

Université de Technologie de Belfort-Montbéliard

Département IMSI / FISE Systèmes industriels

Filière Ingénierie Numérique de Process

# Compte-rendu de projet

**Sujet : Première expérience avec ROS-Industrial**

*Semestre A21*

*UV TW53 : Projet industriel*

Réalisé par HUMBERT Emeric



utbm

---

université de technologie  
Belfort-Montbéliard

## Table des matières

<b>I) Introduction.....</b>	<b>3</b>
<b>II) Présentation du projet .....</b>	<b>4</b>
<b>III) Présentation de ROS .....</b>	<b>5</b>
<b>1) Robot Operating System.....</b>	<b>5</b>
<b>2) ROS-Industrial .....</b>	<b>6</b>
<b>3) Move-It.....</b>	<b>7</b>
<b>IV) Installation des logiciels.....</b>	<b>7</b>
<b>1) Ubuntu .....</b>	<b>7</b>
<b>2) ROS.....</b>	<b>8</b>
<b>3) ROS-Industrial .....</b>	<b>9</b>
<b>4) Move-it.....</b>	<b>9</b>
<b>5) Répertoire Universal Robot .....</b>	<b>9</b>
<b>V) Configuration.....</b>	<b>10</b>
<b>VI) Simulation numérique .....</b>	<b>19</b>
<b>VII) Essais sur robot .....</b>	<b>21</b>
<b>VIII) Conclusion .....</b>	<b>26</b>
<b>IX) Bibliographie .....</b>	<b>27</b>

## I) Introduction

Dans le cadre du suivi de la filière Ingénierie Numérique de Process dans la branche des systèmes industriels à l'Université de Technologie de Belfort Montbéliard, nous avons beaucoup été sensibilisés à l'importance de l'essor de l'industrie 4.0 dans les prochaines années.

L'industrie 4.0 est l'application des nouvelles technologies du 21<sup>e</sup> siècle à l'industrie. Parmi ces nouvelles technologies on peut trouver l'IOT (Internet Of Things), la fabrication additive, la réalité augmentée, le big data, etc... Des outils qui permettent d'améliorer la performance des industries en termes de qualité de produit, de réactivité face aux problèmes, de délais, etc.

C'est dans ce contexte qu'est né ce projet pédagogique qui consiste à la réalisation d'une première expérience avec le logiciel ROS (Robot Operating System) et de son extension, ROS-Industrial. Le système ROS étant un ensemble d'outil informatique permettant le développement de logiciels pour la robotique, celui-ci correspond parfaitement à la philosophie de l'industrie 4.0.

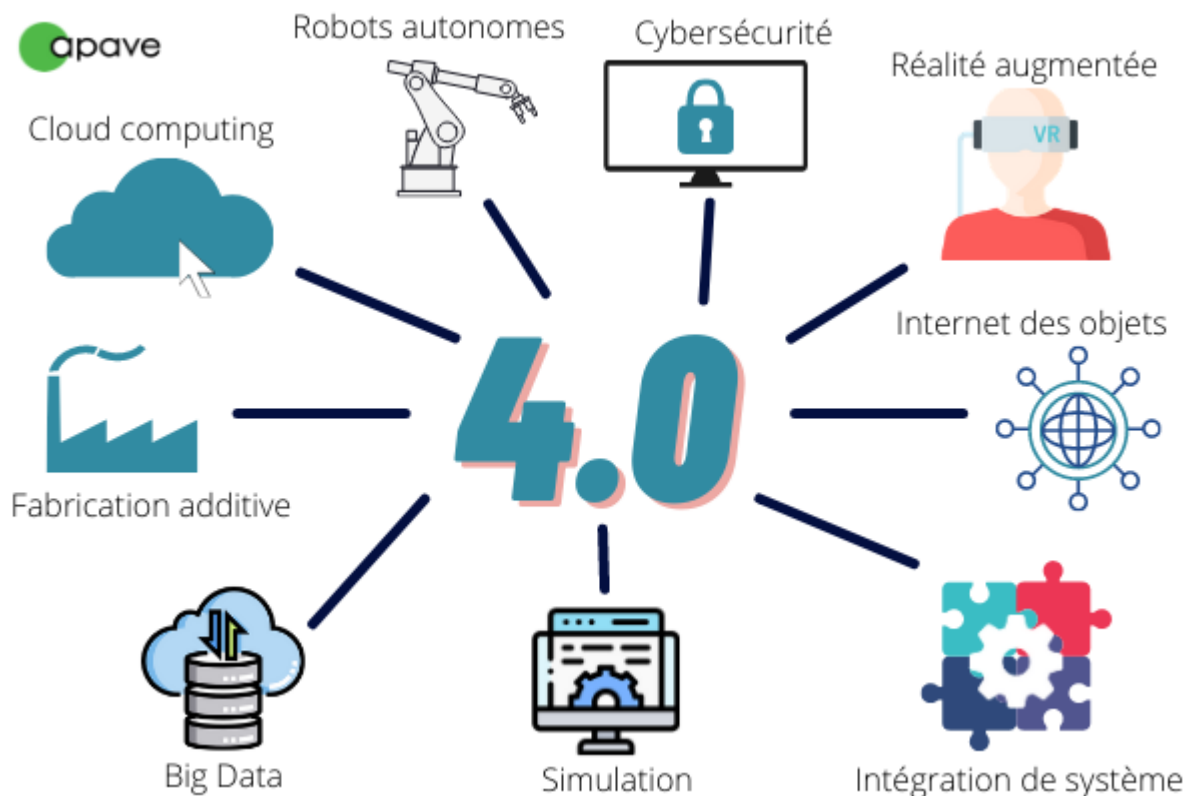


Figure 1 : Illustration de l'industrie 4.0

## II) Présentation du projet

Le but du projet sera de manipuler un robot Universal UR3 disponible en salle de robotique à l'UTBM à l'aide de ROS-Industrial. Il faudra d'abord manipuler un modèle numérique du robot grâce à une simulation puis le vrai robot. Le but est de le manipuler de deux manières différentes : à l'aide de points prédéfinis et à l'aide de la souris de l'ordinateur.



Figure 2 : Robot UR3

Le projet se déroulera en trois grandes parties :

La première partie consistera à installer tous les éléments nécessaires à l'utilisation de ROS. Il faudra également se renseigner sur tous ces éléments (utilité, fonctionnement, ...). Pour ma part, je vais également découvrir le système d'exploitation Linux, préférable pour l'utilisation de ROS.

Pour la deuxième partie, nous passerons à la réalisation de la simulation numérique. Le but sera alors d'intégrer un robot dans l'outil de simulation de ROS(Gazebo) et de le manipuler par deux techniques différentes : avec la souris puis avec des points définis par avance.

Enfin, la dernière partie consistera à la réalisation des tests en salle de robotique. Le but sera de manipuler le vrai robot UR3 à l'aide de la souris et avec des points prédéfinis.

Voici le planning théorique du projet :

Tâche	Semaine	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51	52	1	2	3
<b>Phase 0</b>																				
Choix du sujet		■																		
Présentation du projet		■																		
<b>Phase 1</b>																				
Découverte du sujet et réalisation du cahier des charges			■																	
Installation des logiciels			■																	
Premier tests de ROS						■														
Rendu du premier livrable							■													
<b>Phase 2</b>																				
Importation d'un modèle ABB								■												
Faire bouger le robot avec la souris									■											
Faire bouger le robot avec un code											■									
Rendu du deuxième livrable													■							
<b>Phase 3</b>																				
Essais en salle de robotique															■					
Rendu final																		■		

Figure 3 : Planning Gantt

### III) Présentation de ROS

#### 1) Robot Operating System

**Robot Operating System (ROS)** est un ensemble d'outils informatiques dédiés au développement de logiciel pour la robotique. Ces outils se présentent sous la forme de logiciels libres dit "open-source". Le système ROS a été développé en 2007 par la société américaine Willow Garage. Son développement est aujourd'hui dirigé par l'Open Robotics.

ROS est un méta-système d'exploitation qui peut fonctionner sur un ou plusieurs ordinateurs et qui propose plusieurs fonctionnalités :

- Abstraction du matériel utilisé (peu importe la marque du robot)
- Contrôle des périphériques de bas niveau
- Mise en œuvre de fonctionnalités couramment utilisées (modules réutilisables)
- Transmission de messages entre les processus
- Gestion des packages installés

Malgré ces fonctionnalités, ROS n'est en aucun cas un système d'exploitation.

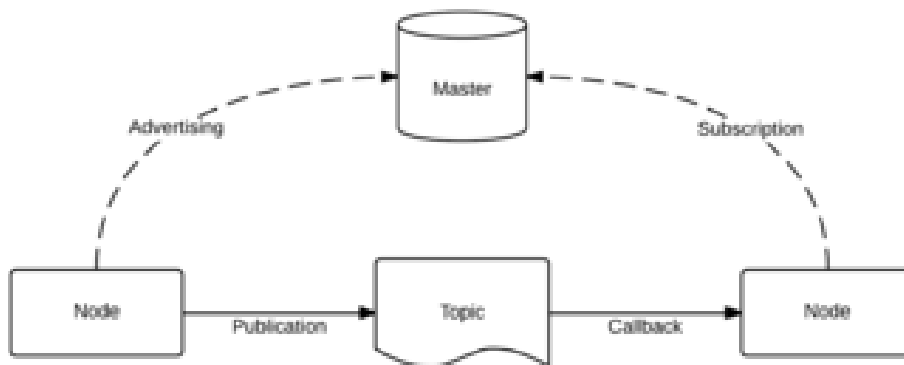
Le logiciel ROS peut se diviser en trois parties :

- Les outils utilisés pour créer, lancer et distribuer des logiciels basés sur ROS (roscore, roslaunch, catkin)
- Les clients ROS pour des langages : roscpp (C++), rospy (Python)
- Les packages contenant des programmes ROS utilisant un ou plusieurs clients ROS

ROS dispose de plusieurs éléments qui facilitent le développement d'applications modulables et souple pour la robotique. Le logiciel dispose d'une architecture de communication inter-processus et inter-machine, d'un serveur de paramètre, d'un système d'enregistrement, d'un système de test et d'un simulateur (gazebo).

Mais passons à un exemple concret pour bien comprendre les atouts de ROS. Imaginons qu'un industriel disposant déjà d'un parc robotique reçoit du nouveau matériel robotique d'une autre marque. Le directeur d'usine a deux choix : il peut utiliser le logiciel constructeur, qui offre des fonctionnalités limitées lorsqu'on cherche à créer des programmes modulables et adaptatifs. Par exemple, l'utilisation de véhicules à guidage automatique en combinaison avec des robots de poses automatique. Le logiciel constructeur (dont le code est bloqué), ne proposant que des tâches préprogrammées, limitera très vite les possibilités d'utilisation. Un autre choix s'offre au directeur : utiliser ROS. ROS étant un code open source, l'utilisateur peut modifier à sa guise le logiciel du robot et cela augmente considérablement les possibilités de création de programmes, d'autant plus avec les nombreuses fonctionnalités vues plus tôt. De plus, le système ROS communique avec le protocole TCP/IP, qui est normalisé et qui permet la communication avec le parc robotique déjà existant dans l'usine.

