launch_description():
   xacro_file = os
   model_args = DeclareLaunchArgument(
       name = "model",
       default_value = xacro_file,
       description = "Absolue path to robot urdf"
   )
os.environ["GAZEBO_MODEL_PATH"] =
os.path.join(get_package_prefix("arduinobot_description"), "share")
robot_description = ParameterValue(Command(
 ['xacro ', LaunchConfiguration("model")]
  ))
 robot_state_publisher = Node(
       package="robot_state_publisher",
       executable="robot_state_publisher",
       name="robot_state_publisher",
       output="screen",
       parameters=[{"robot_description":robot_description}],
       arguments=[xacro_file],  )
   gazebo = IncludeLaunchDescription(
  PythonLaunchDescriptionSource(
os.path.join(get_package_share_directory("gazebo_ros"), "launch", "gazebo.launch.py")
 ))
 spawn_entity = Node(
       package="gazebo_ros",
       executable="spawn_entity.py",
       arguments=["-entity", "arduinobot", "-topic", "robot_description"],
       output="screen",
   )
   returnLaunchDescription([
       model_args,
       robot_state_publisher,
       gazebo,
```

```
        spawn_entity,
    ])
```

- **Définition du fichier XACRO :**

Tu précises le nom de la package contenant les fichiers de description du robot. Le fichier arduinobot.xacro est un fichier XACRO, une version extensible de URDF.

```
description_pkg = "arduinobot_description"
xacro_filename = "arduinobot.xacro"
```

- **Récupération du chemin absolu vers le fichier XACRO :**

Cette ligne construit le chemin absolu vers le fichier XACRO situé dans arduinobot_description/urdf.

```
xacro_file = os.path.join(get_package_share_directory(description_pkg), 'urdf', xacro_filename)
```

- **Argument de lancement :**

Ici tu déclares un argument model permettant de passer un autre fichier URDF ou XACRO depuis la ligne de commande si on le souhaite. Par défaut, il utilisera le xacro_file défini plus haut.

```
model_args = DeclareLaunchArgument(
name = "model",
default_value = xacro_file,
description = "Absolue path to robot urdf"
)
```

- **Définir la variable d'environnement GAZEBO_MODEL_PATH :**

ROS 2 et Gazebo ont besoin de savoir où chercher les modèles.

Tu ajoutes ici le répertoire share de ton package comme chemin où Gazebo peut trouver le modèle.

```
os.environ["GAZEBO_MODEL_PATH"]=os.path.join(get_package_prefix("arduinobot_description"), "share")
```

- **Création du model gazebo à partir de fichier XACRO:**

Dans ROS 2, les fichiers XACRO (XML Macros) permettent de créer des descriptions robotiques modulaires et paramétrables. Pour simuler ce robot dans Gazebo, le fichier XACRO est d'abord transformé en URDF, puis inséré dans l'environnement simulé.

```
robot_description = ParameterValue(Command(
```

['xacro ', LaunchConfiguration("model")]))

.

Il prend le fichier passé à model (par défaut arduinobot.xacro) et le transforme en un fichier URDF que ROS peut utiliser.

- **Lancer robot_state_publisher :**

Ce nœud est responsable de publier les transformations (TF) entre les différentes parties du robot selon le fichier URDF.

Il prend en entrée la robot_description générée par xacro.

```
robot_state_publisher = Node(
package="robot_state_publisher",
executable="robot_state_publisher",
name="robot_state_publisher",
output="screen",
parameters=[{"robot_description":robot_description}],
arguments=[xacro_file],
)
```

- **Lancer le noeud spawn_entity.py du package gazebo_ros :**

Ce bloc démarre le simulateur Gazebo avec les paramètres du robot. Ce nœud fait apparaître le robot dans la simulation Gazebo.Il utilise le nom d'entité "arduinobot" et récupère la description depuis le topic robot_description.

