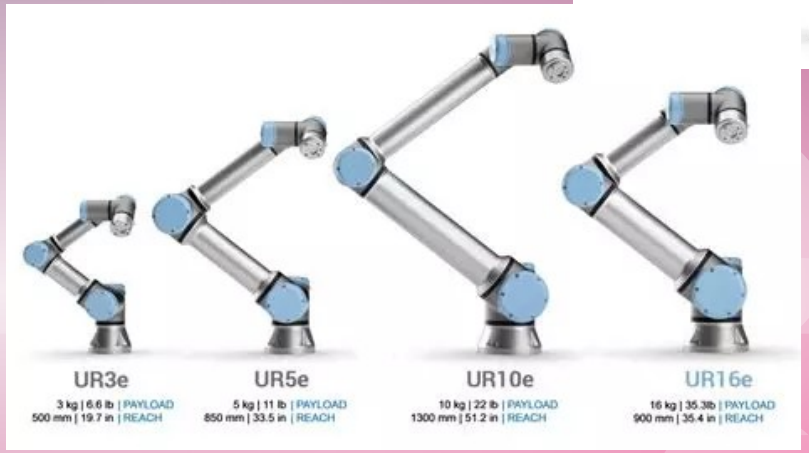
LA ROBOTIQUE DE MANIPULATION

Les bras robotiques



Les défis du contrôle de robotique

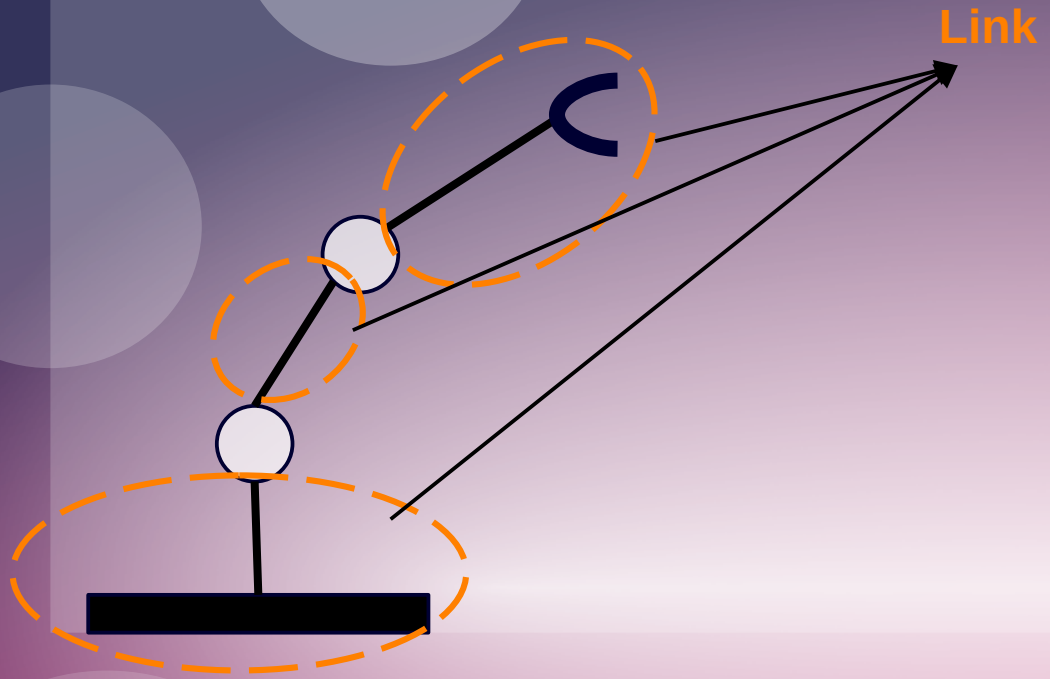
Le contrôle en robotique

1) Il faut savoir à quoi ressemble notre robot
2) Il faut pouvoir récupérer les informations des moteurs et envoyer des commandes
3) Il faut planifier notre action en prenant en compte la tâche et l'environnement.