Voici comment fonctionne ROS :



Les processus ROS sont appelés des nodes. Chaque node peut communiquer avec d'autres nodes via des Topics. La connexion entre ces nodes est gérée par un maître :

- Un premier node avertit le maître qu'il veut partager des données.
- Un deuxième node avertit le maître qu'il souhaite avoir accès à une donnée.
- Le maître crée ensuite une connexion entre les deux nodes.
- Les deux nodes peuvent communiquer entre eux.

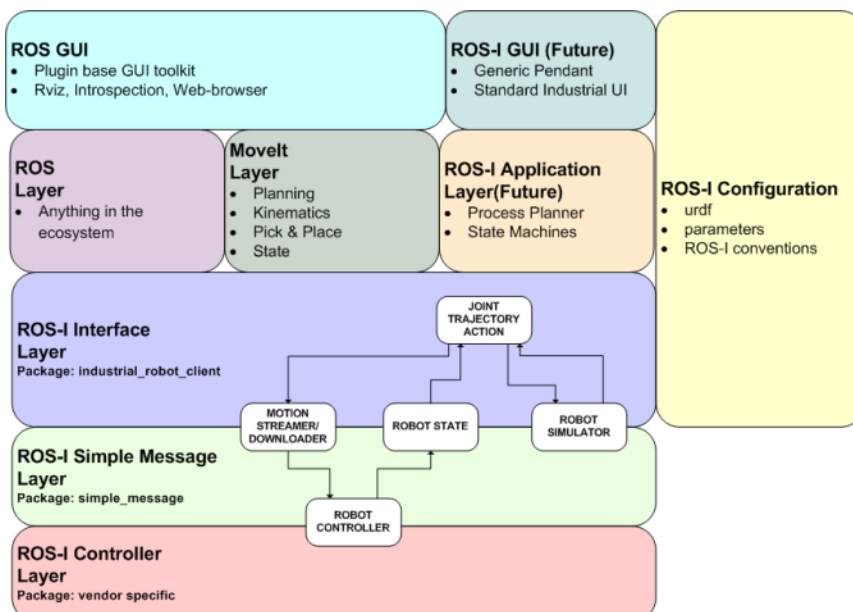
Les messages qui sont communiqués entre les nodes sont standardisés ce qui permet une grande flexibilité et une communication inter-machin si celles-ci ont le même maître.

## 2) ROS-Industrial

ROS-Industrial est également un projet dit open source qui augmente les capacités de ROS et notamment dans le domaine de la robotique et de l'automatisme industriel. Le répertoire ROS-Industrial ajoute notamment des interfaces pour les manipulateurs industriels communs (Universal, ABB, Fanuc), pinces, détecteurs, ...



Voici l'architecture de ROS-Industrial :



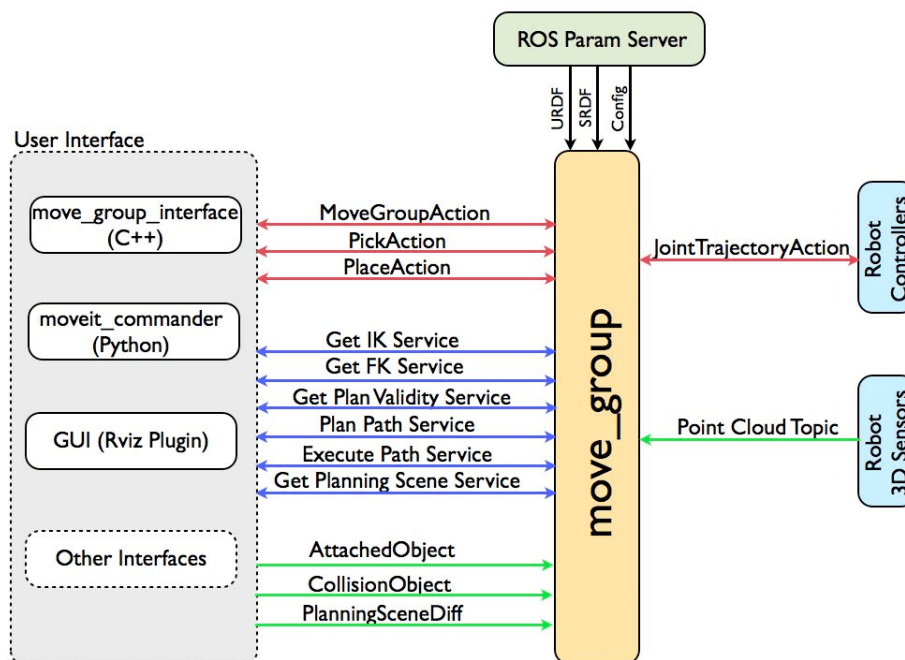
ROS-Industrial High Level Architecture - Rev 0.02.vsd

### 3) Move-It

Le package Moveit est la bibliothèque de ROS-Industrial qui sert à manipuler un bras robotique de base. Moveit propose les fonctionnalités suivantes :

- Planification de mouvement
- Manipulation de robot
- Cinématique inverse du robot
- Contrôle en temps réel du robot
- Visualisation 3D de son environnement avec l'ajout de détecteurs de distance
- Gestion des collisions

Voici l'architecture de Moveit :



## IV) Installation des logiciels

### 1) Ubuntu

Pour pouvoir utiliser ROS, il va falloir installer Linux. J'ai choisi d'utiliser le système d'exploitation Ubuntu, réputé abordable pour les débutants. Plusieurs choix sont possibles : vous pouvez utiliser Ubuntu en dualboot avec Windows (choix que j'ai fait), utiliser un ordinateur virtuel ou utiliser un avec le système d'exploitation déjà présent.

Je ne détaillerai pas l'installation d'Ubuntu ici, je placerai le lien qui m'a aidé dans la bibliographie.

#### Mise en garde :

Un élément est à prendre en compte avant d'installer Ubuntu : chaque version de ROS correspond à une version de Ubuntu comme on peut le voir sur l'image ci-dessous.

ROS



ROS2



Il faudra donc installer la bonne version de Ubuntu selon la version de ros que vous voulez utiliser. De plus, ROS-Industrial n'était pas disponible sur la version "Noetic" de ROS lorsque j'ai réalisé mon projet. J'ai donc perdu beaucoup de temps et j'ai dû réinstaller une version antérieure de Ubuntu pour pouvoir installer la version antérieure de ROS "Melodic" et pouvoir utiliser ROS-Industrial qui était disponible dessus.

## 2) ROS

Comme je vais utiliser la version "Melodic" de ROS, j'ai installé la version 18.04 de Ubuntu. Nous pouvons lancer l'installation. Pour commencer, il faut configurer Ubuntu pour qu'il accepte les répertoires "Restricted", "Universe" et "Multiverse".

Voici les commandes à entrer ensuite dans le terminal :

Cette commande configure votre ordinateur pour accepter le logiciel ROS :

```
sudo sh -c 'echo "deb http://packages.ros.org/ros/ubuntu
$(lsb_release -sc) main" > /etc/apt/sources.list.d/ros-latest.list'
```

```
sudo apt install curl # if you haven't already installed curl
curl -s https://raw.githubusercontent.com/ros/rosdistro/master/ros.asc |
sudo apt-key add -
```

Installation de ROS Melodic (ici la version desktop-full pour avoir directement Gazebo d'installé) :

```
sudo apt update
```

```
sudo apt install ros-melodic-desktop-full
```

Configurer l'environnement :

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



Enfin, pour faciliter l'installation d'autres packages :

```
sudo apt install python-rosdep python-rosinstall python-rosinstall-  
generator python-wstool build-essential
```

```
sudo apt install python-rosdep
```

```
sudo rosdep init
```

```
rosdep update
```

Création d'un espace de travail ROS :

```
mkdir -p ~/catkin_ws/src
```

```
cd ~/catkin_ws/
```

```
catkin_make
```

L'installation de ROS est terminée, si vous souhaitez vous exercer dessus, un lien vers un tutoriel est disponible dans la bibliographie.

### 3) ROS-Industrial

Il suffit d'entrer la commande suivante :

```
apt-get install ros-melodic-industrial-core
```

L'installation de ROS-Industrial est terminée, si vous souhaitez vous exercer dessus, un lien vers un tutoriel est disponible dans la bibliographie.

### 4) Move-it

Moveit est normalement déjà installé lorsqu'on installe ROS-Industrial. Si ce n'est pas le cas :

Il suffit d'entrer la commande suivante :

```
sudo apt install ros-melodic-moveit
```

L'installation de ROS-Industrial est terminée, si vous souhaitez vous exercer dessus, un lien vers un tutoriel est disponible dans la bibliographie.

### 5) Répertoire Universal Robot

Il suffit d'entrer la commande suivante :

```
cd catkin_ws/src
```

```
git clone -b melodic-devel https://github.com/ros-  
industrial/universal\_robot.git
```

## V) Configuration

Le but à présent est de créer une simulation ROS avec Moveit sur le simulateur Gazebo.

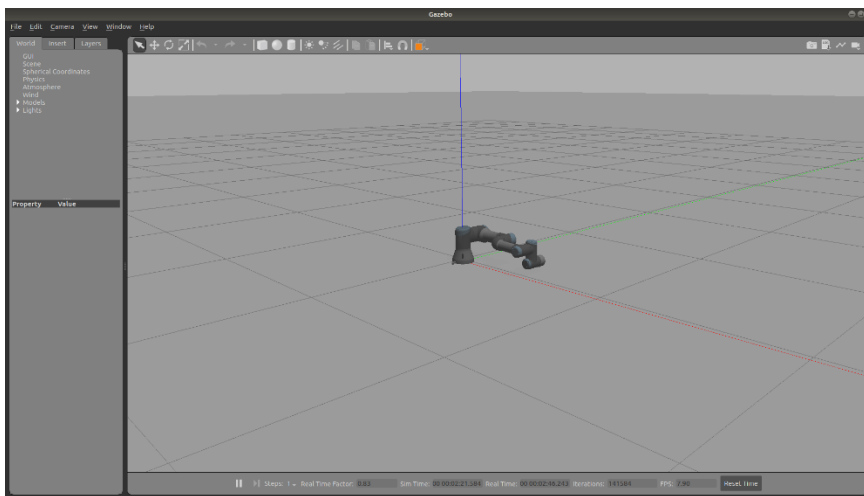
**Attention !**

**A chaque fois qu'on ouvre un nouveau terminal et qu'on veut lancer une commande ROS, il faudra d'abord taper la commande suivante :**

```
source catkin_ws/devel/setup.bash
```

Pour commencer, nous allons lancer le modèle 3D du robot UR3 dans Gazebo. Pour cela, il faut taper la commande suivante :

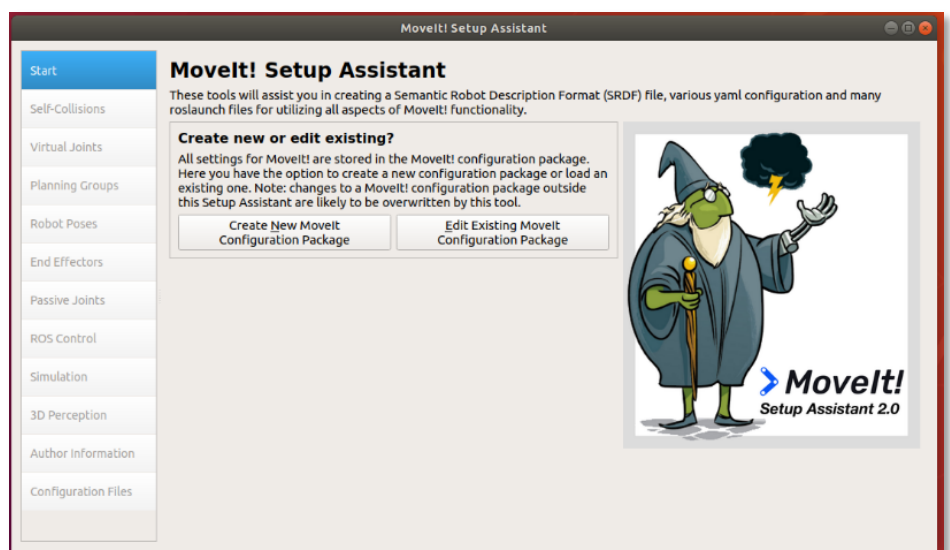
```
roslaunch ~/catkin_ws/src/universal_robot/ur_gazebo/ur3.launch
```



Nous avons à présent le modèle 3D du robot. Néanmoins, on ne peut pas le manipuler.