```
   spawn_entity = Node(
        package="gazebo_ros",
        executable="spawn_entity.py",
        arguments=["-entity", "arduinobot", "-topic", "robot_description"],
        output="screen",
    )
```

IV .4.3 Gazebo et le Fichier xacro crée dans le package arduinobot_description :

Gazebo est un puissant simulateur 3D utilisé en robotique. Il permet de simuler avec précision le comportement des robots dans divers environnements. Gazebo prend en compte la dynamique, la friction, la gravité et les interactions avec les objets, ce qui le rend idéal pour tester des algorithmes de robotique.

Un fichier `.xacro` est un fichier XML macro utilisé pour générer un fichier URDF plus lisible et modulaire, souvent pour modéliser un robot.

Principaux Éléments du Fichier arduinobot.xacro:

- **Déclaration XML :**

<?xml version="1.0" encoding="UTF-8"?>

<robotxmlns:xacro="http://www.ros.org/wiki/xacro" name="arduinobot">

Déclaration XML : Cette ligne déclare que le fichier est au format XML et utilise le codage UTF-8. Tag <robot> : Cela définit l'élément principal du robot. Le robot est nommé arduinobot, et la bibliothèque XACRO est utilisée pour rendre le code plus modulaire et réutilisable (via l'élément xmlns:xacro).

- **Définition des Propriétés :**

<xacro:propertyname="PI" value="3.14159265359"/>

<xacro:propertyname="velocity" value="30.0"/>

<xacro:propertyname="effort" value="10.0"/>

Propriétés : Ici, des variables globales sont définies :

PI : La valeur de Pi, utilisée pour les calculs de rotation.

velocity : La vitesse maximale des joints (en rad/s).

effort : L'effort maximal que chaque joint peut appliquer (en Newton-mètre). Ces propriétés peuvent être réutilisées dans tout le fichier.

- **Macros XACRO :**

Les macros XACRO sont utilisées pour simplifier la réutilisation de certaines sections de code. **3.1 Inertie :**

<xacro:macroname="default_inertial" params="mass">

<inertial>

<mass value="${mass}"/>

<inertiaixx="1.0" ixy="0.0" ixz="0.0" iyy="1.0" iyz="0.0" izz="1.0"/>

</inertial>

</xacro:macro>

Cette macro permet de définir les propriétés d'inertie pour un lien (link). Elle prend en paramètre mass (la masse du lien) et applique des valeurs par défaut pour les moments d'inertie (ixx, iyy, izz).

- **Transmission :**

<xacro:macroname="default_transmission" params="joint">

<transmissionname="${joint}_transmission"><type>transmission_interface/SimpleTransmission</type>
<jointname="${joint}" role = "joint_1">
<mechanicalReduction>1.0</mechanicalReduction>
</joint>
<actuatorname="${joint}_motor" role = "actuator_1"/>
</transmission>
</xacro:macro>

Cette macro définit la transmission entre un joint et un moteur. Elle prend en paramètre joint, et pour chaque joint, crée une transmission simple qui lie le moteur au joint avec un rapport de réduction mécanique de 1.0.

- **Inclusions de Fichiers :**

<xacro:includefilename="$(findarduinobot_description)/urdf/arduinobot_camera.xacro"/><xacro:includefilename="$(findarduinobot_description)/urdf/arduinobot_gazebo.xacro"/><xacro:includefilename="$(findarduinobot_description)/urdf/arduinobot_control.xacro"/>Ces lignes incluent des fichiers externes XACRO pour ajouter des configurations supplémentaires pour la caméra, le modèle Gazebo, et le contrôle ROS2. Cela permet de séparer le code en plusieurs fichiers et de mieux organiser le projet.

- **Définition des Liens (Links) :**

Les liens représentent les différentes parties du robot.

- **World Link**

<linkname="world"/>

Cela définit le lien "monde", qui est souvent utilisé comme le cadre de référence pour l'ensemble du robot.

- **Base Link**

<linkname="base_link">
<xacro:default_inertial mass = "1.0"/>
<visual>
<originxyz="-0.5 -0.5 0.0" rpy="0.0 0.0 0.0"/>
<geometry>
<meshfilename="package://arduinobot_description/meshes/basement.STL" scale="0.01 0.01 0.01"/>
</geometry>

</visual>
<collision>
<originxyz="-0.5 -0.5 0.0" rpy="0.0 0.0 0.0"/>
<geometry>
<meshfilename="package://arduinobot_description/meshes/basement.STL" scale="0.01 0.01 0.01"/>
</geometry>
</collision>
</link>

Base Link : Le lien de base est la fondation du robot. Il représente le corps principal et contient la géométrie visuelle et de collision, qui est définie à partir d'un fichier STL (ici, basement.STL).

Masse et Inertie : Il utilise la macro default_inertial pour définir une masse de 1.0 kg. 5.3 Autres Liens Les autres liens comme base_plate, forward_drive_arm, horizontal_arm, etc., suivent une structure similaire, définissant la géométrie et les collisions à partir de fichiers STL et la masse avec la macro d'inertie.

- **Définition des Joints :**

Les joints connectent les liens entre eux. Chaque joint a un type (révolute, fixe, etc.) et définit comment deux liens peuvent se déplacer les uns par rapport aux autres.

- **Joint Virtuel :**

<jointname="virtual_joint" type="fixed">
<parentlink="world"/>
<childlink="base_link"/>
<originxyz="0.0 0.0 0.0" />
</joint>

Un joint fixe entre le "world" et le "base_link", cela positionne le robot dans l'espace sans mouvement.

- **Joints Révolutifs :**

<jointname="joint_1" type="revolute">
<parentlink="base_link"/>
<childlink="base_plate"/>
<originxyz="0.0 0.0 0.307" rpy="0.0 0.0 0.0"/>
<axisxyz="0.0 0.0 1.0"/>

<limit effort="${effort}" lower="-${PI/2}" upper="${PI/2}"
velocity="${velocity}"/></joint>

Ce joint permet une rotation autour d'un axe spécifié (ici l'axe Z avec axis xyz="0.0 0.0 1.0"). Il a des limites de rotation (de -π/2 à π/2) et une vitesse/effort maximale.

Les autres joints suivent une structure similaire avec différents axes et limites de mouvement pour les bras du robot et la pince.

- **Capteurs et Caméra :**

<linkname="rgb_camera">
<xacro:default_inertial mass="0.001"/>
<visual>
<originxyz="-0.08 0.13 0.13" rpy="${-PI/2} 0 ${-PI/2}"/>
<geometry>
<meshfilename = "package://arduinobot_description/meshes/pi_camera.STL" scale = "0.01 0.01 0.01"/>
</geometry>
</visual>
<collision>
<originxyz="-0.08 0.13 0.13" rpy="${-PI/2} 0 ${-PI/2}"/>
<geometry>
<meshfilename = "package://arduinobot_description/meshes/pi_camera.STL" scale = "0.01 0.01 0.01"/>
</geometry>
</collision>
</link>

Le robot possède un capteur de caméra RGB, attaché à un joint fixe qui le relie au lien base_link.

- **Transmissions :**

Les transmissions définissent la relation entre les joints et les moteurs qui les contrôlent.

<xacro:default_transmission joint = "joint_1"/>
<xacro:default_transmission joint = "joint_2"/>
<xacro:default_transmission joint = "joint_3"/>
<xacro:default_transmission joint = "claw_support_to_right_gripper_finger"/>

Ces lignes ajoutent des transmissions pour chaque joint, permettant de contrôler chaque joint avec un moteur.

Principaux Éléments du Fichier arduinobot_camera.xacro inclut dans arduinobot.xacro:

Ce fichier permet une intégration plug-and-play de la caméra dans le système global, avec une configuration cohérente pour la simulation et le déploiement réel.