Le format URDF

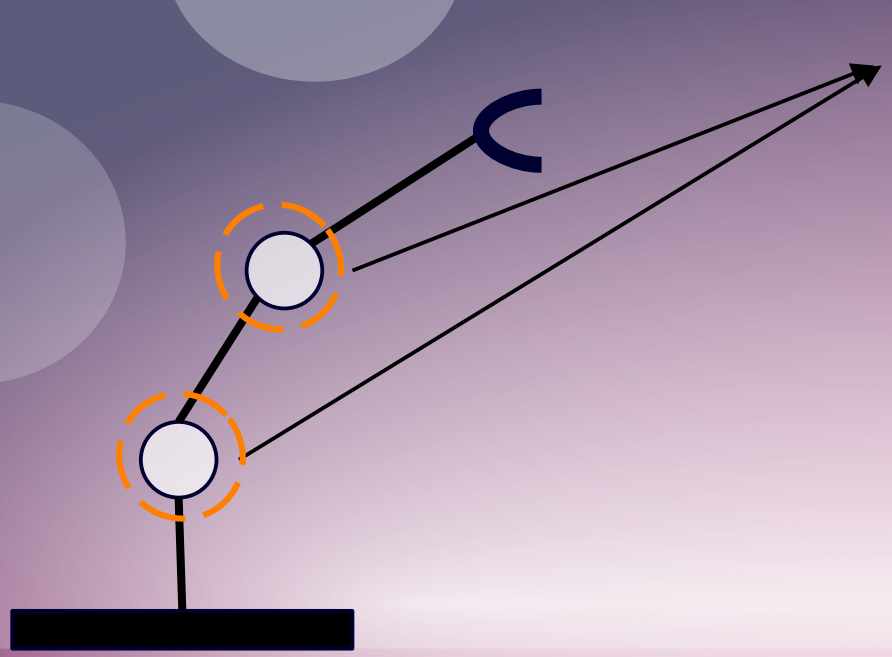
- Format XML de description unifié de robots
- Pourquoi faire ? Besoin d'un modèle du robot pour :
  - Calculer la cinématique, des chemins, collisions ...
- Un fichier URDF = un robot :
  - Liste de joints : linéaires, circulaires, continus
  - Liste de liens qui connectent les joints : base, jambe, tibia … :
- Un modèle visuel (mesh ou forme primitive)
- Un modèle de collision (mesh ou forme primitive)
- URDF paramétrable avec Xacro : variantes, options ...

Le format URDF : link



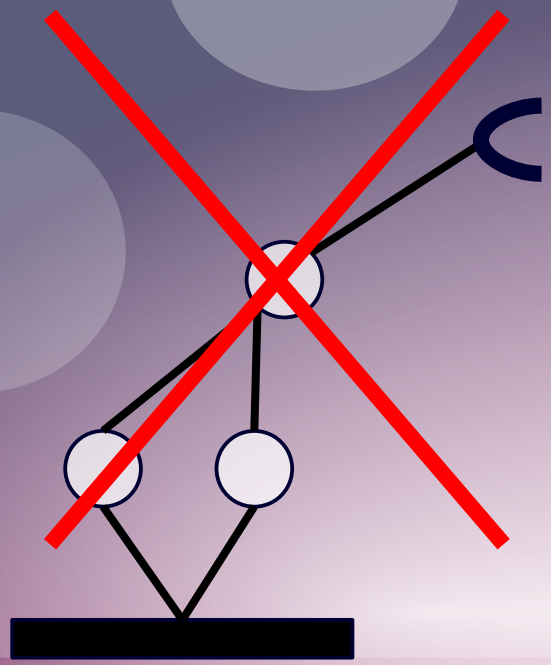

```
<link name=’.’>
        <visual>
                <geometry>
                        <>
                </geometry>
                <origin xyz= ‘. . .’ rpy=’. . .’/>
        </visual>
        <collision>
                <geometry>
                        <>
                </geometry>
                <origin xyz= ‘. . .’ rpy=’. . .’/>
        </collision>
        <inertial>
                <mass value=’.’/>
                <inertia ixx=’.’ ixy=’.’ ixz=’.’ iyy=’.’ iyz=’.
Izz=’.’/>
        </inertial>
</link>
```

Le format URDF : joint




```
<joint name=’.’ type=’.’>
        <parent link=’.’/>
        <child link=’.’/>
        <axis xyz=’. . .’/>
        <origin xyz= ‘. . .’/>
        <lower=’.’. />
        <upper=’.’/>
        <effort=’.’/>
        <velocity=’.’/>
</joint>
```