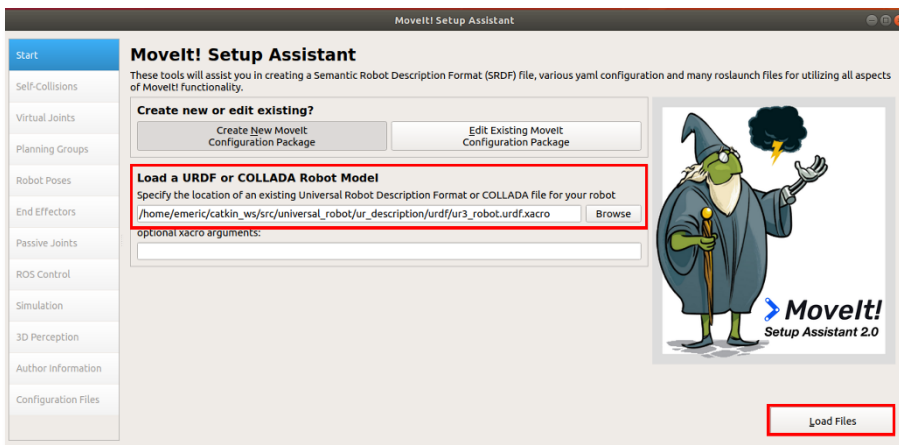
Pour cela, nous allons lancer l'assistant de Moveit dans un nouveau terminal :

```
roslaunch  
moveit_setup_assistant  
setup_assistant.launch
```



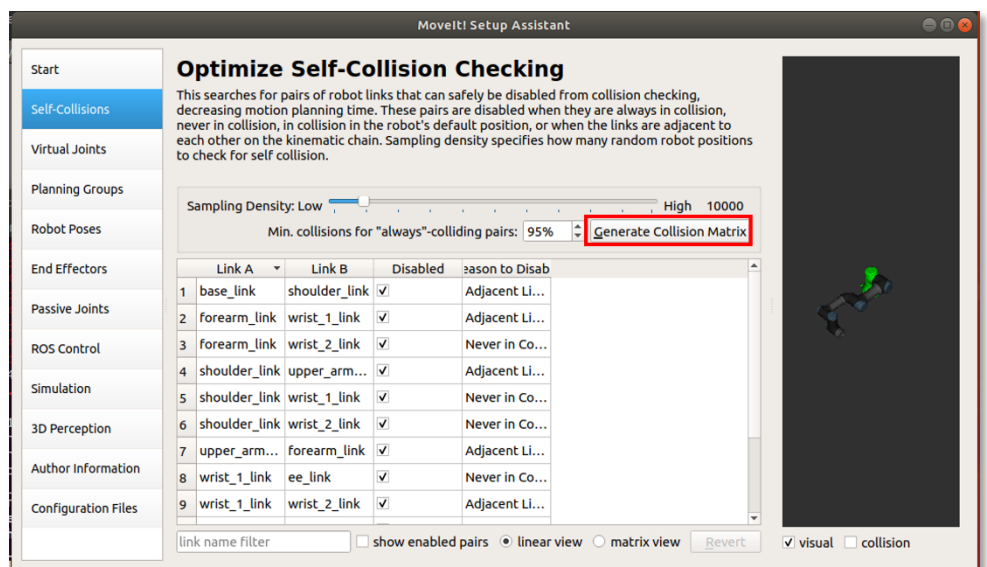
Il faut à présent configurer notre robot :

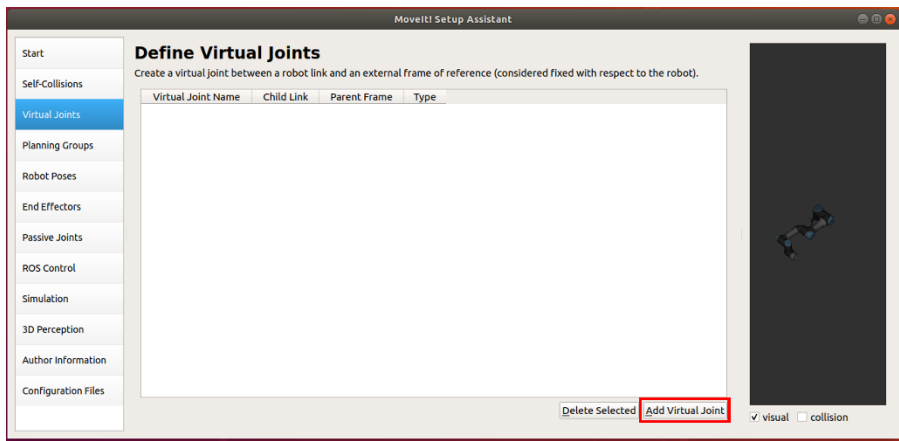
Créer un nouveau package Moveit. Si vous voulez le modifier par la suite, cliquer sur le bouton d'édition et indiquer le chemin du dossier de la configuration Moveit que vous voulez modifier.



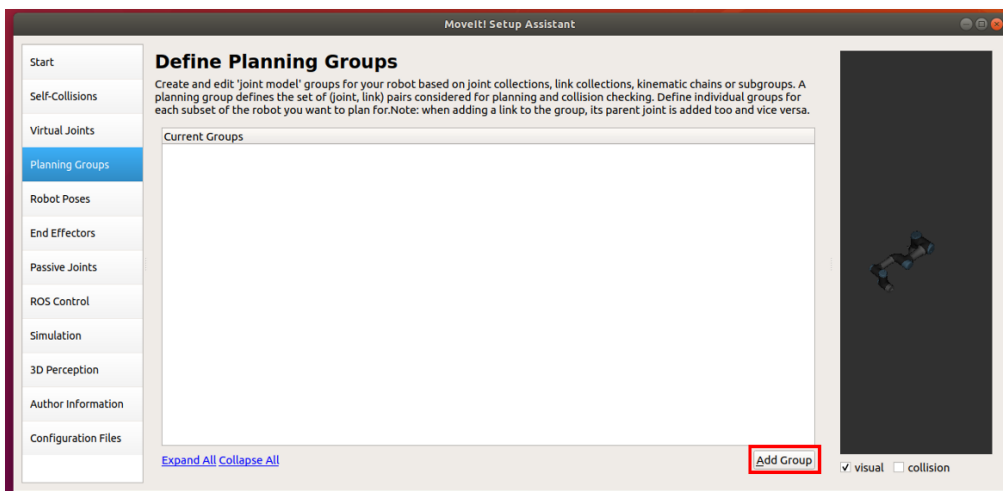
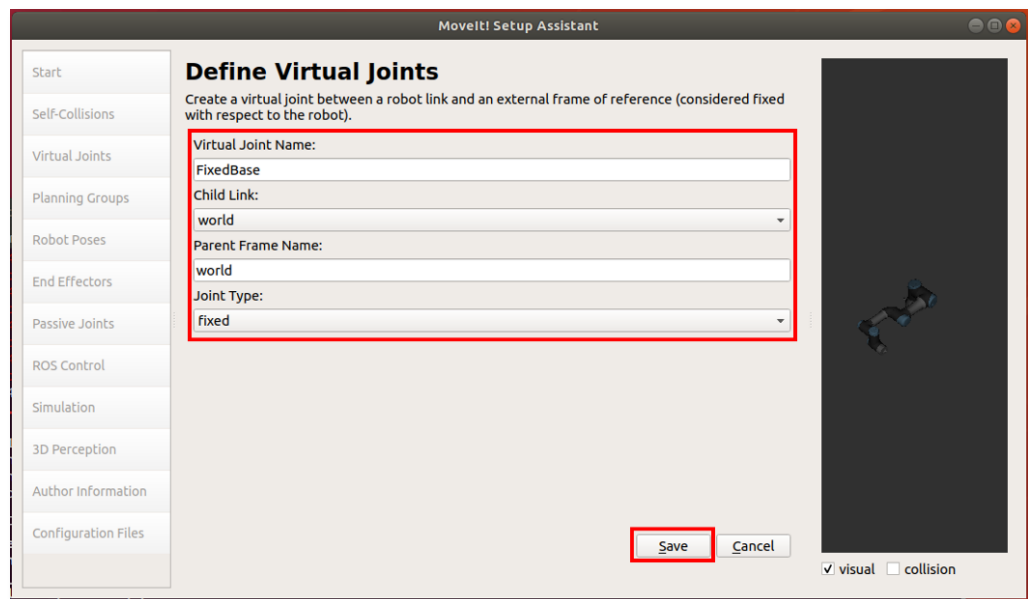
Entrer la localisation du fichier de description du robot.

Générer la matrice de collision.