```
xml version="1.0" encoding="UTF-8"?>
<robotxmlns:xacro="http://www.ros.org/wiki/xacro" name = "arduinobot">
<gazeboreference="rgb_camera">
<sensor type="camera" name="rgb_camera">
<always_on>1</always_on>
<pose>0 0 0 0 0 0</pose>
<visualize>true</visualize>
<update_rate>30.0</update_rate>
<cameraname="rgb_camera">
<horizontal_fov>1.15</horizontal_fov>
<vertical_fov>0.71</vertical_fov>
<image>
<width>2304</width>
<height>1296</height>
<format>R8G8B8</format>
</image>
<clip>
<near>0.05</near>
<far>1.0</far>
</clip>
</camera>
<pluginname="camera_controller" filename="libgazebo_ros_camera.so">
<alwaysOn>true</alwaysOn>
<updateRate>0.0</updateRate>
<cameraName>rgb_cam</cameraName>
<imageTopicName>image_raw</imageTopicName>
<cameraInfoTopicName>camera_info</cameraInfoTopicName>
<frameName>rgb_camera</frameName>
<hackBaseline>0.07</hackBaseline>
<distortionK1>0.0</distortionK1>
```

<distortionK2>0.0</distortionK2>

<distortionK3>0.0</distortionK3>

<distortionT1>0.0</distortionT1>

<distortionT2>0.0</distortionT2>

</plugin>

</sensor>

</gazebo>

</robot>

Explication détaillée du plugin libgazebo_ros_camera.so

Le plugin est le composant central qui permet l'intégration entre Gazebo et ROS 2. Voici ses caractéristiques techniques approfondies :

1. Paramètres fondamentaux

<pluginname="camera_controller" filename="libgazebo_ros_camera.so">

<alwaysOn>true</alwaysOn><!-- *Maintient le capteur actif même sans subscribers* -->

<updateRate>0.0</updateRate><!-- *0.0 = maximum possible (~1kHz)* -->

<cameraName>rgb_cam</cameraName><!--*Namespace ROS* -->

2. Configuration des topics

Paramètre XML	**Topic ROS généré**	**Type de message**
<imageTopicName>	/rgb_cam/image_raw	sensor_msgs/Image
<cameraInfoTopicName>	/rgb_cam/camera_info	sensor_msgs/CameraInfo

3. Calibration avancée

<hackBaseline>0.07</hackBaseline><!-- *Nécessaire pour les setups stéréo* -->

<distortionK1>0.0</distortionK1><!-- *Coefficient de distorsion radiale 1* -->

<distortionK2>0.0</distortionK2><!-- *Coefficient de distorsion radiale 2* -->

<distortionK3>0.0</distortionK3><!-- *Coefficient de distorsion radiale 3* -->

<distortionT1>0.0</distortionT1><!-- *Coefficient de tangage 1* -->

<distortionT2>0.0</distortionT2><!-- *Coefficient de tangage 2* -->

4. Optimisations critiques

- **Performances** :

<updateRate>0.0</updateRate><!-- *Priorité au taux de rafraîchissement maximal* -->

- **Gestion mémoire** :

```
<image>
<width>2304</width><!-- Doit correspondre aux besoins réels -->
<height>1296</height>
</image>
```

5. Bonnes pratiques d'implémentation

1. **Pour les caméras haute résolution** :

```
<updateRate>15.0</updateRate><!-- Réduire pour 4K+ -->
```

2. **Environnements complexes** :

```
<clip>
<near>0.1</near><!-- Augmenter pour éviter le bruit proche -->
<far>100.0</far><!-- Étendre pour les grands espaces -->
</clip>
```

3. **Débogage actif** :

```
ros2 topic hz /rgb_cam/image_raw
ros2 run rqt_image_viewrqt_image_view
```

6. Extensions possibles

- **Ajout de compression** :

```
<imageTopicName>image/compressed</imageTopicName>
```

- **Intégration Depth** :