Le format URDF : limitations



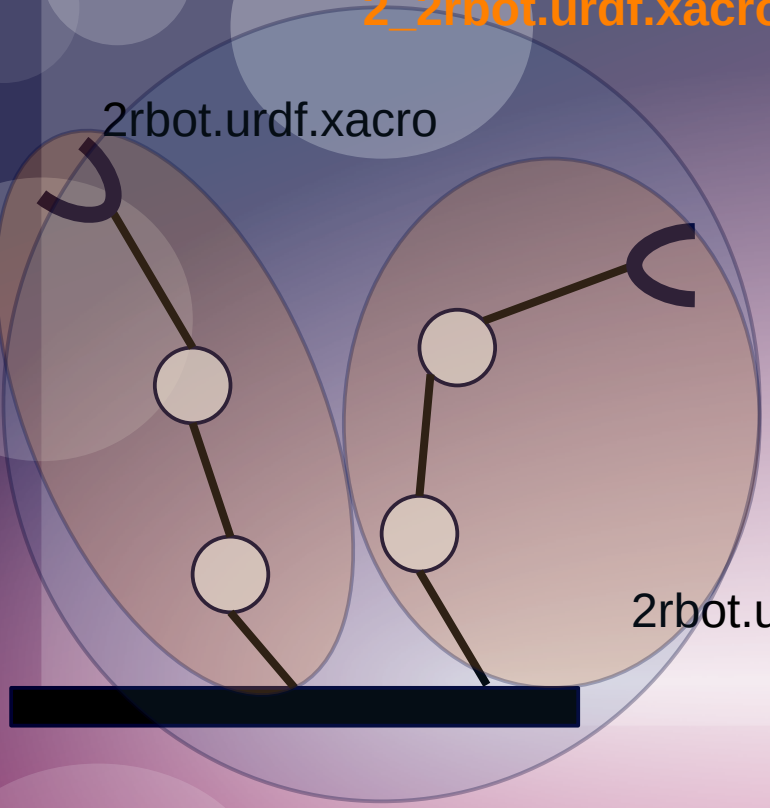
Pas adapté pour les structures paralleles

Type de liaison limité

Le format URDF : extension xacro

2_2rbot.urdf.xacro

2rbot.urdf.xacro



2rbot.urdf.xacro

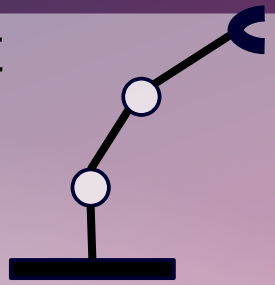
```
<xacro:macro name=’my_urdf’ params=’name’
        <link name=’${name}’>
                ...

                ...
        </link>
</xacro:macro>

<xacro:my_urdf name= ‘.’ />
```

Place à la pratique

1) Construire l'URDF correspondant à ce robot 2R grâce au tutoriel ros



Envoyer les différents fichiers créés par mail en fin de séance

Les contrôleurs moteurs dans ROS2

- Selon les robots, plusieurs types existent (non-exhausitf) :
  - JTAS (Joint Trajectory Action Server) exécute des JointTrajectory pour la commande de trajectoires articulaires.
  - Contrôleurs hardware avec ros2_control (C++ et Python)
    - PositionJointInterface
    - VelocityJointInterface
    - EffortJointInterface
  - ….

Les contrôleurs moteurs dans ROS2 : implémentation

- Définir les parametres du controleur dans un .yalm :

