Workshop ROSCON
De 0 à utiliser un UR10 avec ROS 2

ROSCON FR & DE

17/11/2025

IRT JULES VERNE

LE FUTUR
DE VOS USINES

ROS et ROS 2, une introduction

ROS – L'Histoire

Arrêter de réinventer la roue

- **Créé par la startup Willow Garage en 2007**
- **Objectif : Fournir une base logicielle robotique commune et open source pour la recherche, basée sur**
  - Réutilisation des données facilitée
  - Transfert et déploiement simplifié
- **2010 -** ROS
- **2017 -** 2

Actuellement maintenu par la société open source robotics alliance

Source : https://www.robotshop.com/community/blog/show/robot-humor-dinner-comics

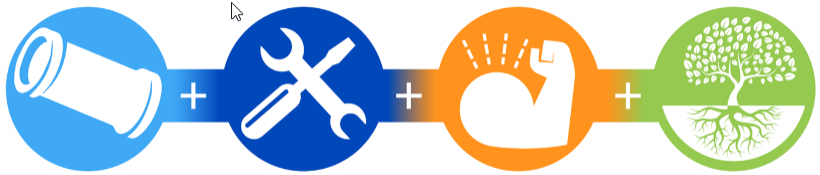
ROS (Robot Operating System) – Les concepts

- Un middleware avec 4 aspects principaux

Couche Middleware

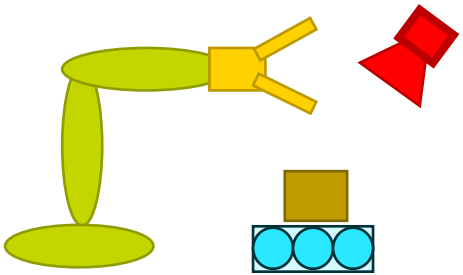
ROS

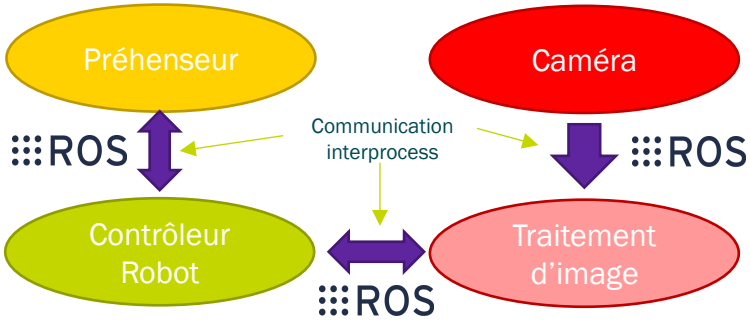
Couche Logicielle



Communication Inter-process

Cas Pick and Place

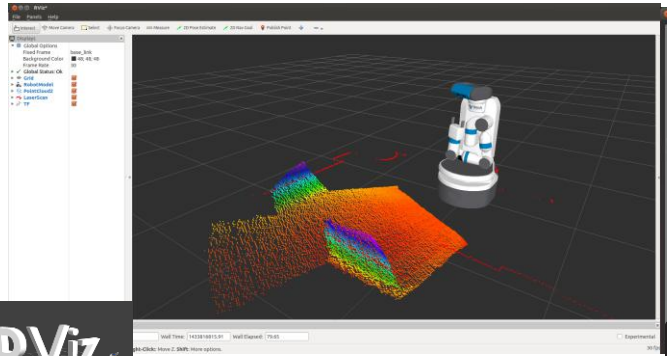




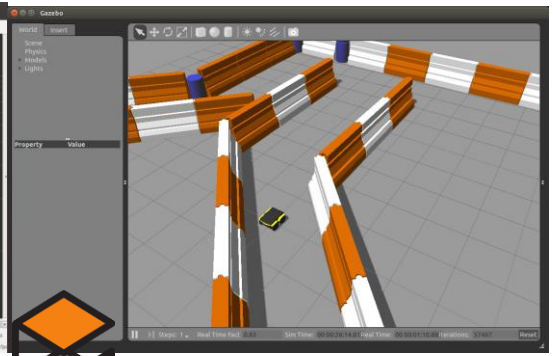
ROS fournit un protocole d'échange d'informations « standardisé » entre les différents modules basé sur la publication et la souscription.

Des outils logiciels

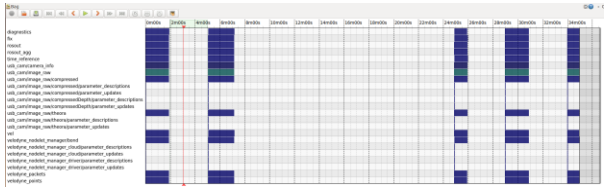
Accélère le développement



Outils de visualisation

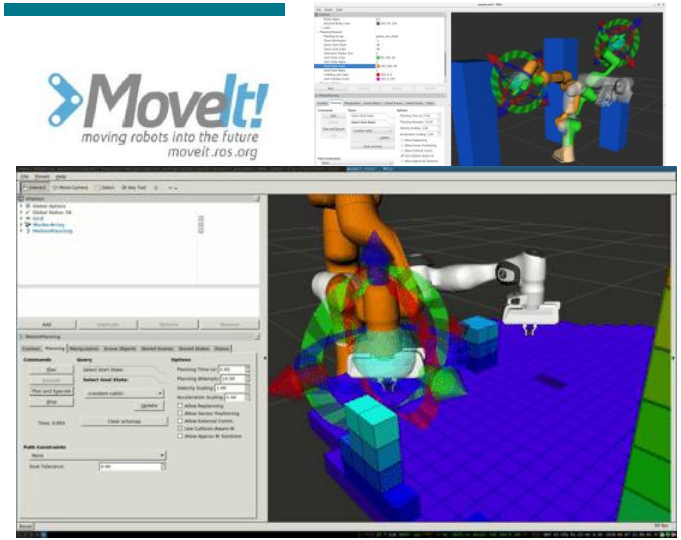


Outils de simulation

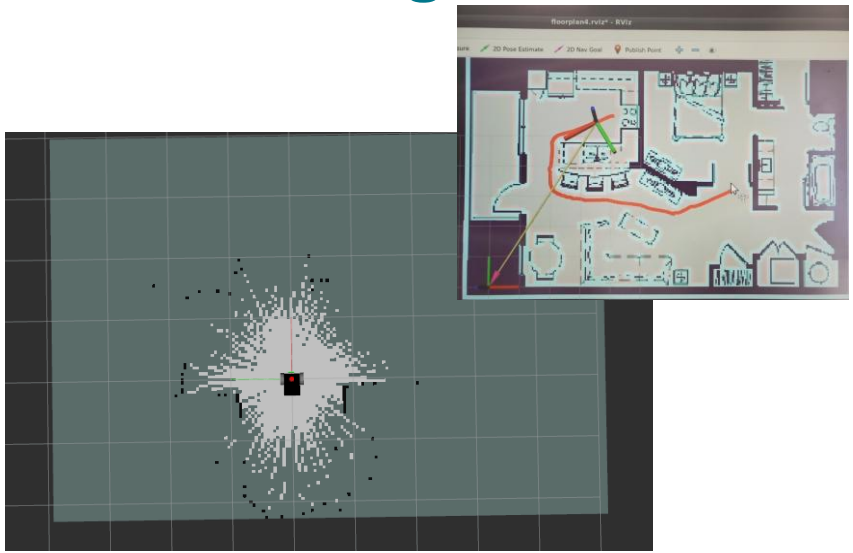


Enregistrement et relecture de données

Algorithmes et compétences gratuites « sur étagère »



Planificateur de trajectoires (avec évitement d'obstacle)