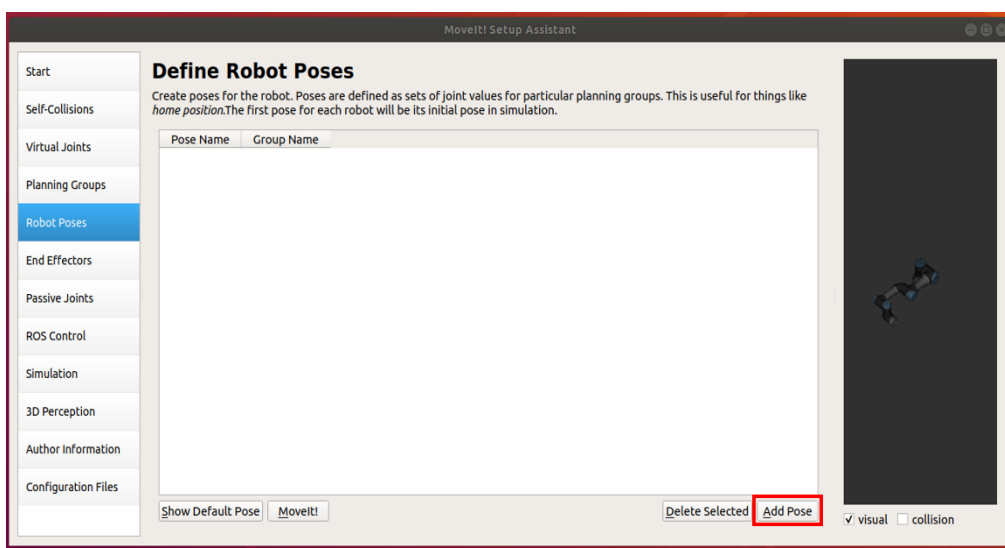
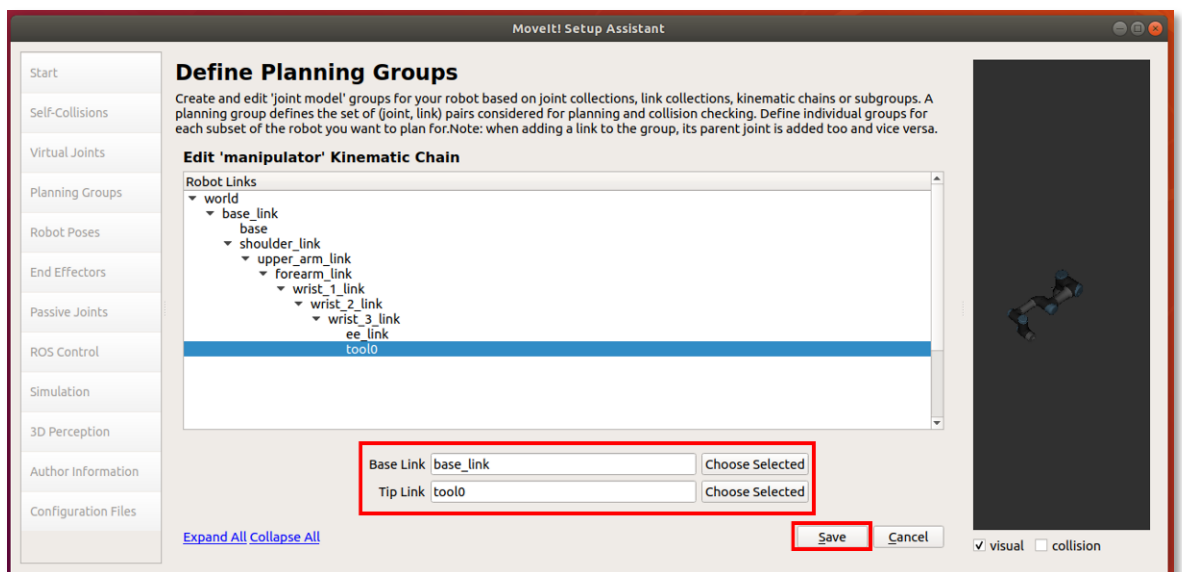
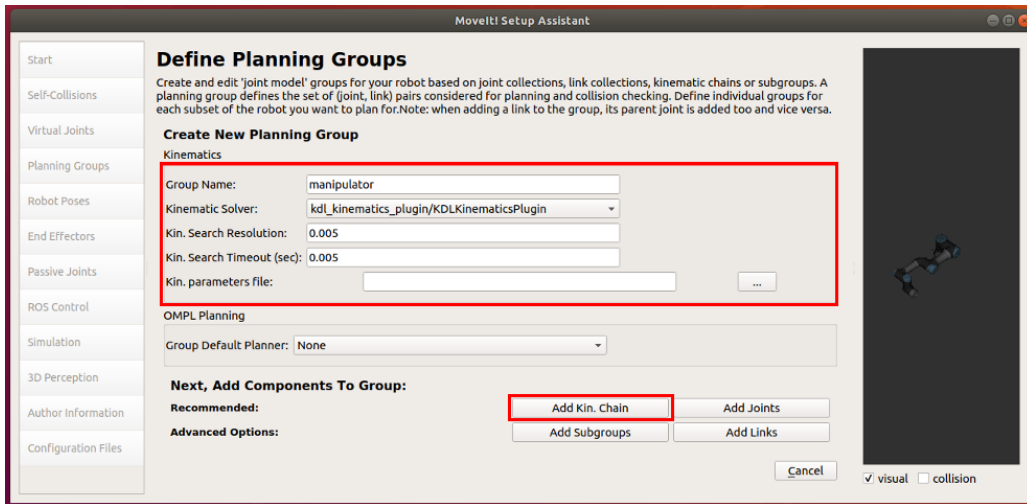




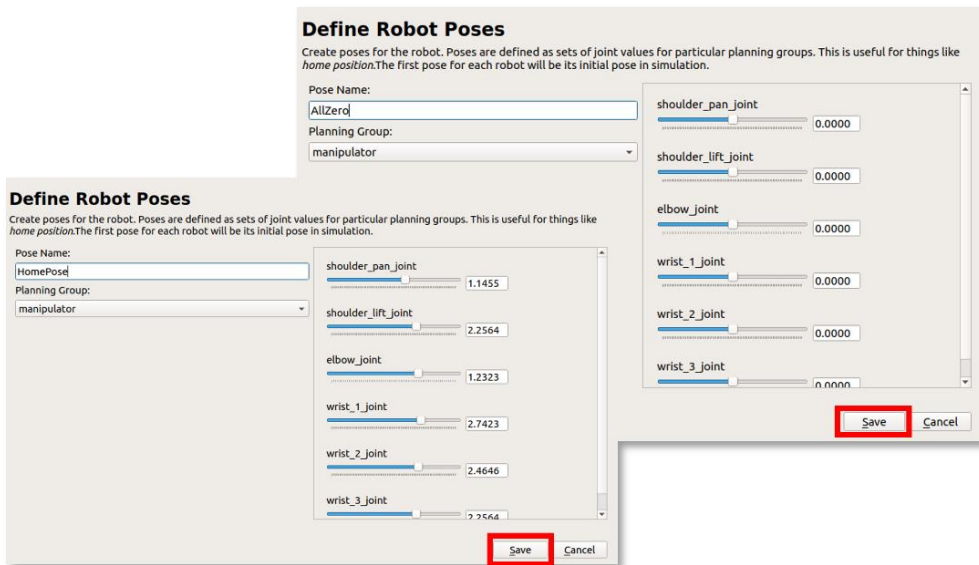
Ajouter un lien virtuel entre la base du robot et le monde aux alentours.



Définir un groupe d'articulation du robot que vous voulez manipuler (dans notre cas, de la base à l'effecteur).

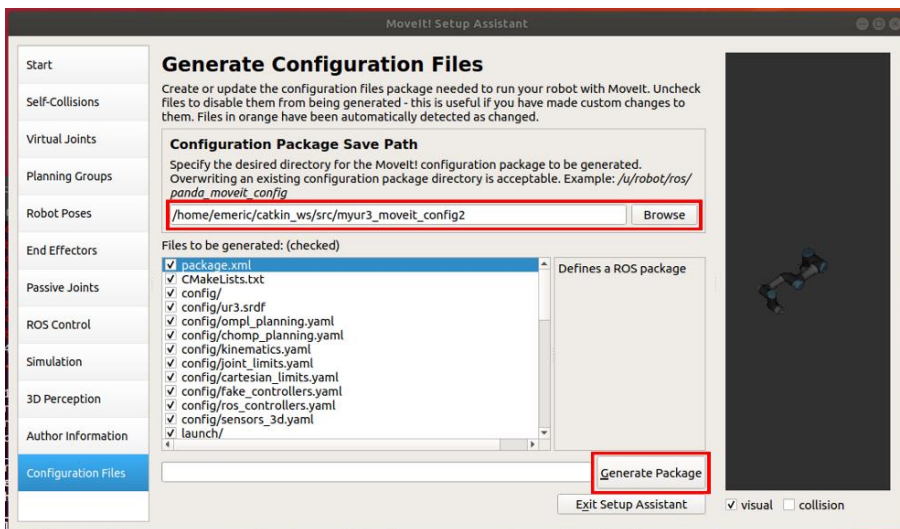
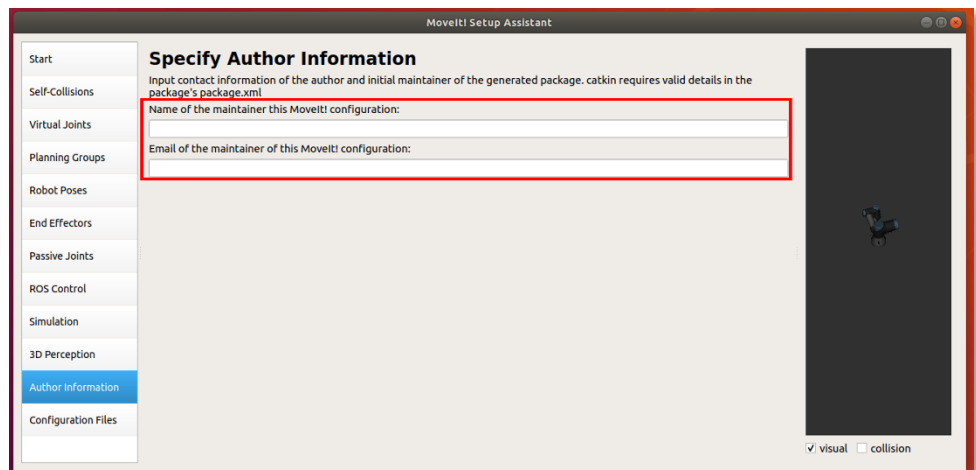


Définir des poses du robot.



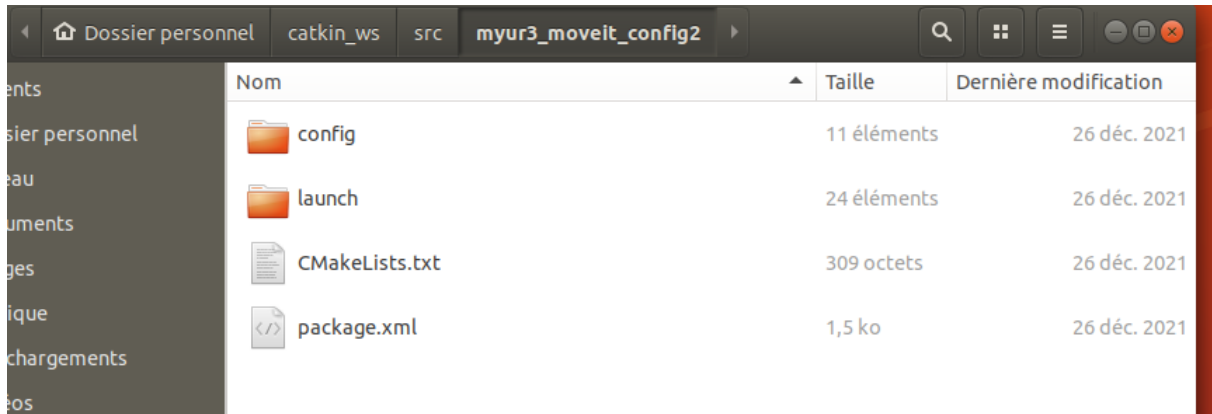
Dans mon cas, j'ai défini une pose avec toutes les articulations en position zéro et une "HomePose", choisie au hasard.

Entrer un nom et une adresse mail.



Créer un dossier "NomRobot\_moveit\_config" dans le dossier catkin\_ws/src et générer le package.