```
<depthCamera>true</depthCamera>
<pointCloudTopicName>points</pointCloudTopicName>
```

Principaux Éléments du Fichier arduinobot_gazebo.xacro inclut dans arduinobot.xacro:

Ce fichier configure l'intégration entre Gazebo et ROS 2 Control pour la simulation physique du robot. Voici une analyse détaillée :

```
xml version="1.0" encoding="UTF-8"?>
<robotxmlns:xacro="http://www.ros.org/wiki/xacro" name="arduinobot">
  <gazebo>
    <pluginname="gazebo_ros2_control" filename="libgazebo_ros2_control.so">
      <robot_param>robot_description</robot_param>
      <robot_param_node>robot_state_publisher</robot_param_node>
```

```
        <parameters>$(find
arduinobot_control)/config/controller_manager_config.yaml</parameters>
      </plugin>
   </gazebo>
</robot>
```

- **Structure de Base :**

```
<?xml version="1.0" encoding="UTF-8"?>
<robotxmlns:xacro="http://www.ros.org/wiki/xacro" name="arduinobot">
```

- **NamespaceXacro** : Permet d'utiliser des macros et variables
- **Nom du robot** : Doit correspondre au nom déclaré dans arduinobot.xacro

2. Plugin Gazebo ROS 2 Control (Coeur du système) :

```
<gazebo>
<pluginname="gazebo_ros2_control" filename="libgazebo_ros2_control.so">
<robot_param>robot_description</robot_param>
<robot_param_node>robot_state_publisher</robot_param_node>
<parameters>$(find
arduinobot_control)/config/controller_manager_config.yaml</parameters>
</plugin>
</gazebo>
```

3. Paramètres Clés du Plugin :

Paramètre	Valeur/Chemin	Description
robot_param	robot_description	Nom du paramètre contenant l'URDF
robot_param_node	robot_state_publisher	Nœud qui publie la description
parameters	(find arduinobot_control)/config/controller_manager_config.yaml	Fichier de configuration descontrôleus

4. Fonctionnement Global

1. **Chargement** : Gazebo charge le plugin au démarrage
2. **Initialisation** :
    - Récupère la description URDF via robot_description
    - Configure les contrôleurs via le fichier YAML
3. **Exécution** :
    - Synchronise la simulation Gazebo avec ROS 2 Control
    - Gère les interfaces matérielles simulées

5. Fichier de Configuration Associé

Exemple typique de controller_manager_config.yaml :

```
controller_manager:
ros__parameters:
update_rate:100  # Hz
joint_state_broadcaster:
type:joint_state_broadcaster/JointStateBroadcaster
arm_controller:
type:position_controllers/JointTrajectoryController
joints:
      - joint1
      - joint2
```

6. Bonnes Pratiques

- **Performance** : Ajuster update_rate selon la complexité du robot
- **Modularité** : Séparer les contrôleurs par sous-systèmes (base, bras, etc.)
- **Débogage** :

```
ros2 control list_controllers
ros2 control list_hardware_interfaces
```

7. Intégration dans arduinobot.xacro

```
<xacro:includefilename="$(findarduinobot_description)/urdf/arduinobot_gazebo.xacro"/>
```

8. Extensions Possibles

- **Capteurs Simulés** : Ajouter des plugins pour IMU/Lidar
- **Mode Hybride** : Interface avec du vrai matériel via ros2_control

Principaux Éléments du Fichier arduinobot_control.xacro inclut dans arduinobot.xacro:

```
xml version="1.0" encoding="UTF-8"?>
<robotxmlns:xacro="http://www.ros.org/wiki/xacro" name="arduinobot">
  <ros2_control name = "ArduinobotSim" type="System">
    <hardware>
      <plugin>gazebo_ros2_control/GazeboSystem</plugin>
    </hardware>
    <jointname="joint_1">
      <command_interfacename = "position">
        <paramname = "min">-${PI}</param>
        <paramname = "max">${PI}</param>
      </command_interface>
      <state_interfacename = "position"/>
    </joint>
    <jointname = "joint_2">
      <command_interfacename = "position">
        <paramname = "min">-${PI}</param>
        <paramname = "max">${PI}</param>
      </command_interface>
      <state_interfacename = "position"/>
    </joint>
    <jointname = "joint_3">
      <command_interfacename = "position">
```

```
            <paramname = "min">-${PI}</param>
            <paramname = "max">${PI}</param>
        </command_interface>
        <state_interfacename = "position"/>
    </joint>
    <jointname = "claw_support_to_right_gripper_finger">
        <command_interfacename = "position">
            <paramname = "min">-${PI}</param>
            <paramname = "max">0.0</param>
        </command_interface>
        <state_interfacename = "position"/>
    </joint>
    <jointname = "claw_support_to_left_gripper_finger">
        <paramname="mimic">claw_support_to_right_gripper_finger</param>
        <paramname="multiplier">-1</param>
        <command_interfacename = "position">
            <paramname = "min">0.0</param>
            <paramname = "max">${PI}</param>
        </command_interface>
    </joint>
  </ros2_control>
</robot>
```

Ce fichier configure les interfaces de contrôle matériel pour ROS 2 Control. Voici son analyse détaillée :

1. Structure de Base