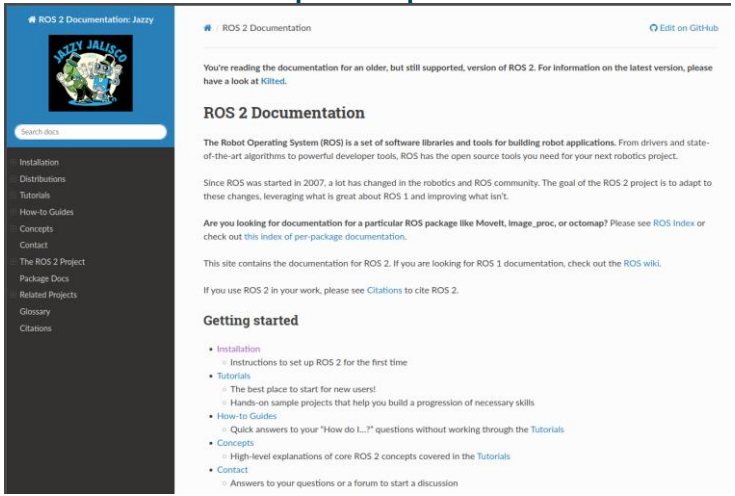


Outils de cartographie et de navigation 2D

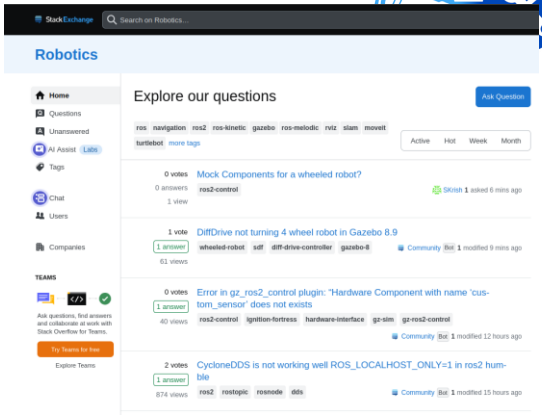
Une grande communauté

ROS.org

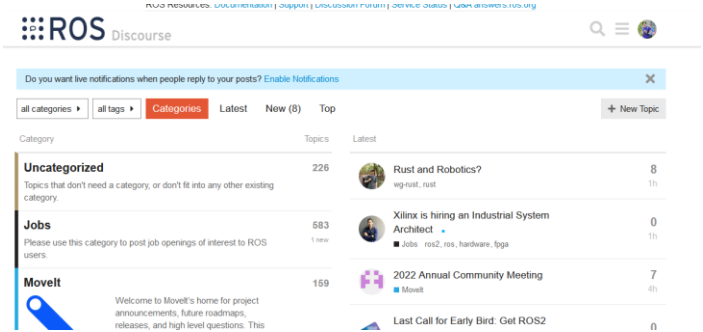
Site principal



https://docs.ros.org/en/jazzy/index.html



Questions-réponses (Robotics Stackexchange)



Forum de la communauté (Discourse)

Pourquoi ROS ?

- **Profiter d'algorithmes open-sources à l'état de l'art**
- **Faciliter l'intégration avec différent matériels**
- **Partager les données facilement**

« Les robots et les challenges robotiques sont tellement complexes qu'ils deviennent impossibles à répondre seul »
O.Stasse, directeur de l'équipe GEPETO du CNRS LAAS,

Journée Francophone ROS 2017

Introduction à ROS

- Questions ?

Installation ROS 2

Choisir sa version de ROS en fonction de sa version d'UBUNTU

Version	Ubuntu Noble	Ubuntu Jammy	Windows 10	MacOS
Kilted Kaiju	+	×	•••	•
Jazzy Jalisco	+	×	•••	•
Humble Hawksbill	×	+	•••	•
Rolling Ridley	•••	×	•••	•

+ **Tier 1: Fully Supported & Recommended for New Users** ••• Tier 1: Fully Supported •• Tier 2: Limited Support • Tier 3: Community Support

Comment installer ROS ?

Suivre les instructions disponibles sur : https://docs.ros.org/

En debian package.

Installer :

- **ROS-DEV-TOOLS**
- **DESKTOP**

Faire les commandes :

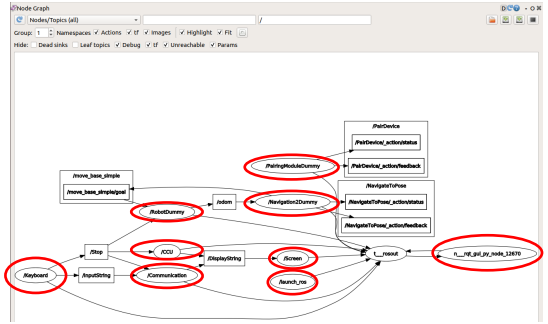
```
sudo rosdep init
```

```
rosdep update
```

Les concepts de bases de ROS 2

Les noeuds

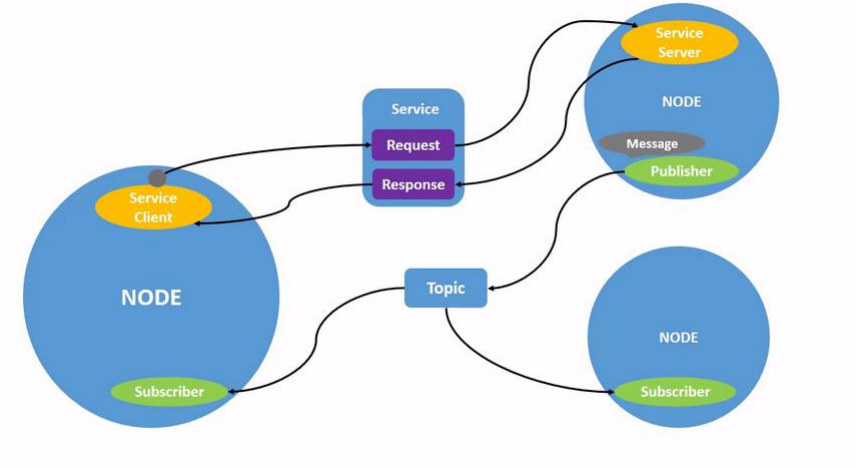
Le concept central



Les nœuds (Nodes) – Ce sont les processus qui communiquent entre eux.

Ils peuvent être :

- Dans un même process
- Dans des process différents sur le même PC
- Sur des PC différents



Un nœud (indépendamment de son implémentation) peut communiquer avec les autres nœuds.

Ex : Un nœud en C++ peut communiquer avec un nœud en python

Les topics

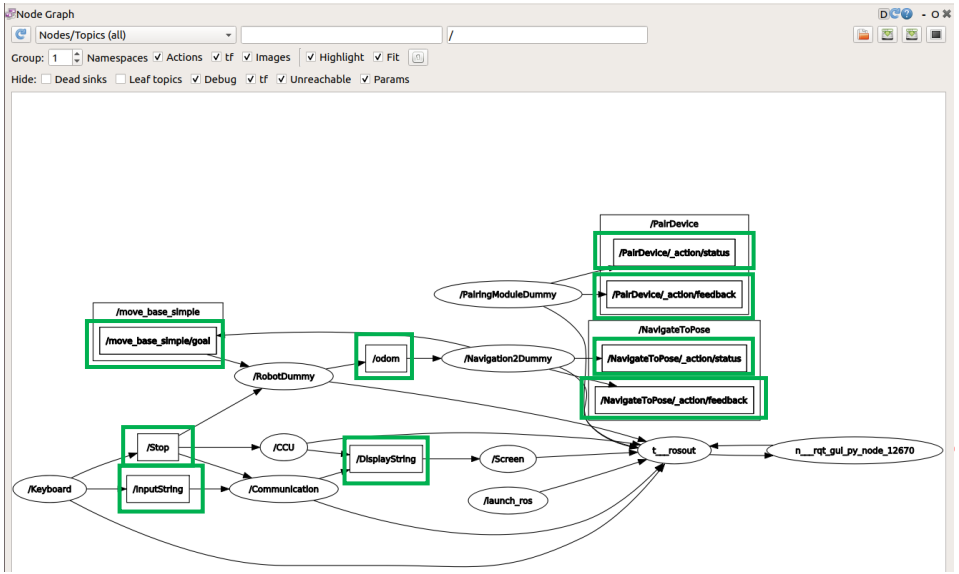
La base de la communication

Les topics – Ce sont des « lieux d'échanges » de données. Ils sont nommés.