Voici ce qu'a généré Moveit :

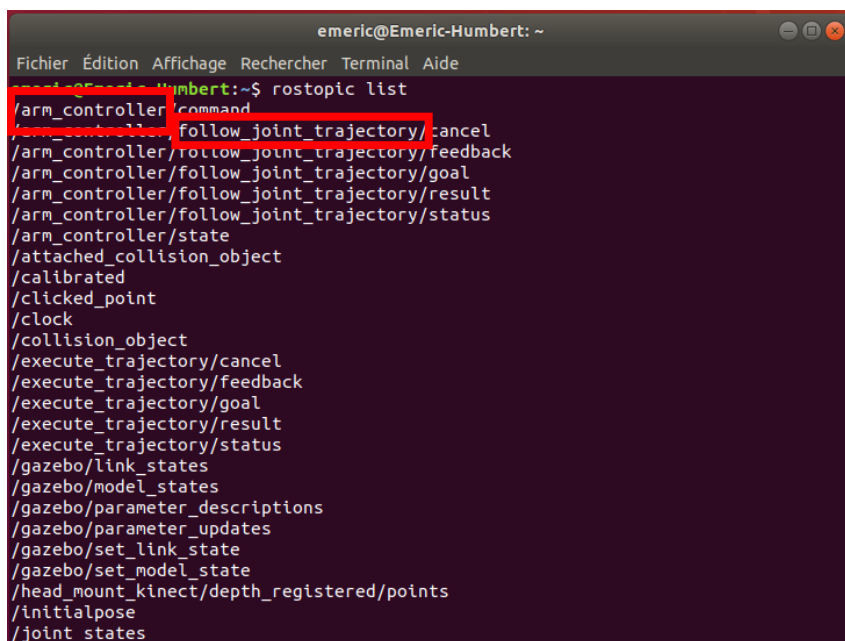


Pour continuer, nous allons créer deux fichiers dans le dossier "config" :

- "controllers.yaml" :

Ce fichier va définir comment les articulations du vrai robot vont être contrôlées. Tout d'abord, il faut entrer le type de serveur d'action que nous utiliserons pour contrôler le robot. Pour le connaître, taper la formule suivante dans le terminal :

```
rostopic list
```



```
emeric@Emeric-Humbert: ~  
Fichier Édition Affichage Rechercher Terminal Aide  
emeric@Emeric-Humbert:~$ rostopic list  
/arm_controller/command  
/arm_controller/follow_joint_trajectory/cancel  
/arm_controller/follow_joint_trajectory/feedback  
/arm_controller/follow_joint_trajectory/goal  
/arm_controller/follow_joint_trajectory/result  
/arm_controller/follow_joint_trajectory/status  
/arm_controller/state  
/attached_collision_object  
/calibrated  
/clicked_point  
/clock  
/collision_object  
/execute_trajectory/cancel  
/execute_trajectory/feedback  
/execute_trajectory/goal  
/execute_trajectory/result  
/execute_trajectory/status  
/gazebo/link_states  
/gazebo/model_states  
/gazebo/parameter_descriptions  
/gazebo/parameter_updates  
/gazebo/set_link_state  
/gazebo/set_model_state  
/head_mount_kinect/depth_registered/points  
/initialpose  
/joint_states
```

Il faut ensuite entrer les noms des articulations. Ceux-ci se trouvent dans le fichier `catkin_ws/src/universal_robot/ur_description/urdf/NomRobot.urdf.xacro` :

Ouvrir ▾




```
</geometry>
</collision>
<xacro:cylinder_inertial radius="0.075" length="0.038" mass="{base_mass}">
  <origin xyz="0.0 0.0 0.0" rpy="0 0 0" />
</xacro:cylinder_inertial>
</link>

<joint name="{prefix}shoulder_pan_joint" type="revolute">
  <parent link="{prefix}base_link" />
  <child link = "{prefix}shoulder_link" />
  <origin xyz="0.0 0.0 {shoulder_height}" rpy="0.0 0.0 0.0" />
  <axis xyz="0 0 1" />
  <xacro:unless value="{joint_limited}">
    <limit lower="{-2.0 * pi}" upper="{2.0 * pi}" effort="330.0" velocity="2.16"/>
  <xacro:if value="{safety_limits}">
    <safety_controller soft_lower_limit="{-2.0 * pi + safety_pos_margin}" soft_upper_limit="{2.0 * pi - safety_pos_margin}" />
  </xacro:if>
</xacro:unless>
```

Nom de la première articulation.

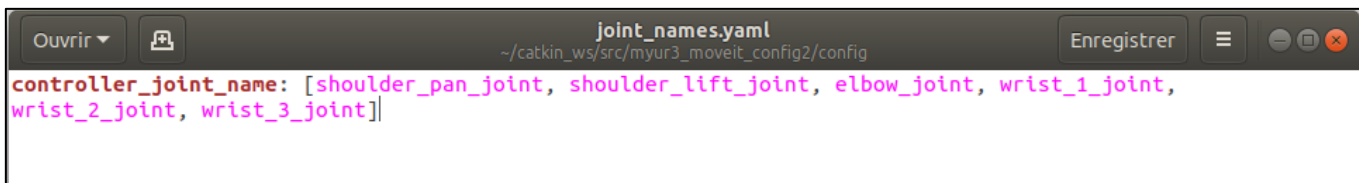
Voici la forme du fichier "controllers.yaml" :



```
controller_list:
- name: arm_controller
  action_ns: "follow_joint_trajectory"
  type: FollowJointTrajectory
  joints: [shoulder_pan_joint, shoulder_lift_joint, elbow_joint, wrist_1_joint, wrist_2_joint, wrist_3_joint]
```

- "Joint\_names.yaml" :

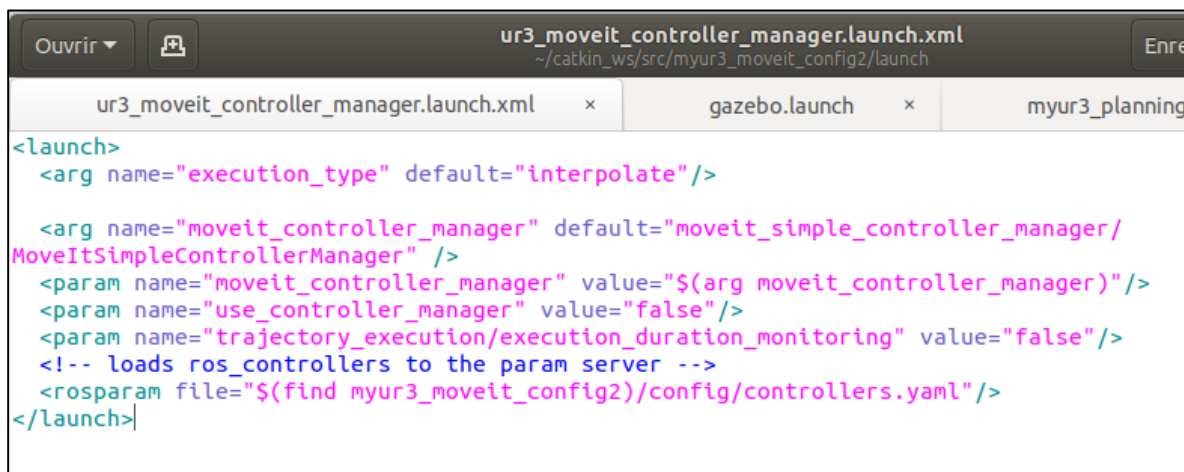
Il suffit d'entrer à nouveau le nom des articulations sous la forme suivante :



```
controller_joint_name: [shoulder_pan_joint, shoulder_lift_joint, elbow_joint, wrist_1_joint, wrist_2_joint, wrist_3_joint]
```

Pour terminer, il nous reste deux fichiers à modifier et un fichier à créer dans le dossier launch de notre configuration :

- Fichier à modifier "NomRobot\_moveit\_controller\_manager.launch.xml"



```
<launch>
  <arg name="execution_type" default="interpolate"/>

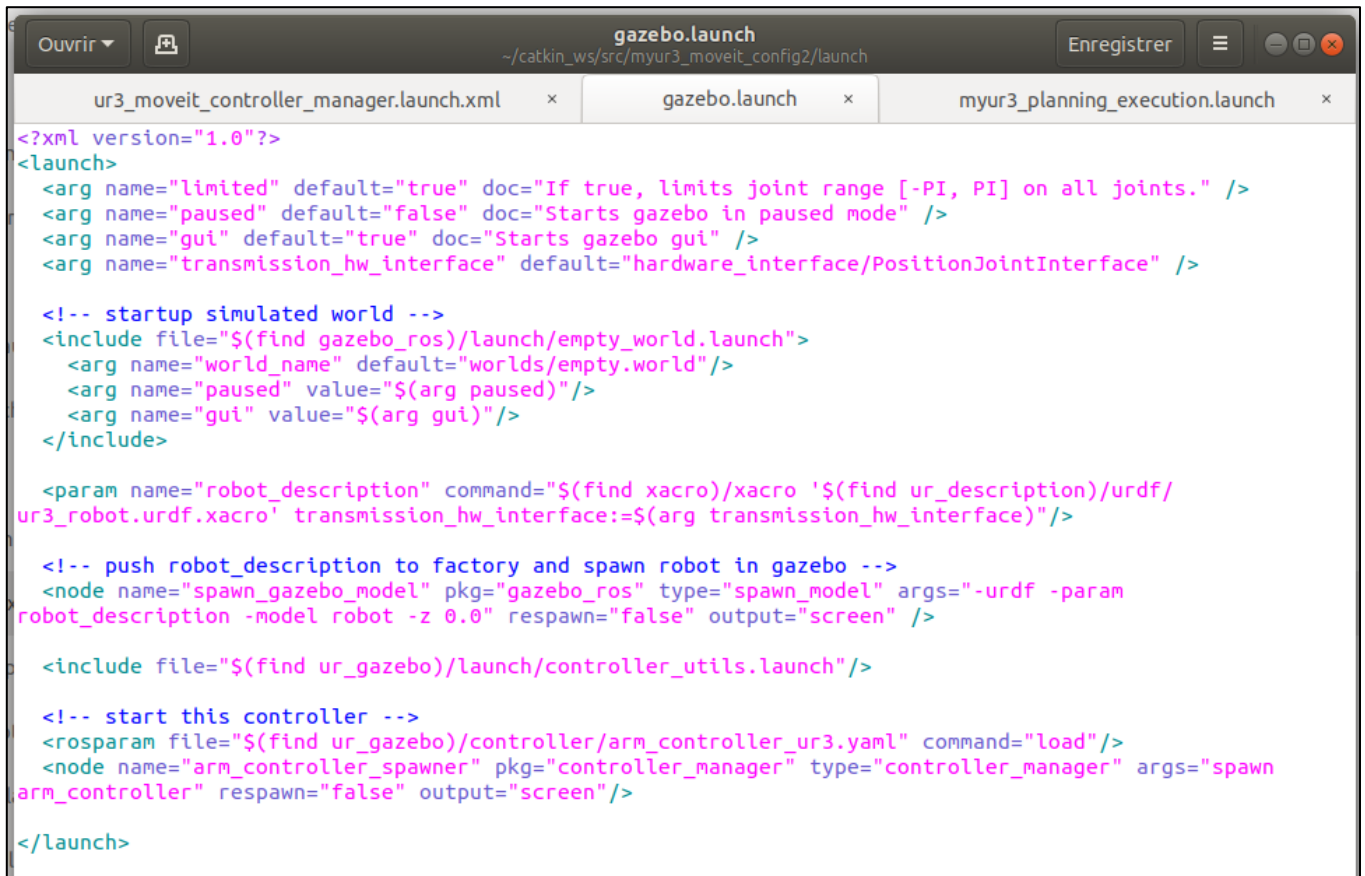
  <arg name="moveit_controller_manager" default="moveit_simple_controller_manager/MoveItSimpleControllerManager" />
  <param name="moveit_controller_manager" value="{arg moveit_controller_manager}" />
  <param name="use_controller_manager" value="false"/>
  <param name="trajectory_execution/execution_duration_monitoring" value="false"/>
  <!-- loads ros_controllers to the param server -->
  <rosparam file="{(find myur3_moveit_config2)/config/controllers.yaml}" />
</launch>
```



La première ligne sert à éviter une erreur. En effet, si je lance la simulation sans cette ligne, cela engendre un message d'erreur car d'autres fichiers envoient un argument nommé "execution\_type" à ce fichier, mais celui-ci ne l'utilise pas. Je créer donc un argument du même nom pour faire marcher le code.