```
<ros2_control name="ArduinobotSim" type="System">
```

- **Type System** : Indique un système complet (vs "Sensor" ou "Actuator")
- **Nom** : Doit correspondre à la référence dans les contrôleurs

2. Configuration Matérielle

```
<hardware>
<plugin>gazebo_ros2_control/GazeboSystem</plugin>
</hardware>
```

- **Plugin Gazebo** : Interface pour la simulation physique

- **Alternative réelle** : Remplacer par hardware_interface/GenericSystem pour du vrai hardware

3. Interfaces des Joints

Chaque joint possède :

- **Command Interface** : Pour envoyer des commandes
- **State Interface** : Pour lire l'état actuel

Exemple type :

```
<jointname="joint_1">
<command_interfacename="position">
<paramname="min">-${PI}</param><!-- -3.14 rad -->
<paramname="max">${PI}</param><!-- 3.14 rad -->
</command_interface>
<state_interfacename="position"/>
</joint>
```

4. Cas Spécial : Pince Mimétique

```
<jointname="claw_support_to_left_gripper_finger">
<paramname="mimic">claw_support_to_right_gripper_finger</param>
<paramname="multiplier">-1</param>
<!-- ... -->
</joint>
```

- **Mimic** : Crée un mouvement miroir inversé (-1) par rapport à la pince droite
- **Gain d'espace** : Évite d'avoir à créer un contrôleur séparé

5. Paramètres Clés

Paramètre	Valeur Typique	Description
command_interface	position	Mode contrôle (peut être velocity ou effort)
min/max	-π à +π	Limites physiques en radians
mimic	Nom du joint	Pour les mécanismes couplés

6. Bonnes Pratiques

1. **Consistance** : Les noms des joints doivent matcher exactement l'URDF
2. **Sécurité** : Toujours définir des limites min/max réalistes
3. **Performance** : Limiter le nombre d'interfaces au strict nécessaire

7. Intégration avec arduinobot.xacro

```
<xacro:includefilename="$(findarduinobot_description)/urdf/arduinobot_control.xacro"/>
```

8. Workflow Typique

1. Simulation :

```
ros2 launch arduinobot_description display.launch.py
```

2. Contrôle :

```
ros2 control load_controllerjoint_trajectory_controller
```

IV.4.4 Explication de fichier : arduinobot_controllers.launch.py

Le nœud spawner du package controller_manager permet de lancer dynamiquement des contrôleurs en spécifiant leur nom et différents paramètres de configuration. Pour lancer un contrôleur particulier (par exemple arm_controller), on utilise la syntaxe ROS 2 suivante :

```
import os
from launch import LaunchDescription
fromlaunch_ros.actions import Node
fromlaunch.actions import DeclareLaunchArgument
fromlaunch.substitutions import Command, LaunchConfiguration
fromlaunch_ros.parameter_descriptions import ParameterValue
fromament_index_python.packages import get_package_share_directory
defgenerate_launch_description():
    xacro_filename = "arduinobot.xacro"
    xacro_file = os.path.join(get_package_share_directory("arduinobot_description"), "urdf", xacro_filename)
    model_args = DeclareLaunchArgument(
        name = "model",
        default_value = xacro_file,
        description = "Absolue path to robot urdf"
    )
    robot_description = ParameterValue(
        Command(
            ['xacro ', LaunchConfiguration("model")]
        )
    )
    robot_state_publisher = Node(
        package="robot_state_publisher",
        executable="robot_state_publisher",
```

```
        name="robot_state_publisher",
        output="screen",
        parameters=[{"robot_description":robot_description}],
        arguments=[xacro_file],
    )
    joint_trajectory_controller = Node(
            package="controller_manager",
            executable="spawner",
            arguments=[
                "arm_controller",
                "--controller-manager", "/controller_manager",
            ]
    )
    joint_state_broadcaster = Node(
            package="controller_manager",
            executable="spawner",
            arguments=[
                "joint_state_broadcaster",
                "--controller-manager", "/controller_manager",
            ]
        )

    gripper_controller = Node(
            package="controller_manager",
            executable="spawner",
            arguments=[
                "gripper_controller",
                "--controller-manager", "/controller_manager",
            ]
    )
    returnLaunchDescription(
        [
            model_args,
            robot_state_publisher,
```

```
        joint_state_broadcaster,
        joint_trajectory_controller,
        gripper_controller,
    ]
  )
```

Lancer les contrôleurs avec spawner :

Se connecter au controller manager (qui doit déjà être lancé dans ton système).

Tu dois t'assurer qu'un nœud controller_manager tourne (souvent lancé via un autre launch file avec ros2_control_node).

Pour **lancer les contrôleurs avec `spawner` dans ROS 2 (avec `ros2_control`)**, voici les étapes typiques à suivre. Cela suppose que vous avez :

- Un fichier URDF ou XACRO qui charge un `ros2_control` hardware interface.
- Un contrôleur défini dans un fichier de configuration YAML.
- Le `controller_manager` actif via un `robot_state_publisher` et le chargement du robot.