Il connecte toujours :

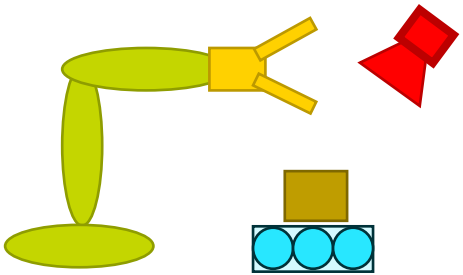
- Un (ou plusieurs) nœud(s) fournisseur de données (**Publisher**)
- Un ou plusieurs nœuds consommateurs de données (**Subscriber**)

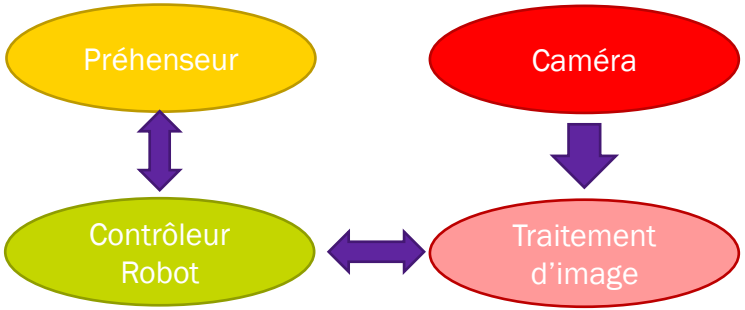
Le Publisher et le Subscriber doivent s'être accorder sur la structure des données échangés (Message)



Exemple

Pick and place



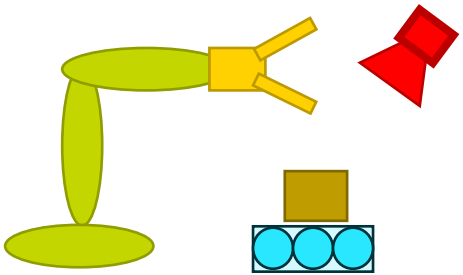


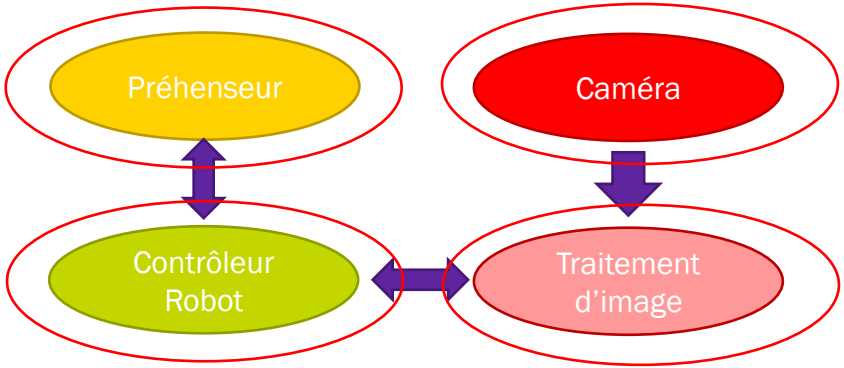
Trouvé les nœuds ?

Trouvé les topics ?

Exemple

Pick and place

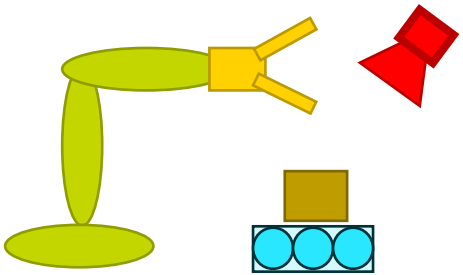


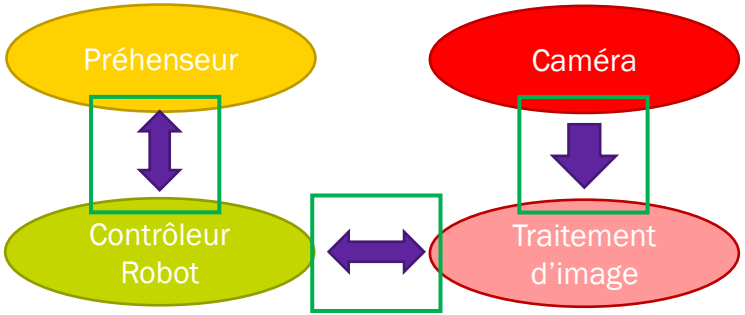


Les nœuds sont les éléments qui communiquent ensemble

Exemple

Pick and place





Les topics sont les lieux d'échanges des données

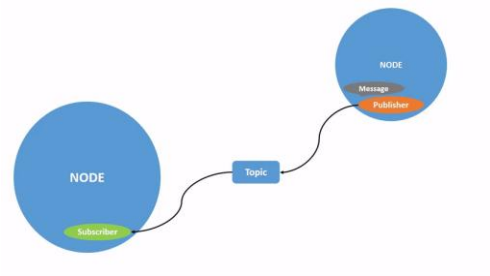
Les échanges de données sur ROS

3 types d'échanges principaux

1. **Messages**
2. **Services**
3. **Actions**

(Paramètres)

Les topics et les messages



Echange asynchrone et unilatérale

Il est possible d'avoir plusieurs subscriber pour chaque topic

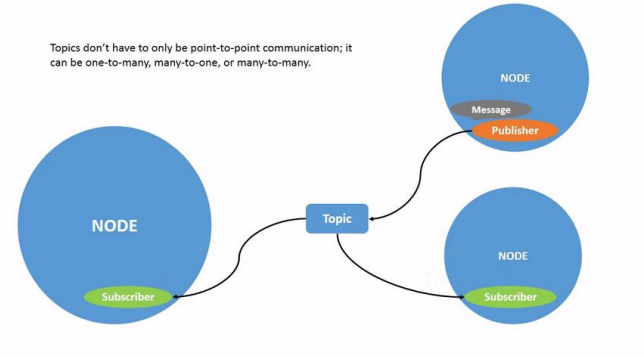
Il est possible d'avoir plusieurs publisher pour chaque topic (mais attention, mauvaise pratique!)

Fonctionne de manière asynchrone

- Pas besoin d'attendre que le subscriber
  est lu le message pour envoyer un nouveau message
- Possibilité d'avoir une mémoire/buffer (Queue) côté publisher

Chaque topic contient une structure de donnée appeler Message

- Il existe des messages « standards »
- Il est possible de créer des messages « custom »



Les messages

Une structure de données

Défini dans un fichier .msg

- Peut contenir plusieurs champs « simple »
- Peut-être composé d'autre messages

geometry_msgs/Point Message

File: geometry_msgs/Point.msg

Raw Message Definition

```
# This contains the position of a point in free space
float64 x
float64 y
float64 z
```

Compact Message Definition

```
float64 x
float64 y
float64 z
```

geometry_msgs/Pose Message

File: geometry_msgs/Pose.msg

Raw Message Definition

```
# A representation of pose in free space, composed of posi
Point position
Quaternion orientation
```

Compact Message Definition

```
geometry_msgs/Point position
geometry_msgs/Quaternion orientation
```

Il existe des messages « communs » et des messages « standards »

- http://wiki.ros.org/common_msgs => Liés à leur utilisation (Pose, Image, …)
- http://wiki.ros.org/std_msgs => Format de données « standards » (Booléen, Tableau de float64, …)

Démonstration Live

- **Publisher**
- **Subscriber**
- **Subscriber 2**
- **Lancement Publisher 2**
- **Graph**
- **Commande ROS 2 TOPIC**