Le reste des lignes servent à charger le fichier "controllers.yaml" et donnent les paramètres pour ce fichier.

- Fichier à modifier "gazebo.launch" :



```
<?xml version="1.0"?>
<launch>
  <arg name="limited" default="true" doc="If true, limits joint range [-PI, PI] on all joints." />
  <arg name="paused" default="false" doc="Starts gazebo in paused mode" />
  <arg name="gui" default="true" doc="Starts gazebo gui" />
  <arg name="transmission_hw_interface" default="hardware_interface/PositionJointInterface" />

  <!-- startup simulated world -->
  <include file="$(find gazebo_ros)/launch/empty_world.launch">
    <arg name="world_name" default="worlds/empty.world"/>
    <arg name="paused" value="$(arg paused)"/>
    <arg name="gui" value="$(arg gui)"/>
  </include>

  <param name="robot_description" command="$(find xacro)/xacro '$(find ur_description)/urdf/ur3_robot.urdf.xacro' transmission_hw_interface:=$(arg transmission_hw_interface)"/>

  <!-- push robot_description to factory and spawn robot in gazebo -->
  <node name="spawn_gazebo_model" pkg="gazebo_ros" type="spawn_model" args="-urdf -param robot_description -model robot -z 0.0" respawn="false" output="screen" />

  <include file="$(find ur_gazebo)/launch/controller_utils.launch"/>

  <!-- start this controller -->
  <rosparam file="$(find ur_gazebo)/controller/arm_controller_ur3.yaml" command="load"/>
  <node name="arm_controller_spawner" pkg="controller_manager" type="controller_manager" args="spawn arm_controller" respawn="false" output="screen"/>
</launch>
```

Ces lignes servent à charger un monde vide dans gazebo, d'y ajouter notre modèle de robot et de charger les contrôleurs du robot. Ce sera ce fichier qui fera la simulation numérique.

- Fichier à créer “*NomRobot\_planning\_execution.launch*”

```

<launch>

<rosparam command="load" file="$(find myur3_moveit_config2)/config/joint_names.yaml"/>

<include file="$(find myur3_moveit_config2)/launch/planning_context.launch">
  <arg name="load_robot_description" value="true"/>
</include>

<node name="joint_state_publisher" pkg="joint_state_publisher" type="joint_state_publisher">
  <param name="/use_gui" value="false"/>
  <rosparam param="/source_list">[/joint_states]</rosparam>
</node>

<include file="$(find myur3_moveit_config2)/launch/move_group.launch">
  <arg name="publish_monitored_planning_scene" value="true"/>
</include>

<include file="$(find myur3_moveit_config2)/launch/moveit_rviz.launch">
  |
</include>

</launch>

```

Ce programme commence par charger des fichiers utiles à la simulation, dont celui que nous avons créé précédemment “joint\_names.yaml”.

Pour le paramètre [/joint\_states] vérifiez, à nouveau avec la commande `rostopic list`, le nom du paramètre.

```

emeric@Emeric-Humbert:~$ rostopic list
/arm_controller/command
/arm_controller/follow_joint_trajectory/cancel
/arm_controller/follow_joint_trajectory/feedback
/arm_controller/follow_joint_trajectory/goal
/arm_controller/follow_joint_trajectory/result
/arm_controller/follow_joint_trajectory/status
/arm_controller/state
/attached_collision_object
/calibrated
/clicked_point
/clock
/collision_object
/execute_trajectory/cancel
/execute_trajectory/feedback
/execute_trajectory/goal
/execute_trajectory/result
/execute_trajectory/status
/gazebo/link_states
/gazebo/parameter_descriptions
/gazebo/parameter_updates
/gazebo/set_link_state
/gazebo/set_model_state
/head_mount_kinect/depth_registered/points
/initialpose
/joint_states

```

Le fichier lance ensuite Moveit pour contrôler notre simulation.

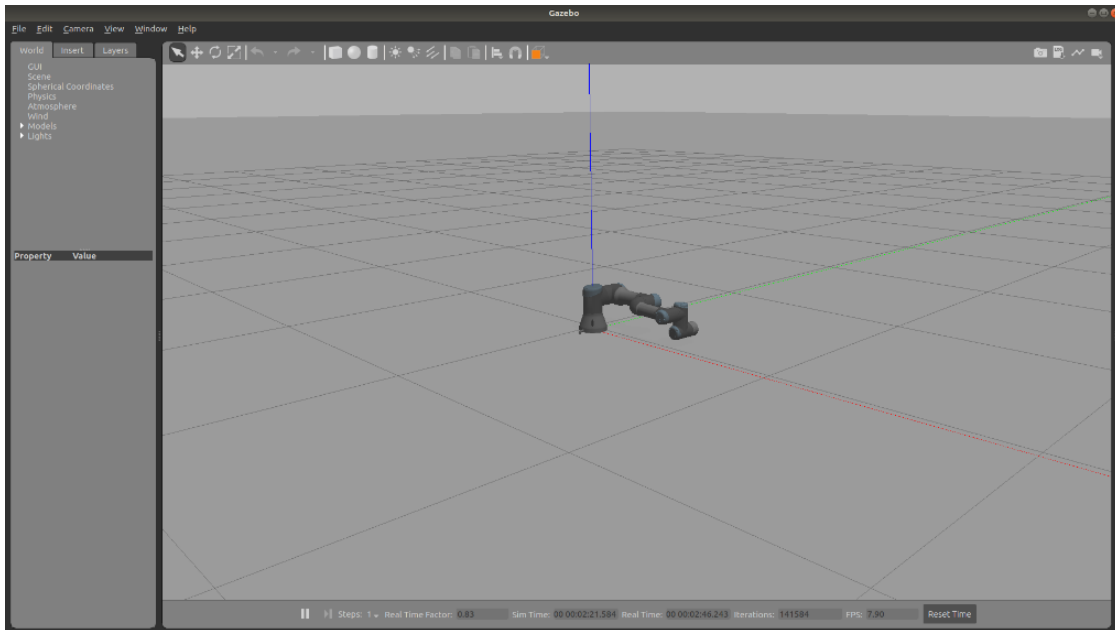
Le paramétrage de nos fichiers sont terminés, nous pouvons passer aux essais.

## VI) Simulation numérique

Pour réaliser la simulation numérique, le fichier Gazebo que nous avons modifié plus tôt servira de remplaçant pour le vrai robot. Nous allons donc commencer par lancer ce fichier.

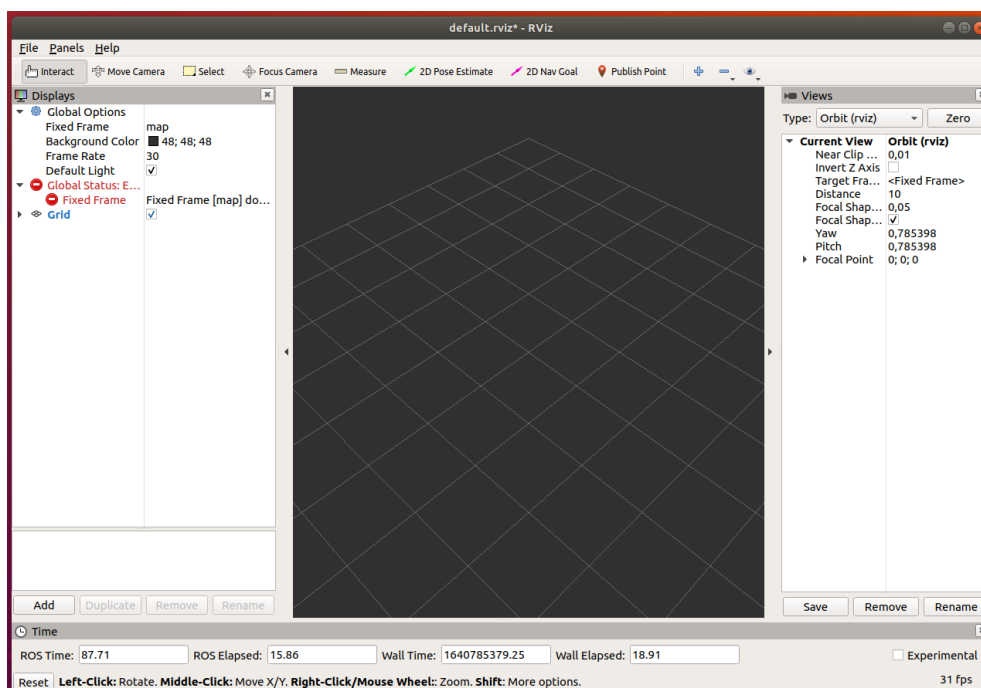
Pour cela, il suffit de taper la formule suivant dans le terminal :

```
roslaunch ~/catkin_ws/src/NomRobot_moveit_config/launch/gazebo.launch
```



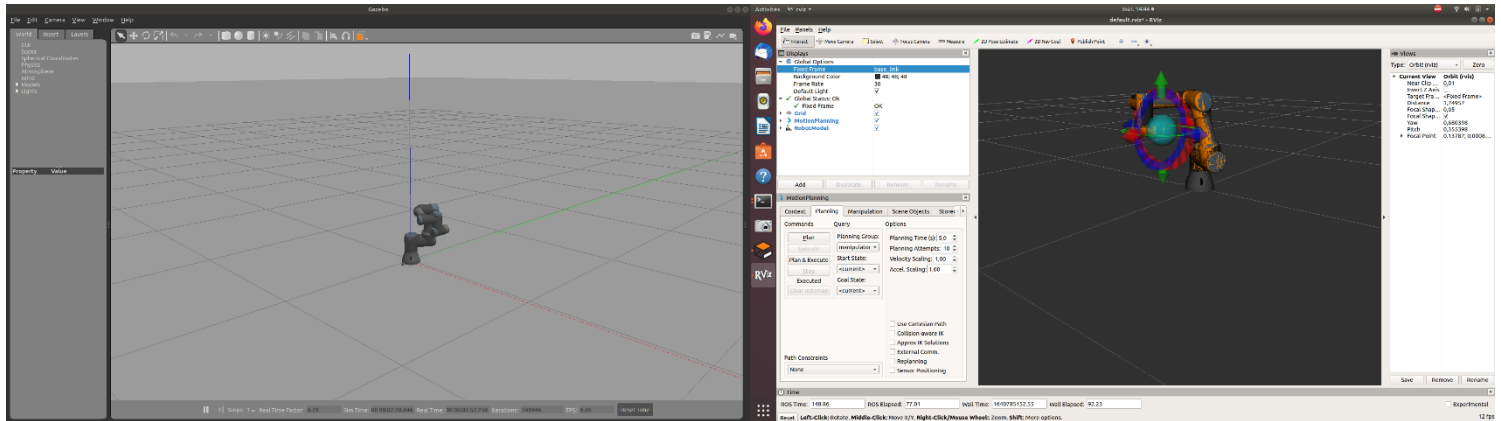
On peut ensuite lancer Moveit :

```
roslaunch  
~/catkin_ws/src/NomRobot_moveit_config/launch/NomRobot_planning_execution.l  
aunch
```



Comme on peut le voir, il faut encore insérer le robot dans Moveit. Pour cela, il faut définir le paramètre "Fixed Frame" sur "base\_link". Ensuite, il faut insérer "MotionPlanning" et "RobotModel" grâce au bouton "Add". "RobotModel" insère dans Moveit le modèle 3D du robot ainsi que sa cinématique. "MotionPlanning" sert à manipuler le robot ou la simulation du robot.