```
joint_state_broadcaster = Node(
package="controller_manager",
 executable="spawner",
 arguments=[
"argument_name",
"--controller-manager", "/controller_manager",
 ] )
```

Selonl'argumentargument_name:

- arm _contoroller:
- joint_state_broadcaster
- gripper_controller

IV.4.5 Explication de fichier : arduinobot_moveit.launch.py

Dans cette configuration, **le bras est contrôlé par move_group**, le composant principal de MoveIt 2 chargé de la planification et de l'exécution des mouvements. Le lancement du système utilise quelques paramètres importants comme use_sim_time, qui

permet de synchroniser le temps avec la simulation dans Gazebo. De plus, le robot est **affiché dans RViz**, ce qui permet à l'utilisateur de visualiser sa position actuelle, ses mouvements planifiés, ainsi que les trajectoires prévues dans l'environnement 3D

```
import os
from launch import LaunchDescription
frommoveit_configs_utils import MoveItConfigsBuilder
fromlaunch_ros.actions import Node
fromlaunch.actions import DeclareLaunchArgument
fromlaunch.substitutions import LaunchConfiguration
fromament_index_python.packages import get_package_share_directory
defgenerate_launch_description():
  is_sim = LaunchConfiguration('is_sim')
  is_sim_arg = DeclareLaunchArgument(
      'is_sim',
      default_value='True'
  )
  moveit_config = (
      MoveItConfigsBuilder("arduinobot", package_name="arduinobot_moveit2")
      .robot_description(file_path=os.path.join(
          get_package_share_directory("arduinobot_description"),
          "urdf",
          "arduinobot.xacro"
          )
      )
      .robot_description_semantic(file_path="config/arduinobot.srdf")
      .trajectory_execution(file_path="config/moveit_controllers.yaml")
      .to_moveit_configs()
  )
  move_group_node = Node(
      package="moveit_ros_move_group",
      executable="move_group",
      output="screen",
      parameters=[moveit_config.to_dict(),
                  {'use_sim_time':is_sim},
```

```
                {'publish_robot_description_semantic':True}],
        arguments=["--ros-args", "--log-level", "info"],
    )
    rviz_config = os.path.join(
        get_package_share_directory("arduinobot_moveit2"),
            "config",
            "moveit.rviz",
    )
    rviz_node = Node(
        package="rviz2",
        executable="rviz2",
        name="rviz2",
        output="log",
        arguments=["-d", rviz_config],
        parameters=[
            moveit_config.robot_description,
            moveit_config.robot_description_semantic,
            moveit_config.robot_description_kinematics,
            moveit_config.joint_limits,
        ],
    )
    returnLaunchDescription(
        [
            is_sim_arg,
            move_group_node,
            rviz_node
        ]
    )
```

- **Déclaration d'un argument :use_sim_time**

L'option use_sim_time joue un rôle fondamental dans les projets robotiques simulés avec ROS 2. Elle permet d'indiquer à un nœud ROS s'il doit utiliser **l'horloge simulée** (généralement fournie par Gazebo) au lieu de l'horloge réelle du système.

Cela est particulièrement important pour garantir la **synchronisation temporelle** entre les différents nœuds pendant la simulation (comme les capteurs, les visualisations RViz, ou le contrôleur MoveIt).

```
is_sim = LaunchConfiguration('is_sim')
is_sim_arg = DeclareLaunchArgument(
'is_sim', default_value='True'
)
```

- **Configuration de MoveIt2:**

```
moveit_config = (
MoveItConfigsBuilder("arduinobot", package_name="arduinobot_moveit2")
.robot_description(file_path=os.path.join(
get_package_share_directory("arduinobot_description"),
"urdf",
 "arduinobot.xacro"
)
)
.robot_description_semantic(file_path="config/arduinobot.srdf")
.trajectory_execution(file_path="config/moveit_controllers.yaml")
.to_moveit_configs()
)
```

- **Charge tous les fichiers nécessaires :**

xacro → description du robot.

urdf → informations sémantiques (groupes de joints, liens d'EE, etc.).

moveit_controllers.yaml → quels contrôleurs MoveIt doit utiliser.

- **Lancer le nœudmove_group:**

Lance le serveur MoveIt 2, qui reçoit les requêtes de planification, de mouvement, etc. Il reçoit tous les paramètres du robot + une activation du temps simulé si nécessaire. ajoutes aussi un niveau de log "info".

```
move_group_node = Node(
 package="moveit_ros_move_group",
executable="move_group",
output="screen",
parameters=[moveit_config.to_dict(),
{'use_sim_time': is_sim},
```

```
{'publish_robot_description_semantic': True}],
 arguments=["--ros-args", "--log-level", "info"],
)
```

- **Lancer RViz2 avec MoveIt :**

Charges un fichier de configuration .rviz pour préconfigurer RViz avec ton robot et MoveIt.

```
rviz_config = os.path.join(
get_package_share_directory("arduinobot_moveit2"),
"config",
"moveit.rviz", )
```

Ce nœud lance RViz2 avec les bonnes descriptions et paramètres.

Il permet de voir ton robot, les contrôleurs, les trajectoires planifiées, etc.