Les services

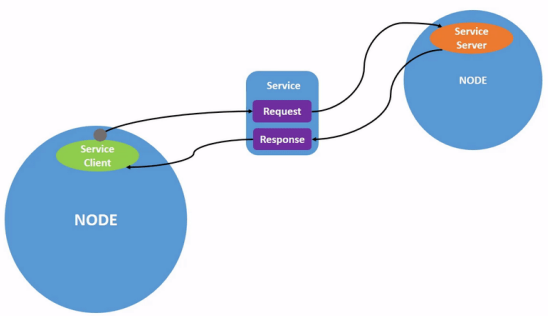
Communication synchrone

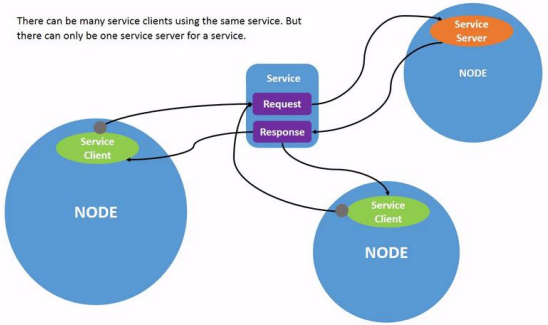
Fonctionnement sur la base de Requête (Request) et Réponse (Response)

- Le serveur attend une requête (qui peut contenir des données) et renvoie une réponse (qui peut contenir des données)
- Echange ponctuelle. La connexion est, en générale, recrée à chaque appel.

Un service est nommé et est lié à une structure de donnée double (Requête et Réponse)

Echanges court terme avec des besoins de synchronisations sur les échanges





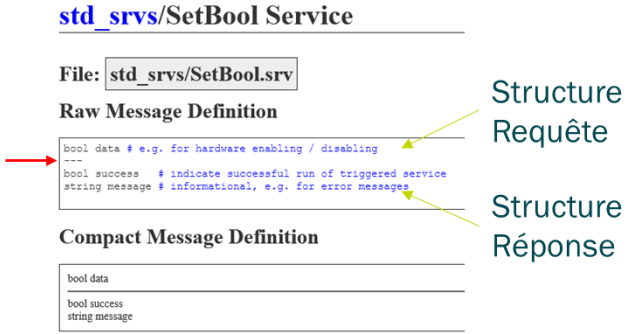
La structure des services (.srv)

Structure de données double

Défini dans un fichier .srv

- Même construction de donnée qu'un .msg (donnée standard, ou d'autre structure de messages)
- Contient à la fois, la structure de la requête et la structure de la réponse séparée par un « --- »

Il existe très peu de services standards, mais beaucoup d'algorithmes et d'outils ROS fournissent des services



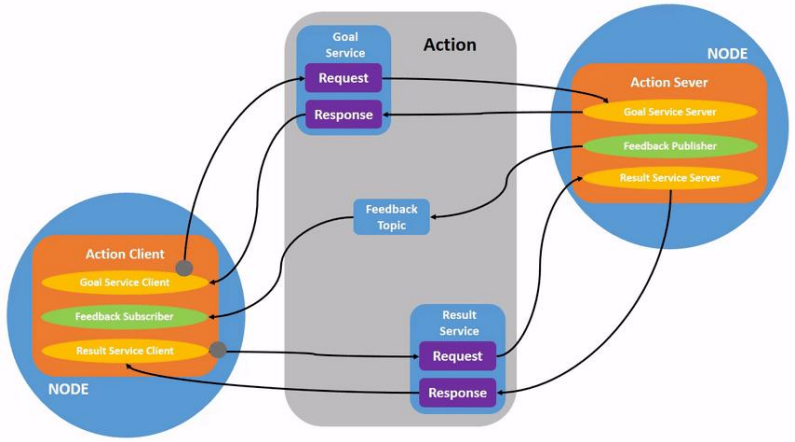
Les actions

Communication annulable synchrone avec feedback

Fonctionnement complexe basé sur des services et des topics synchronisés par une machine à état interne.

- Le serveur attend une requête (Goal) (qui peut contenir des données) et renvoie des données standards (acceptation ou refus)
- Le serveur exécute l'action en publiant (généralement) des messages Feedback sur un topic
- Le client peut à tout moment annuler l'action (Cancel)
- Une fois l'exécution terminé, le serveur renvoie une réponse (Result) qui contient des données et le statut (Succeeded, Aborted, Canceled)

Echange long terme prévu pour des actions longues



La gestion des fichiers dans ROS

Les grands concepts

ROS fonctionne grandement avec des variables d'environnements pour trouver et faciliter l'accès au différents éléments (bibliothèque, exécutable)

ROS utilise un système de compilation configurable appelé Ament, lui-même appeler par un outil appeler COLCON

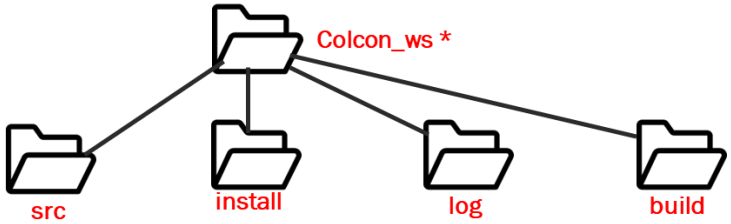
Les concepts importants pour la gestion des fichiers :

- **Les workspaces**
- **Les packages**
- **Les manifests**

Les workspace

La racine des dossiers ROS

Un workspace est l'endroit où sont stockés et organisés les fichiers ROS



Colcon_ws * - Dossier d'origine du workspace (peut avoir n'importe quelle nom)

Src – Dossier contenant les fichiers sources organisés sous forme de packages. Ce dossier est à créer « à la main » !

Install – Dossier généré automatiquement par colcon qui contient les exécutables et les librairies compilés ou copiés (si pas compiler). Il contient également les scripts pour les variables d'environnements pour utiliser le workspace

Log – Dossier généré automatiquement par colcon contenant les logs des opérations de compilations

Build – Dossier généré automatiquement par colcon contenant les fichiers « cache » des opérations de compilations

Package

Brique minimale dans le système de fichier de ROS

Elément minimal pour la compilation et l'exécution d'un système sous ROS.

Il peut être installer deux manières :

- **Installer « déjà compilé » dans le workspace d'origine de ROS à l'aide du système de packages de l'OS (par exemple Debian pour Ubuntu)**
- **Compiler et installer localement depuis les sources dans un workspace créé manuellement. Les sources font alors partie du package.**

L'élément minimal pour un package pour être reconnu par COLCON est le Manifest (package.xml). D'autres éléments peuvent être nécessaire pour la compilation (CMakeLists.txt pour les fichiers C++ par exemple).

Le Manifest

Le Fichier Package.xml

Un fichier permettant à ROS de comprendre le contenue, les dépendances et la manière de compiler le package. C'est la carte d'identité du package.

Permet d'éviter le problème de « ça fonctionne sur mon PC » si bien rempli.

xml — Copier le code

```xml
<package format="3">
  <name>my_package</name>
  <version>0.0.0</version>
  <description>My ROS 2 package</description>
  <maintainer email="your_email@example.com">Your Name</maintainer>
  <license>Apache-2.0</license>

  <buildtool_depend>ament_cmake</buildtool_depend>
  <buildtool_depend>ament_python</buildtool_depend>

  <exec_depend>rclpy</exec_depend>
  <exec_depend>std_msgs</exec_depend>

  <test_depend>ament_lint_auto</test_depend>
  <test_depend>ament_lint_common</test_depend>
</package>
```

La compilation

Simple et efficace

Une fois à la racine d'un workspace (!!! Important !!!), pour compiler l'ensemble des packages contenues dans le dossier « src » d'un workspace

La commande `colcon build` **permet de compiler tout le workspace.**

Pour charger les variables d'environnement de ce workspace, il est nécessaire d'utiliser le script qui est générer si la compilation a abouti. Pour Ubuntu :

```
source ws_path/install/setup.bash
```

ROS 2 – Concepts et workspaces

- Questions ?