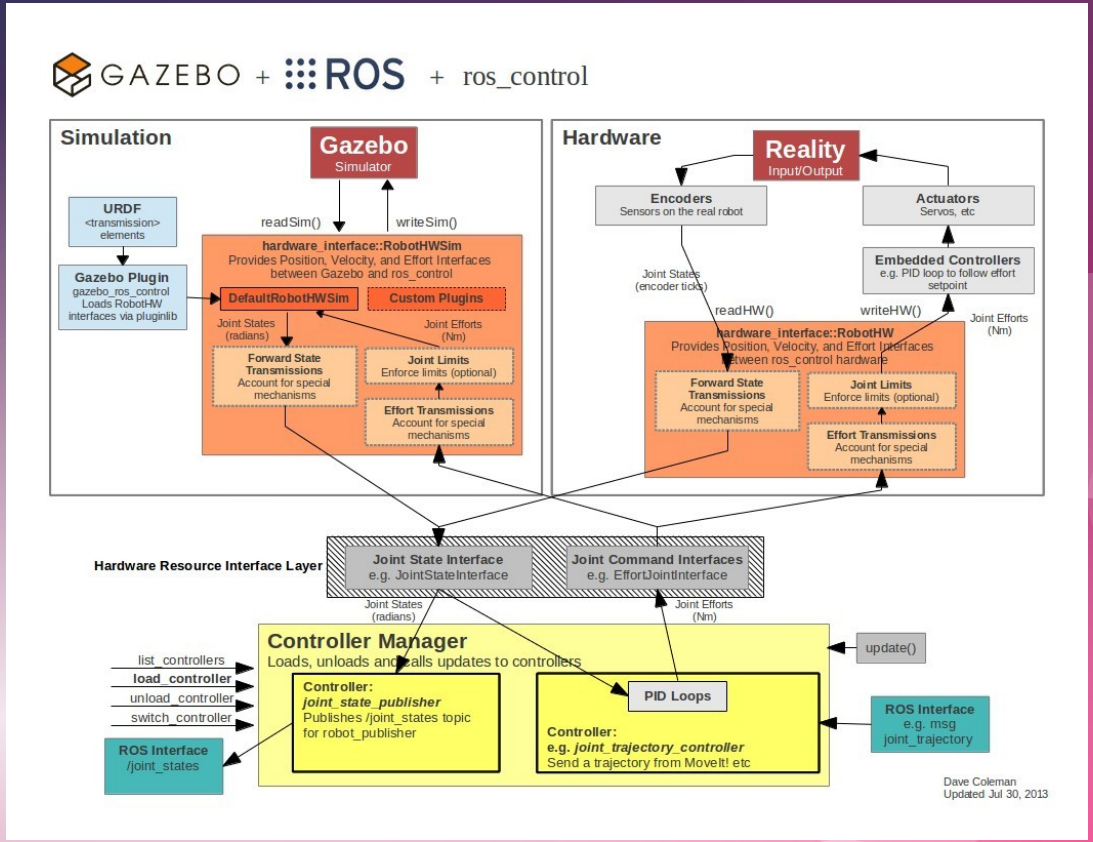
```
controller_manager :
        ros_parameters :
                joint1_position_controller :
                        type: "position_controllers/JointPositionController"
                        joint: "joint1"
                        pid: {p: 100.0, i: 0.01, d: 0.5}
…
```

- Créer une classe *hardware_interface::RobotHW* pour communiquer avec le robot
- Lancer une launch file avec :
  - Un node *controller_manager* pour chaque joint
  - Un node *ros2_control _node*
  - Un node *robot_state_publisher* qui vient lire la description urdf du robot

Les contrôleurs moteurs dans ROS2 : simulation avec Gazebo

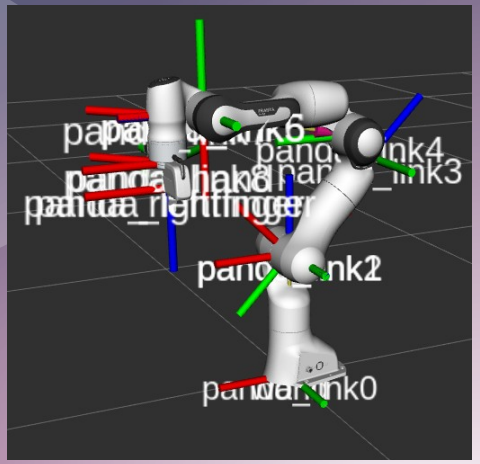
- Utilisation modulaire grâce à un plugin qui s'ajoute à l'urdf
- Permet de tester facilement un contrôle avant de tout casser en réalité

Pour plus d'information voir : https://control.ros.org/humble/doc/getting_started/getting_started.html