```
rviz_node = Node(
package="rviz2",
executable="rviz2",
 name="rviz2", o
utput="log",
arguments=["-d", rviz_config],
parameters=[
moveit_config.robot_description,
moveit_config.robot_description_semantic,
moveit_config.robot_description_kinematics, moveit_config.joint_limits, ],
)
```

IV.5 Résultats de simulation :

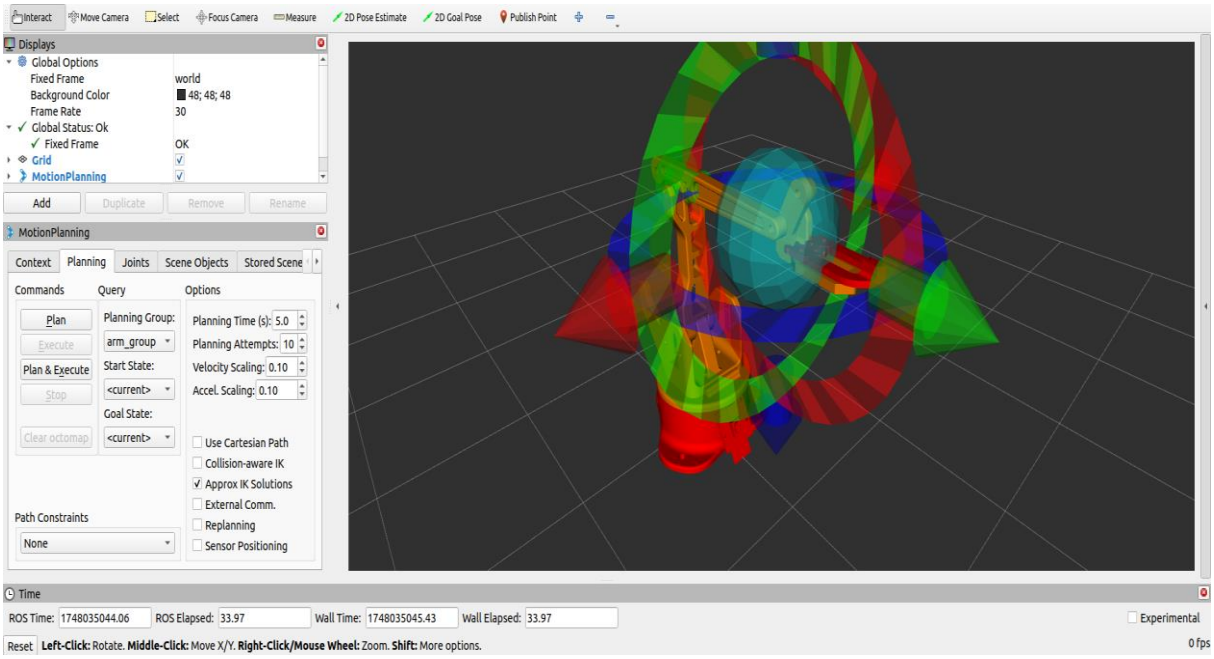
Pour simuler le robot, on utilise la fenêtre terminal de Ubuntu (ctrl+alt-T) et écrire les commandes suivantes :

cd my_ros2_ws

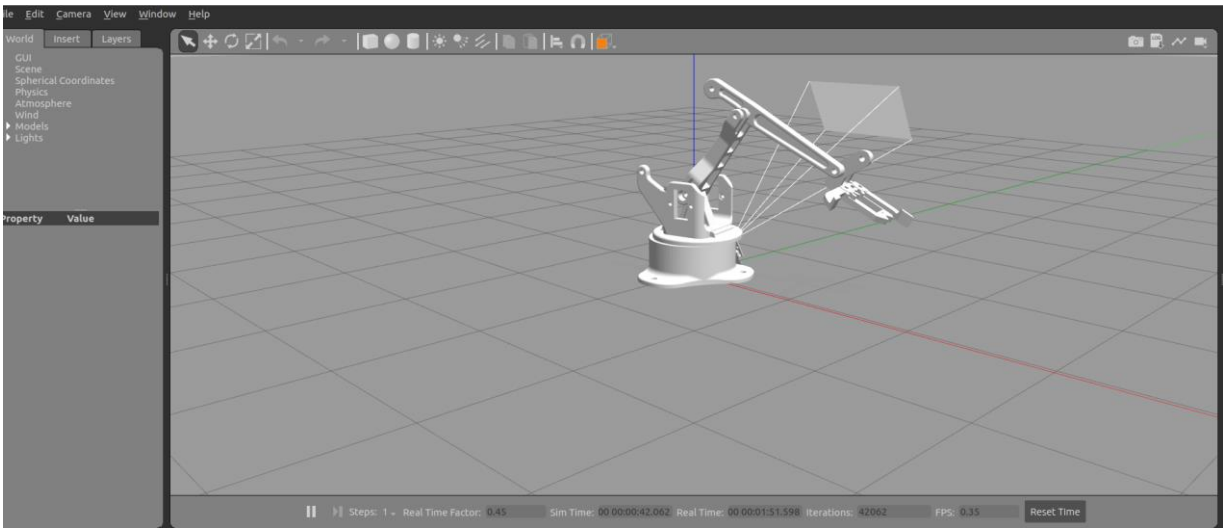
colcon build –symlink-install

source install/setup.bash

Ensuite on peut lancer les packaes :



Fegure31. Le bras sur RVIZ



Fegure32. Le bras sur Gazebo

IV. 6 Conclusion :

Dans ce chapitre, les concepts et théories abordés dans les chapitres précédents ont été appliqués dans un environnement de simulation avancé en utilisant ROS 2 et Gazebo, dans le but de concevoir un modèle réaliste et fonctionnel d'un bras robotisé.

Les capteurs et actionneurs ont été intégrés dans le modèle du robot à l'aide de fichiers XACRO, qui permettent de définir dynamiquement les composants du robot en utilisant la syntaxe URDF enrichie. Les capteurs (comme les caméras, IMU ou LiDAR) sont ajoutés

via des balises <gazebo> contenant des balises <sensor>, ce qui permet de simuler leur comportement dans l'environnement virtuel. Quant aux actionneurs, ils sont représentés par des joints associés à des plugins de contrôle comme gazebo_ros_control, permettant d'envoyer des commandes de mouvement aux articulations du bras.

Les résultats obtenus ont démontré une grande précision dans le contrôle ainsi qu'une interaction efficace avec les capteurs et les actionneurs, confirmant ainsi la pertinence de ces outils pour le développement de robots avant leur déploiement dans des environnements réels.

ROS 2 et Gazebo ont prouvé leur efficacité en tant que plateformes intégrées permettant la création de systèmes robotiques flexibles et évolutifs. Cette approche contribue à la réduction des coûts et des risques liés aux tests physiques, tout en améliorant la qualité globale du développement robotique

Conclusion générale:

Ce mémoire a pour objectif l'étude et la simulation d'un bras robotique en utilisant l'environnement ROS 2 et le simulateur Gazebo, dans le cadre du développement de systèmes robotiques efficaces pouvant être testés virtuellement avant toute implémentation réelle. Ce travail constitue une étape essentielle pour comprendre les mécanismes de contrôle logiciel et l'intégration physique des différents composants du bras robotique, notamment les capteurs et les actionneurs, dans un environnement de simulation open source.