Exercice 1

Créer un workspace, un package, et compiler des sources

Correction

Les outils de ROS

Les lignes de commande

Le B-A-BA de ROS

Les lignes de commandes permettent de faire énormément de choses sur ROS :

- **Lancer un ou plusieurs exécutables**
- **Enregistrer ou relire des données échangées sur le réseau**
- **Faire de l'introspection sur ce qu'il se passe actuellement**
- **Publier / Ecouter sur des topics**
- **Appeler des services**
- **…**

Toute les commandes ROS commence par :

```
ros2
```

Exemple

```
ros2 --help
```

Les lignes de commandes

Lancer un programme

- **Exécute un programme du package (doit être executable)**

ros2 run *package_name nom_du_programme* arguments

- **Lance le fichier launch (et les programmes associés)**
  - Possibilité de passer les arguments déclarées (utiliser « :=»)

ros2 launch *package_name fichier_launch*

Les lignes de commandes

Les topics

ros2 topic info *nom_du_topic -v*

- **Donne toutes les informations sur le topic (publishers, subscribers, type, QoS)**

ros2 topic echo *nom_du_topic*

- **Souscrit au topic et affiche le contenue des messages reçus**

ros2 topic pub *nom_du_topic type_de_message “{contenue}”* –r *X*

- **Crée un publisher sur le topic qui publie a une fréquence de X le contenue**
  - Il faut mettre le contenue sous forme de dictionnaire python.

Les lignes de commandes

Les topics

`ros2 topic list`

- **Donne tous les topics présents sur le réseau**

`ros2 topic hz *nom_du_topic*`

- **Souscrit au topic et affiche la fréquence des messages reçus**

Les lignes de commandes

Les services

ros2 service info *nom_du_service*

- **Donne toutes les informations sur le service (nœud server, type)**

ros2 service list

- **Liste les services actifs**

ros2 service call *nom_du_service type_de_service “contenue”*

- **Fait une requête sur le service et affiche la réponse**
  - Contenue sous forme de dictionnaire python

Les lignes de commandes

Les noeuds

ros2 node info *nom_du_node*

- **Donne toutes les informations sur le noeud (service server, publishers, subscriber, IP, PID)**

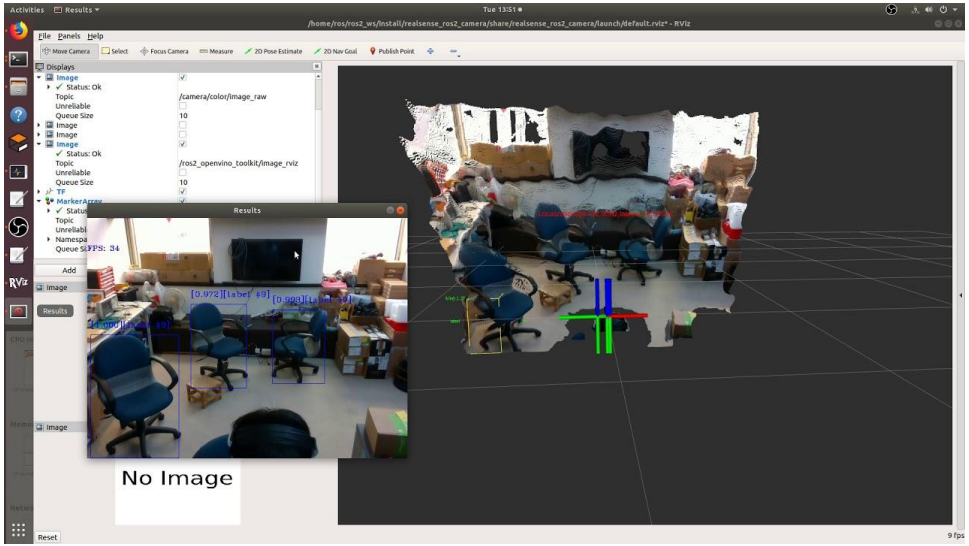
ros2 node list

- **Liste les nœuds présents sur le réseau**

RVIZ 2

Outil de visualisation

rviz2



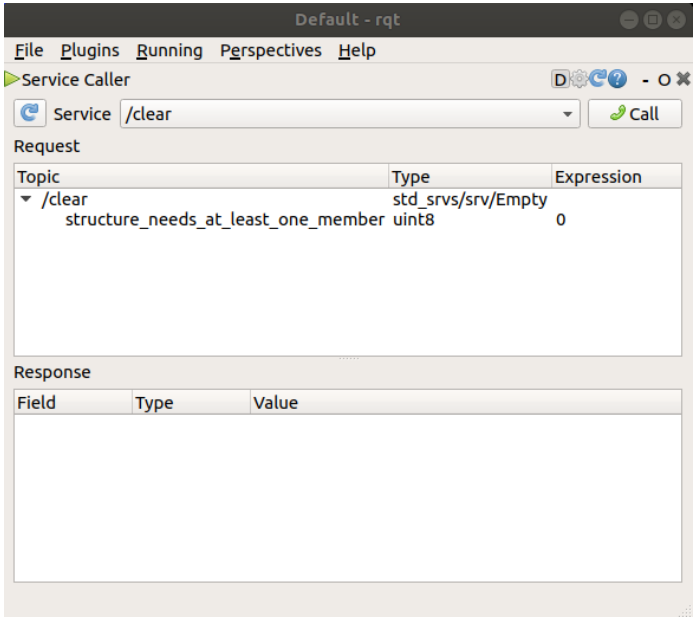
- **Contient des visualisations préparer**
  - Modèle 3D d'un robot
  - Repères géométriques
  - Nuages de points
  - Images
  - ….
- **Possibilité d'éditer complétement le layout**

RQT

- L'outil de debug ultime - https://docs.ros.org/en/humble/Concepts/Intermediate/About-RQt.html

rqt

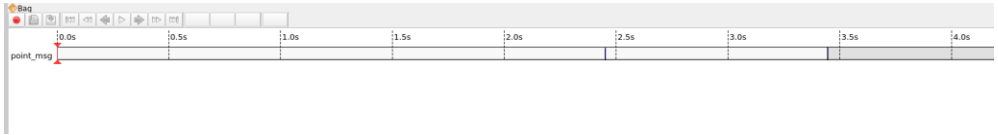
- **Contient beaucoup d'éléments utiles**
  - Loggers
  - Node Graph
  - Publisher
  - Service Caller
  - …



ROSBag

Enregistrement des topics

rqt_bag



Permet d'enregistrer le contenue des messages, leurs timing et de les rejouer

Outil très utile pour le logging / debugging à distance / tests

Outils

- Questions ?

Exercice 2

Utiliser les outils ROS

Correction

Manipulation avec ROS

Les bases

Package Transform (TF)

Un package pour gérer tous les repères géométriques

Problèmes classiques en robotique :

- Quelle est la matrice de passage du repère base robot au repère effecteur ?
- Quelle est la coordonnée de ce point dans le repère du robot ?
- Comment on passe d'un angle en Euler XYZ a un quaternion ?
- Où était le robot il y a 10 secondes par rapport au centre de cette image ?

➔ **TF (TF2) et ses outils permettent de répondre à tous ces problèmes**

Il donne un ensemble outils mathématiques (et d'APIs pour ROS) permettant de faire toutes ces opérations simplement.