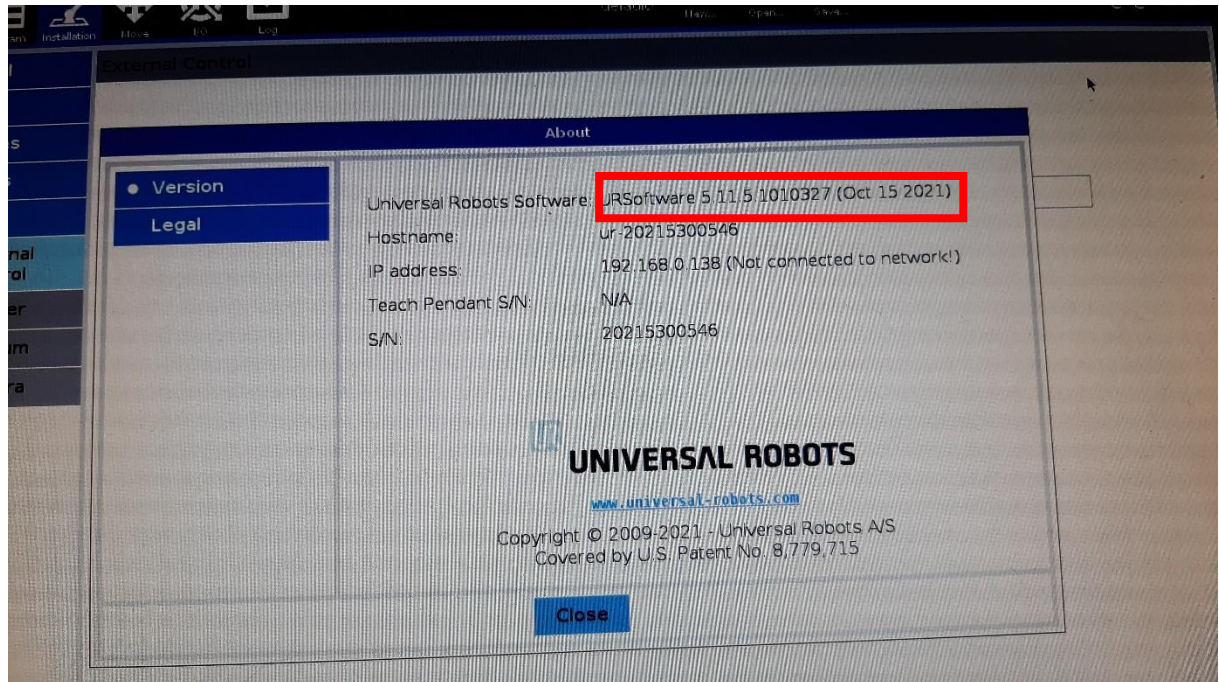
Voici le résultat :



On peut ensuite manipuler le robot de deux manières différentes : On peut mettre le robot dans la position souhaitée en bougeant l'effecteur avec la souris ou en entrant un point prédéfini dans "Goal State". Il suffit ensuite de planifier (plan) et d'exécuter (execute). Le robot dans le simulateur va bouger, c'est le signe que ROS est bien connecté à Moveit.

## VII) Essais sur robot

La première des choses à savoir est la version du logiciel du robot sur le pupitre de commande. En effet, les démarches ne seront pas les mêmes selon qu'il s'agit d'une version V1.5X, V1.8X, V3.X, V5.X. Dans mon cas, il s'agit d'une version V5.X, c'est donc cette version de la démarche que je vais détailler. Je mettrai également en bibliographie les ressources pour les autres versions.



Voici donc la démarche pour manipuler un vrai robot UR3 sous logiciel version V5.X avec ROS-Industrial :

Premièrement, je conseille de créer un deuxième dossier différent de celui de la simulation pour clarifier la chose :

```
source /opt/ros/melodic/setup.bash
mkdir -p catkin_ur/src && cd catkin_ur
```

Télécharger ensuite les ressources nécessaires :

```
git clone
https://github.com/UniversalRobots/Universal_Robots_ROS_Driver.git
src/Universal_Robots_ROS_Driver

git clone -b calibration_devel
https://github.com/fmauch/universal_robot.git src/fmauch_universal_robot
```

Puis, installer les dépendances et construisez l'espace de travail :

```
sudo apt update -qq
rosdep update
```

```
rosdep install --from-paths src --ignore-src -y
```

```
catkin_make
```

```
source devel/setup.bash
```

Créer ensuite un dossier qui accueillera les calibrations de notre robot :

```
cd src
```

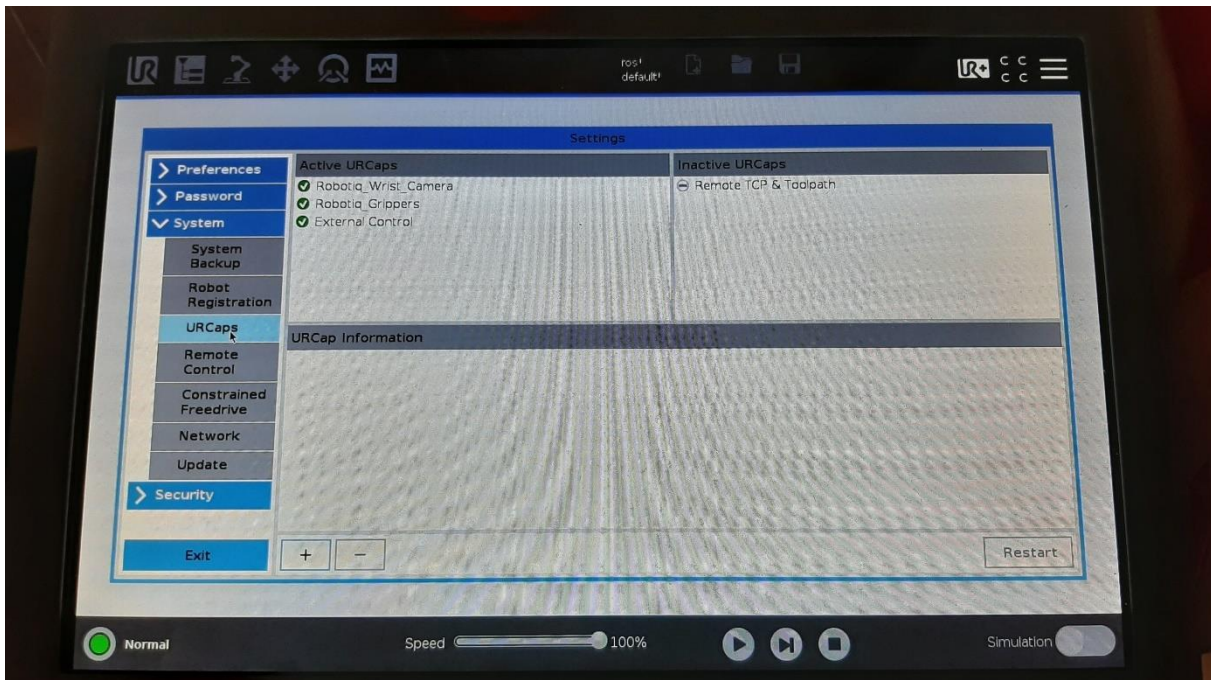
```
catkin_create_pkg robot_calibration \  
-D "Package containing calibrations and launch files for our UR robots."
```

```
mkdir -p ur_calibration/etc
```

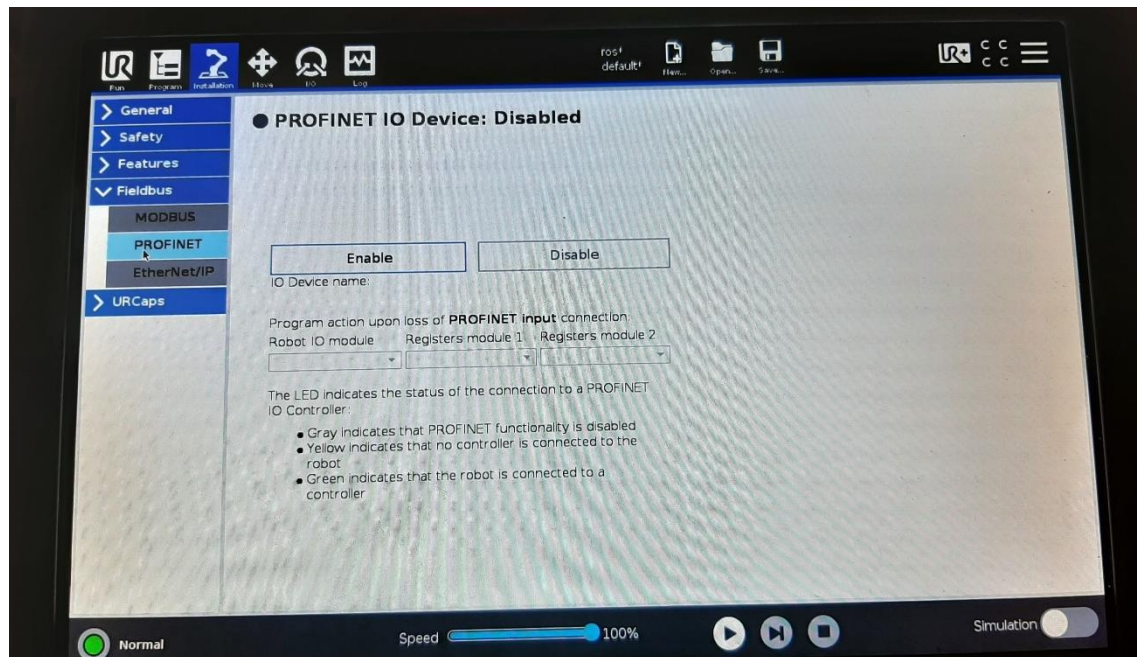
```
mkdir -p ur_calibration/launch
```

Pour continuer, il va falloir installer un "URCap" dans le pupitre du robot. Il s'agit de fonctionnalités que l'on peut ajouter au robot pour augmenter les possibilités de programmation de celui-ci. Dans notre cas, l'extension que nous allons installer va permettre au robot de recevoir un contrôle externe.