La méthodologie adoptée repose sur l'analyse de l'environnement ROS 2 et de ses composants de base, l'étude de la structure mécanique du bras robotique, ainsi que la sélection et l'intégration des capteurs et actionneurs appropriés. Ensuite, un modèle de simulation réaliste a été mis en œuvre dans l'environnement Gazebo à l'aide des outils de ROS 2, avec un accent particulier sur le contrôle du mouvement et la réponse du bras aux entrées sensorielles.

Les résultats de la simulation ont démontré l'efficacité du système à exécuter les mouvements requis avec une précision acceptable. L'intégration entre ROS 2 et Gazebo a facilité la validation de divers scénarios opérationnels sans recourir à un prototype physique. Il a été conclu que l'utilisation de ces outils peut accélérer le développement des systèmes robotiques et réduire les coûts liés aux tests matériels.

Cette étude met en évidence l'importance de la simulation dans la conception et le contrôle, et recommande l'exploitation des environnements open source dans la recherche, l'enseignement et le développement industriel en robotique.

Bibliographie :

Chapitre 1 : Mise en place et contrôle du bras UR3 avec ROS2

1. Jarque, M. Set up and control of a UR3 Robot using ROS2 Humble. Rapport technique, 2023.

2. Ebin, R. A Concise Introduction to Robot Programming with ROS2. Document local, sans date.

3. Introduction à ROS – ROS_Robot_Operating_System_Introduction_FR.pdf, document interne/local.

4. rviz2 GitHub Repository. Disponible sur : https://github.com/ros2/rviz

5. Documentation officielle ROS 2 Humble. Disponible sur : https://docs.ros.org/en/humble/index.html

Chapitre 2 : Modélisation et commande des robots

1. Siciliano, B., Sciavicco, L., Villani, L., & Oriolo, G. Robotics: Modelling, Planning and Control. Springer.

2. Craig, J. J. Introduction to Robotics: Mechanics and Control. Pearson.

3. Corke, P. Robotics, Vision and Control: Fundamental Algorithms in MATLAB. Springer

4. Spong, M. W., Hutchinson, S., & Vidyasagar, M. (2020). Robot Modeling and Control (2nd ed.). Wiley.

5. Kurfess, T. R. (2005). Robotics and Automation Handbook. CRC Press.

6. Siciliano, B., & Khatib, O. (Eds.). (2020). Springer Handbook of Robotics (2nd ed.). Springer.

Chapitre 3 : Systèmes autonomes et perception

1. Siciliano, B., & Khatib, O. (Eds.). (2016). Springer Handbook of Robotics (2nd ed.). Springer. ISBN: 978-3-319-32552-1

2. Correll, N., Hayes, B., et al. Introduction to Autonomous Robots: Mechanisms, Sensors, Actuators, and Algorithms.

3.Klafter, R. D., Chmielewski, T. A., & Negin, M. Robotics: Control, Sensing, Vision, and Intelligence.

4. Fu, K. S., Gonzalez, R. C., & Lee, C. S. G. Robotics: Control, Sensing, Vision, and Intelligence.

5. Nikolaus K. S., Chien, G. F. E. L. (Eds.). Introduction to Autonomous Robots: Mechanisms, Sensors, Actuators, and Algorithms.

Chapitre 4 : Ressources en ligne et référentiels

1. ROS 2 GitHub Repository – https://github.com/ros2