Package Transform (TF)

Des APIs particulières

Système complexe basés sur un topic « /tf » (éventuellement « /static_tf »)

- Ce système tient à jour une liste des données géométriques grâce a ce topic.
- Il est très difficile de comprendre manuellement ce qui se passe sur ce topic.
- Il ne faut pas utiliser des Publishers ou des Subscribers « classiques ».
- Heureusement, il y a déjà beaucoup de choses qui sont déjà automatisées

Si l'on souhaite ajouter / calculer une transformée géométrique, on utilise :

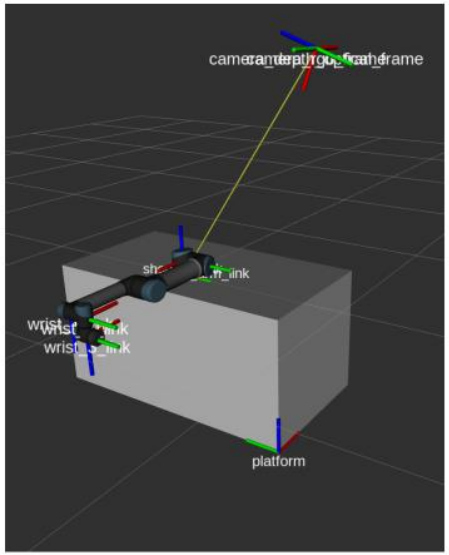
- Un broacaster (pour envoyer une transformée géométrique)
- Un listener (pour calculer une transformée géométrique)

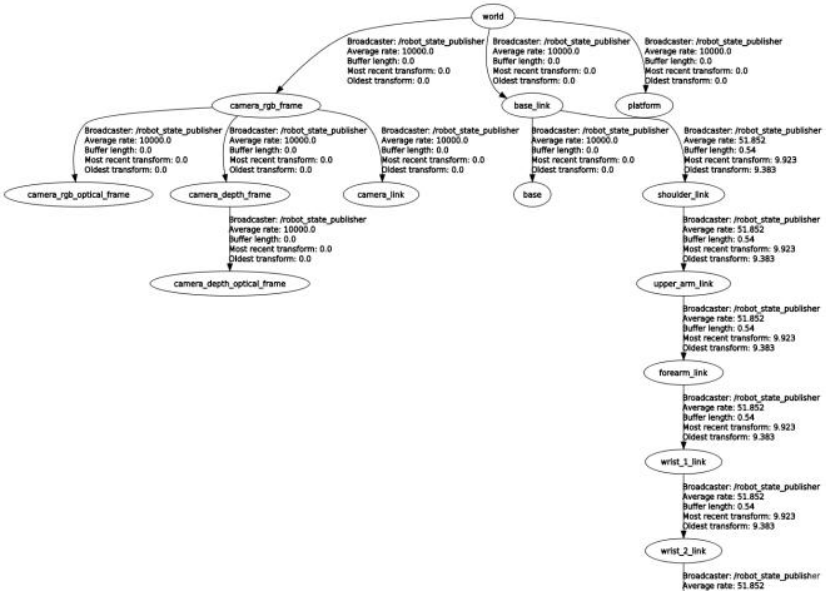
TF2 – Arbre de transformés

Un outil pour savoir qui est placé par rapport à quoi

Très utiles pour débugger

Chaines ouvertes uniquement

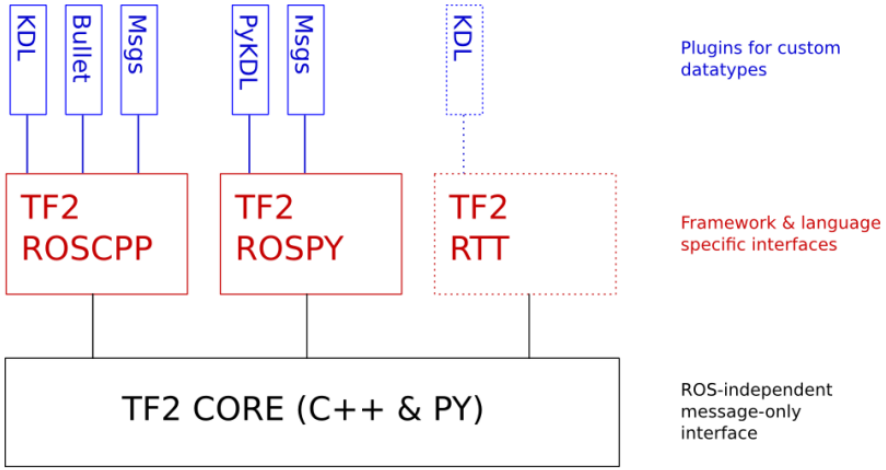




TF2 - Architecture

Indépendant de ROS

- **Possibilité d'utiliser le système en dehors de ROS**
- **Interfaçage avec ROS automatique**

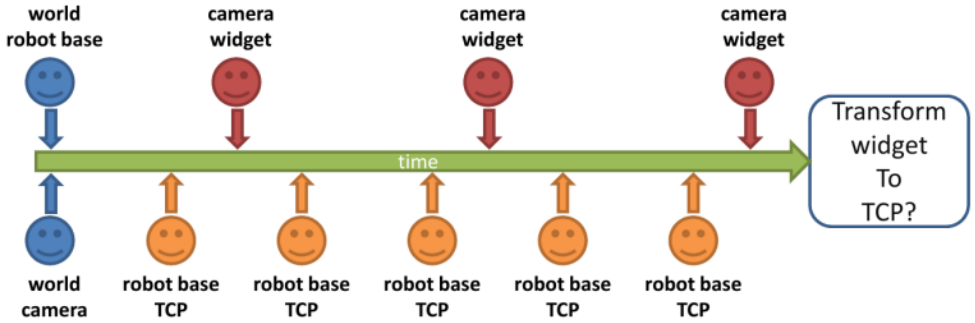


TF2 – Synchronisation Temporel

TF2 permet d'obtenir des approximations de la position dans le temps

- **Buffer de 10 secondes pour chaque broadcaster (configurable)**
- **Possibilité de définir le moment pour obtenir une transformé (la dernière en date, il y a 1 seconde)**
- **Possibilité de faire de l'interpolation (dans le passé)**

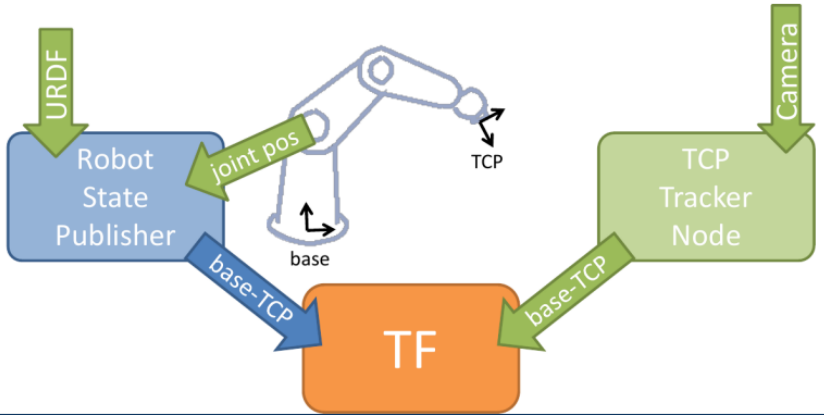
- **Fortement influencé par la fréquence de « publication »**



TF2 - Broadcasters

Les publishers de TF2

- **Possibilité d'en créer à la main (via les API)**
- **En général, on utilise, pour un bras robot des broadcaster automatique appelés: « Robot State Publisher »**
- **Récupère les données articulaires du robot (topic « joint_state ») et un fichier de description géométrique (format URDF) pour automatiquement générer toute les transformés**
- **Il existe des broadcaster « statiques » pour envoyer des informations de transformé « fixes »**



Démonstration

Lancer un exemple de broadcaster

Déplacer le robot

Faire un echo sur une position

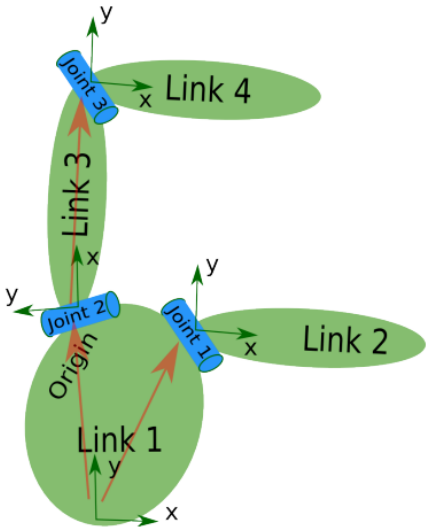
TF2

- Questions ?