La planification : passage en cartésien

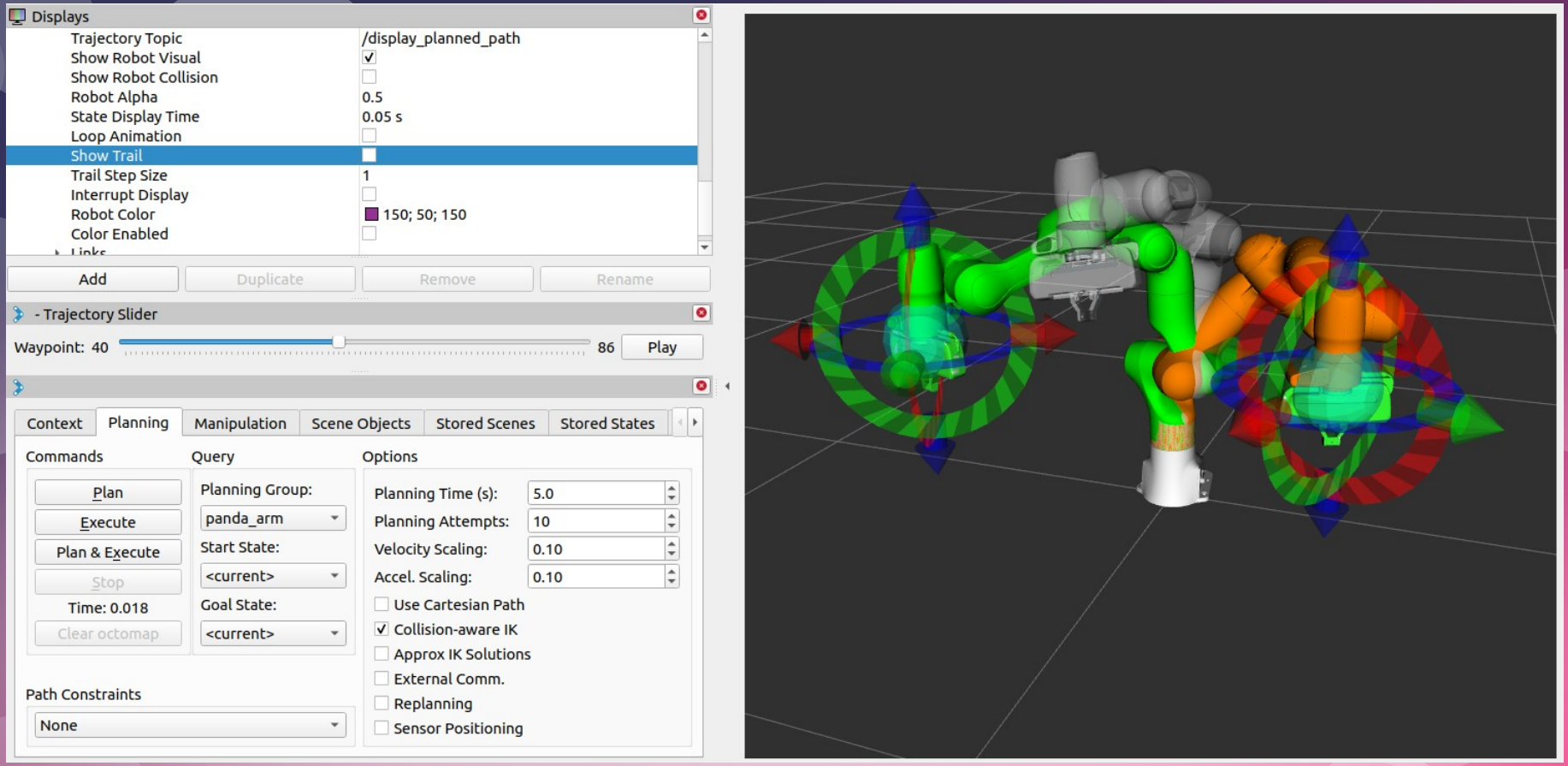
- Pour des tâches plus compliquées, on va devoir passer dans l'espace cartésien :
  - Calcule de la cinématique inverse : Pinocchio



Tf tree

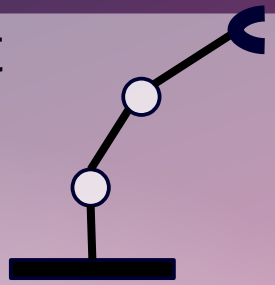
```
aboulay@vitis22:~/ros_ws/test_ws$ ros2 run tf2_ros tf2_echo panda_link0 panda_hand
[INFO] [1734477686.706726462] [tf2_echo]: Waiting for transform panda_link0 ->  panda_hand: Invalid f
rame ID "panda_link0" passed to canTransform argument target_frame - frame does not exist
At time 1734477687.627880068
- Translation: [0.307, -0.000, 0.590]
- Rotation: in Quaternion [1.000, 0.000, -0.000, 0.000]
- Rotation: in RPY (radian) [3.142, 0.000, 0.000]
- Rotation: in RPY (degree) [180.000, 0.000, 0.023]
- Matrix:
  1.000  0.000  0.000  0.307
  0.000 -1.000 -0.000 -0.000
 -0.000  0.000 -1.000  0.590
  0.000  0.000  0.000  1.000
At time 1734477688.677935059
```

La planification : moveit2



Place à la pratique

1) Construire l'URDF correspondant à ce robot 2R grâce au tutoriel ros



2) Créer un scénario de pick&place sur moveit2 avec 2 obstacles à éviter grâce au tutoriel moveit
3) Bonus : Prendre le contrôle du 2R réalisé en 1)

Envoyer les différents fichiers créés par mail en fin de séance