Modèle d’un robot URDF

Exemple d'URDF

- **Toutes les URDFs utilisent les balises « robot »**
- **On utilise principalement les balises link et joint**
- **L'origine est toujours un link**
- **Un joint est toujours entre deux link (balise parent et chlid)**

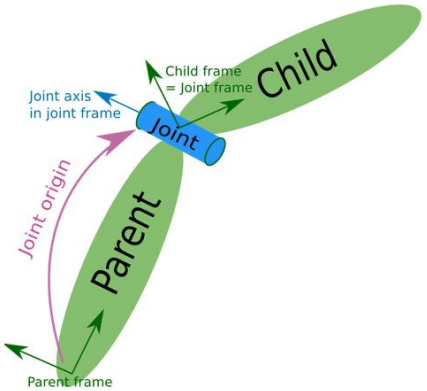



```
<robot name="test_robot">
  <link name="link1" />
  <link name="link2" />
  <link name="link3" />
  <link name="link4" />

  <joint name="joint1" type="continuous">
    <parent link="link1"/>
    <child link="link2"/>
  </joint>

  <joint name="joint2" type="continuous">
    <parent link="link1"/>
    <child link="link3"/>
  </joint>

  <joint name="joint3" type="continuous">
    <parent link="link3"/>
    <child link="link4"/>
  </joint>
</robot>
```

Ajout des dimensions

- http://wiki.ros.org/urdf/XML/joint

- **Exprimé dans le repère parent**
- **Le link enfant est exprimé dans le repère du joint**
- **Le joint précédent est le repère parent**

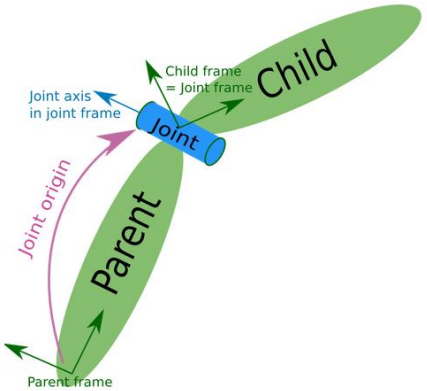



```xml
<robot name="test_robot">
  <link name="link1" />
  <link name="link2" />
  <link name="link3" />
  <link name="link4" />


  <joint name="joint1" type="continuous">
    <parent link="link1"/>
    <child link="link2"/>
    <origin xyz="5 3 0" rpy="0 0 0" />
  </joint>

  <joint name="joint2" type="continuous">
    <parent link="link1"/>
    <child link="link3"/>
    <origin xyz="-2 5 0" rpy="0 0 1.57" />
  </joint>

  <joint name="joint3" type="continuous">
    <parent link="link3"/>
    <child link="link4"/>
    <origin xyz="5 0 0" rpy="0 0 -1.57" />
  </joint>
</robot>
```

Ajout des propriétés cinématiques

- http://wiki.ros.org/urdf/XML/joint

- **L'axe est exprimé dans le repère du joint (après avoir été positionné dans le repère du parent)**
- **Possibilité de définir des limites (positions, vitesses, effort)**



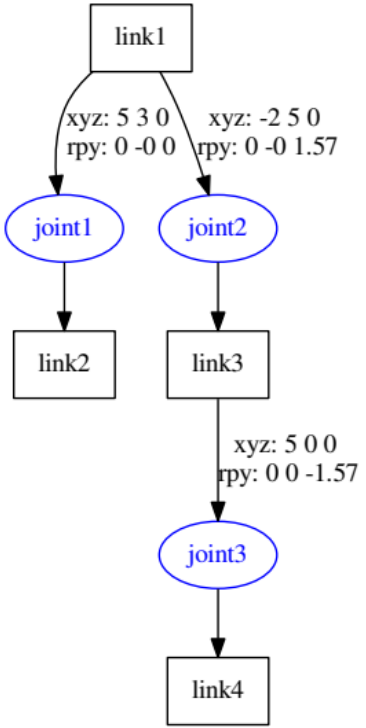

```
<robot name="test_robot">
  <link name="link1" />
  <link name="link2" />
  <link name="link3" />
  <link name="link4" />
  <joint name="joint1" type="continuous">
    <parent link="link1"/>
    <child link="link2"/>
    <origin xyz="5 3 0" rpy="0 0 0" />
    <axis xyz="−0.9 0.15 0" />
  </joint>
  <joint name="joint2" type="continuous">
    <parent link="link1"/>
    <child link="link3"/>
    <origin xyz="−2 5 0" rpy="0 0 1.57" />
    <axis xyz="−0.707 0.707 0" />
  </joint>
  <joint name="joint3" type="continuous">
    <parent link="link3"/>
    <child link="link4"/>
    <origin xyz="5 0 0" rpy="0 0 −1.57" />
    <axis xyz="0.707 −0.707 0" />
  </joint>
</robot>
```

Arbre de l'URDF

Une visualisation d'un URDF sous forme d'arbre

- **Utilisation du logiciel graphiz pour afficher le graphe**

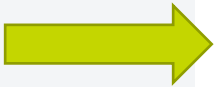
urdf_to_graphiz *fichier.urdf*



Eviter les duplications en utilisant XACRO

Simplifie et permet de générer du code URDF

```
<link name="base_link">
  <visual>
    <geometry>
      <cylinder length="0.6" radius="0.2"/>
    </geometry>
    <material name="blue"/>
  </visual>
  <collision>
    <geometry>
      <cylinder length="0.6" radius="0.2"/>
    </geometry>
  </collision>
</link>
```



```
<xacro:property name="width" value="0.2" />
<xacro:property name="bodylen" value="0.6" />
<link name="base_link">
    <visual>
        <geometry>
            <cylinder radius="${width}" length="${bodylen}"/>
        </geometry>
        <material name="blue"/>
    </visual>
    <collision>
        <geometry>
            <cylinder radius="${width}" length="${bodylen}"/>
        </geometry>
    </collision>
</link>
```

XACRO

Permets également

- **Spécifié des constates / variables**

```
<xacro:property name="the_radius" value="2.1" />
<xacro:property name="the_length" value="4.5" />

<geometry type="cylinder" radius="${the_radius}" length="${the_length}" />
```

- **Faire des fonctions mathématiques simples (additions, multiplications, divisions)**

```
<xacro:property name="R" value="2" />
<xacro:property name="alpha" value="${30/180*pi}" />
<circle circumference="${2 * pi * R}" pos="${sin(alpha)} ${cos(alpha)}" />
<limit lower="${radians(-90)}" upper="${radians(90)}" effort="0" velocity="${radians(75)}" />
```

- **Faire des opérations logiques (if, unless)**

```
<xacro:if value="<expression>">
  <... some xml code here ...>
</xacro:if>
<xacro:unless value="<expression>">
  <... some xml code here ...>
</xacro:unless>
```

Les Macro Xacro

Créer des blocs / fonctions

- Macro avec des paramètres
- Possibilités d'être appelé plusieurs fois
- Possibilités d'être importé dans d'autre fichiers

```
<xacro:include filename="$(find package)/other_file.xacro" />
<xacro:include filename="other_file.xacro" />
<xacro:include filename="$(cwd)/other_file.xacro" />
```

```
<xacro:macro name="leg" params="prefix reflect">
    <link name="${prefix}_leg">
        <visual>
            <geometry>
                <box size="${leglen} 0.1 0.2"/>
            </geometry>
            <origin xyz="0 0 -${leglen/2}" rpy="0 ${pi/2} 0"/>
            <material name="white"/>
        </visual>
        <collision>
            <geometry>
                <box size="${leglen} 0.1 0.2"/>
            </geometry>
            <origin xyz="0 0 -${leglen/2}" rpy="0 ${pi/2} 0"/>
        </collision>
        <xacro:default_inertial mass="10"/>
    </link>

    <joint name="base_to_${prefix}_leg" type="fixed">
        <parent link="base_link"/>
        <child link="${prefix}_leg"/>
        <origin xyz="0 ${reflect*(width+.02)} 0.25" />
    </joint>
    <!-- A bunch of stuff cut -->
</xacro:macro>
<xacro:leg prefix="right" reflect="1" />
<xacro:leg prefix="left" reflect="-1" />
```

Démonstration

URDF / Xacro

- Questions ?

Exercice 3

Remplacer un robot dans un URDF

Correction

Manipulation avec ROS

MoveIT! 2

Qu’est-ce que MoveIt ?

Un planificateur de trajectoire « On Line »

Un planificateur de trajectoire abstrait du robot:

- Prend en compte la cinématique du robot au chargement (URDF + SRDF)
- Gestion des autocollisions et des collisions avec l’environnement
- Possibilité de spécifié très précisément le problème (mais alors risque de ne pas trouver de solutions)

Intégration rapide avec les systèmes de contrôle sous ROS (ros_control):

- Peut superviser l’exécution de la trajectoire (si trop lent / trop d’écart, peut arrêter)
- Peut lancer les trajectoires avec les API moveit

Des outils:

- Génération automatique de la configuration
- Contrôle du robot « directe »

Architecture MoveIT

Interface utilisateur

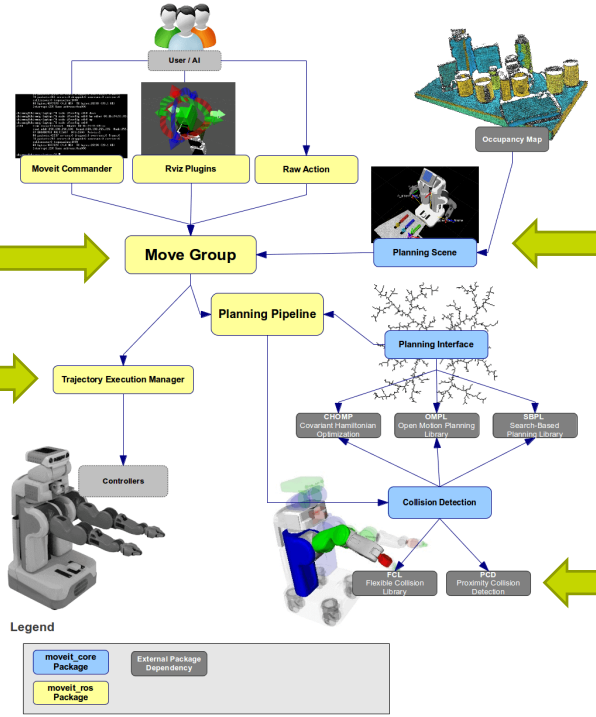
Le cœur de moveit

Interface pour l'exécution

Contenue de l'environnement

Planification de chemin

Détecteur de collision



Les interfaces utilisateurs

Permettent d'utiliser MoveIt en restant sain d'esprit

- Simplifie la création de trajectoire planifié
- Simplifie la modification de la scène (ajout d'objets, etc…)
- Possibilité de lancer la planification et l'exécution en 1 ligne

- **Interface déjà intégré avec Rviz (pratique pour tester, mais concrètement assez difficile à utiliser en situation réelle, peut être pratique pour modifié la scène).**

Démonstration

- **Interface RVIZ**
- **Planification**
- **Obstacle**
- **Exécution Virtuelle**

Le move group

Le Coeur de MoveIt!

- **Synchronise les actions**
  - Requête utilisateur ⇔ Planification ⇔ Scène Environnement ⇔ Exécution
- **Synchronise les information robots/camera/capteurs (Robot Interface)**
  - Ecoute le topic /joint_state pour connaitre la position actuelle du robot (planification depuis la position actuelle, éxecution de trajectoire, ….)
- **Configuration totale – Charge beaucoup de paramètres**
  - Utilise le paramètre robot_description (URDF)
  - Configuration des planneurs / capteurs / contrôleur

MoveIT2

- Questions ?

Manipulation partie 2

ROS 2 Control

Le package le plus “dur” de ROS

Pourquoi ROS2_control

- **Fournir un niveau d’abstraction entre le hardware (spécifique) et les fonctions de commandes comme le suivie de trajectoire (indépendant)**
- **Fonctionne avec les concepts contrôleurs (et de broadcasters) et de ressources contrôlées**
- **Pas besoin de beaucoup de codes pour fonctionner, mais le code doit être précis et bien fait**

- **Contrôleur déjà prêt :**
  - https://github.com/ros-controls/ros2_controllers
  - Position, Vitesse, Effort, Trajectoire, …

Architecture

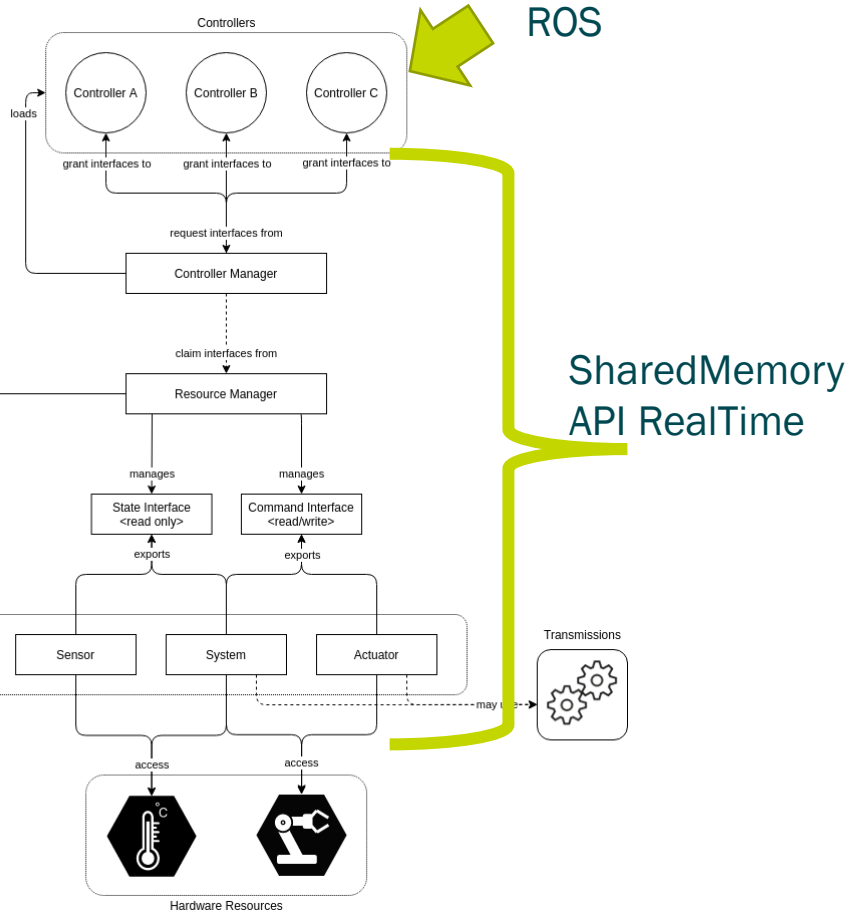
Contrôleur

Contrôleur Manager

Ressource Manager

Interfaces « Ressource »

Contrôleur du robot
(PLC / Microcontrôleur/ Baie)



ROS 2 CONTROL

- Questions ?

Exercice 4

Déplacer un robot avec RVIZ

Configurer un robot avec MoveIT!

Correction

Bilan

Vu aujourd'hui

Les bases de ROS

Les concepts clefs

Les outils principaux

Les bases de la manipulation avec ROS

Les outils pour la manipulation avec ROS

ROS 2

- Questions ?

LE FUTUR
DE VOS USINES

IRT JULES VERNE

Chemin du Chaffault,
44 340 Bouguenais, France
contact@ irt-jules-verne.fr

WWW.IRT-JULES-VERNE.FR

Rejoignez-nous sur :