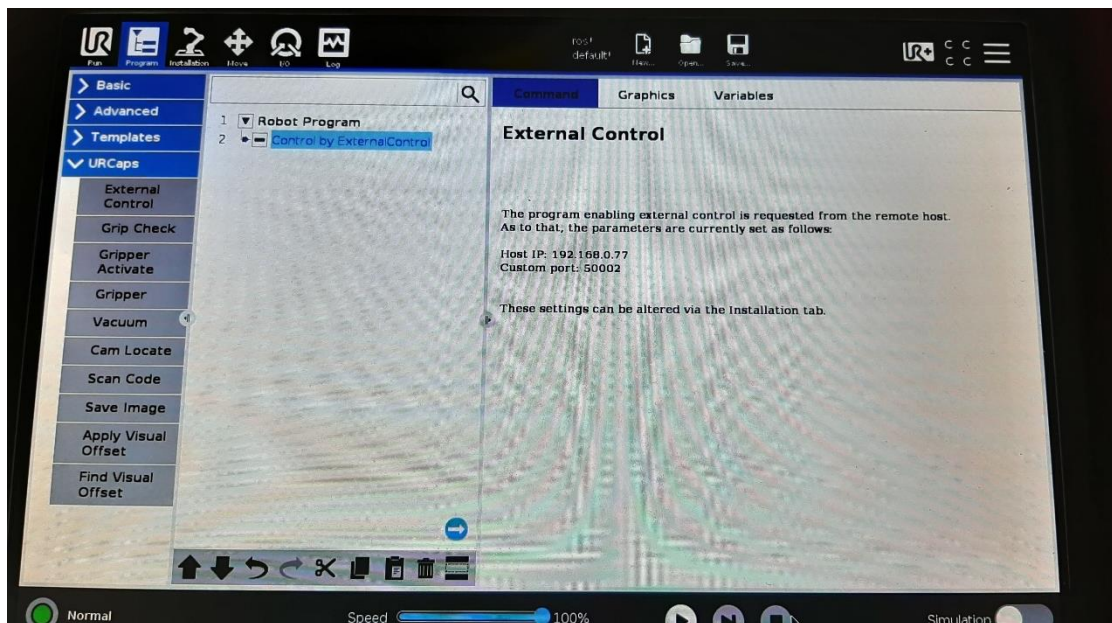
Pour réaliser cela, vous trouverez un fichier *externalcontrol-1.0.5.urcap* dans le dossier *catkin\_ur/ur\_robot\_driver/ressources*. Placez-le sur une clé usb puis la brancher sur le pupitre de commande du robot UR3. Rendez vous ensuite dans les paramètres du robot et activez l'URCap.



Avant de créer notre programme sur le pupitre, vérifiez que le paramètre PROFINET IO Device est bien désactivé sans quoi le contrôle externe ne pourra pas marcher :



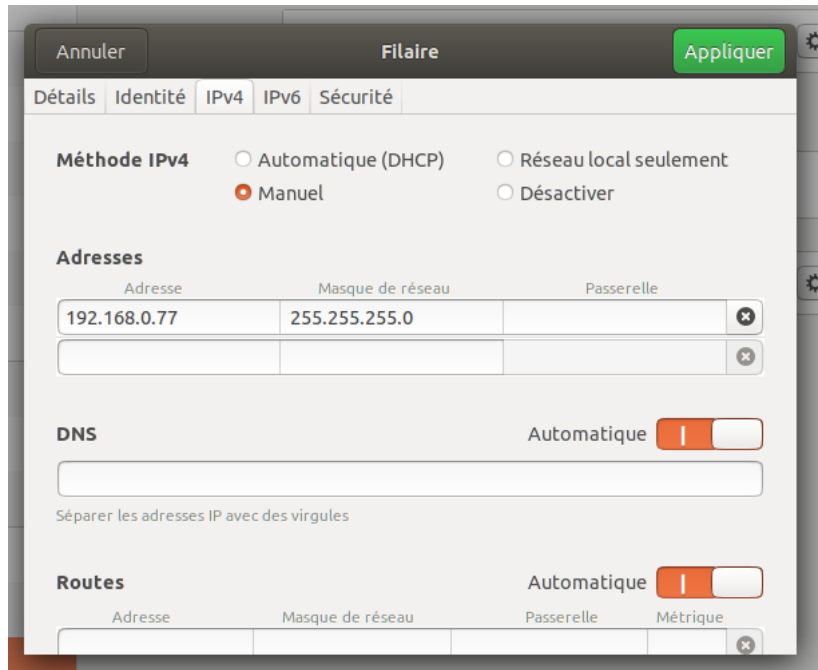
On peut à présent créer notre programme sur le pupitre du robot. Celui-ci est assez simple, il suffit d'une ligne "ExternalControl". Cette fonctionnalité se trouve dans les commandes "URCaps" lorsqu'on crée un programme.



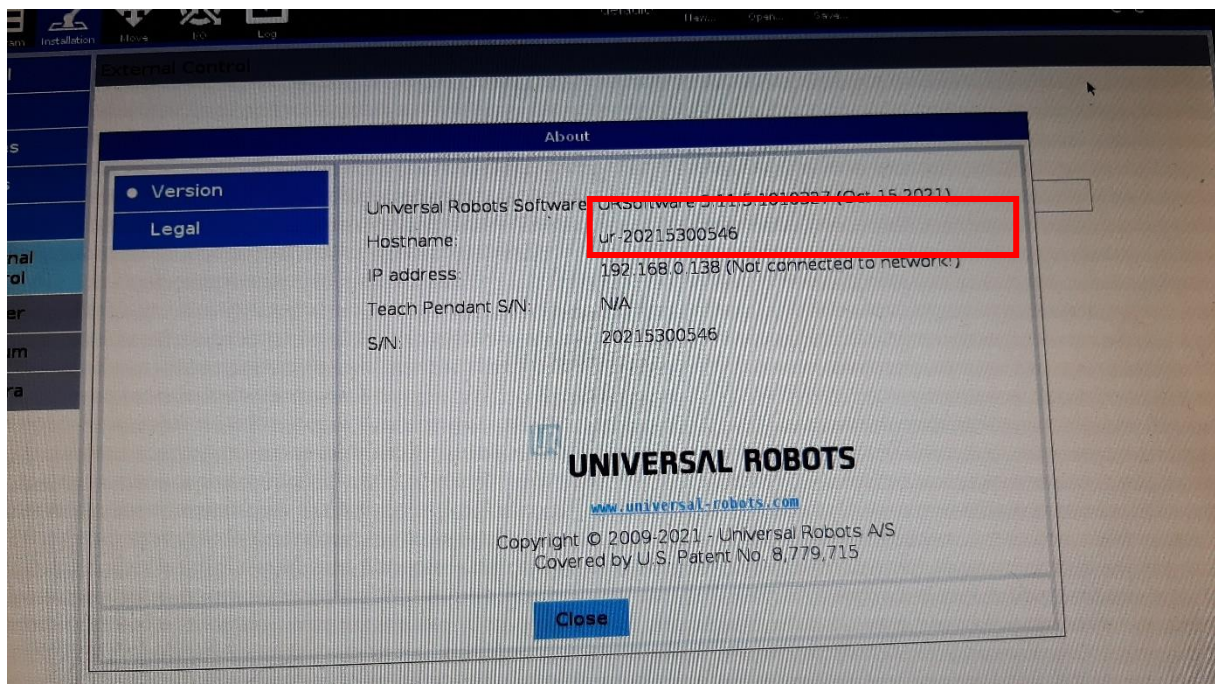
Lorsqu'on met cette ligne de code, une adresse IP est créée, elle sera importante pour la suite.

Lorsqu'on essaye de lancer le programme, celui-ci affiche un message d'erreur, c'est normal, il faut maintenant passer à la partie ordinateur.

Nous pouvons à présent brancher notre ordinateur au robot via un câble Ethernet. Rendez vous ensuite dans les paramètres de connexion filaire de l'ordinateur et entrez les informations relevées précédemment, ici l'adresse IP créée par le programme ExternalControl :



L'étape suivante consiste à trouver l'adresse IP du robot dans ses paramètres. Cette adresse nous servira par la suite.





Tout d'abord, il faut créer un fichier "controllers.yaml" dans le dossier catkin\_ur/src/fmauch\_universal\_robot/ur3\_moveit\_config/config de la manière suivante :



```
controller_list:
- name: ""
  action_ns: /scaled_pos_joint_traj_controller/follow_joint_trajectory
  type: FollowJointTrajectory
  joints:|
    - shoulder_pan_joint
    - shoulder_lift_joint
    - elbow_joint
    - wrist_1_joint
    - wrist_2_joint
    - wrist_3_joint
```

Nous pouvons enfin lancer ROS. La première des choses à faire est de créer le fichier de calibration du robot. Pour cela, on lance la commande suivante en y mettant l'adresse IP du robot vue précédemment (dans mon cas 192.168.0.138) :

```
source catkin_ur/devel/setup.bash

roslaunch
~/catkin_ur/src/Universal_Robot_ROS_Driver/ur_calibration/launch/calibration_correction.launch robot_ip:=<robot_ip> target_filename:="$(rospack find ur_calibration)/etc/ex-ur3_calibration.yaml"
```

Une fois la calibration créée, nous pouvons connecter notre ordinateur au robot. Dans un nouveau terminal :

```
source catkin_ur/devel/setup.bash

roslaunch
~/catkin_ur/src/Universal_Robot_ROS_Driver/ur_robot_driver/launch/ur3_bring_up.launch robot_ip:=<robot_ip> [reverse_port:=REVERSE_PORT] kinematics_config:=$(rospack find ur_calibration)/etc/ex-ur3_calibration.yaml
```

Lorsque cette commande est lancée, elle attend que notre programme ExternalControl soit lancé dans le pupitre du robot. On peut donc lancer ce programme.

Taper ensuite la commande suivante dans un nouveau terminal :

```
source catkin_ur/devel/setup.bash

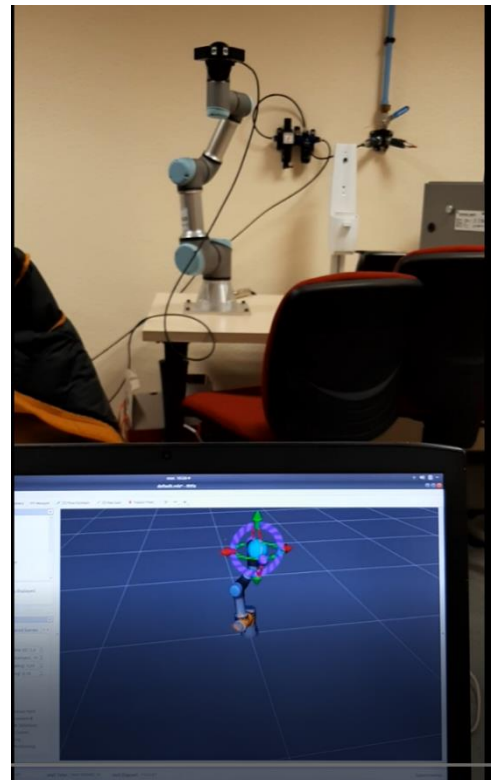
roslaunch
~/catkin_ur/src/fmauch_universal_robot/ur3_moveit_config/launch/ur3_moveit_planning_execution.launch limited:=true
```

Puis dans un dernier terminal :

```
roslaunch
~/catkin_ur/src/fmauch_universal_robot/ur3_moveit_config/launch/moveit_rviz.launch config:=true
```

Il vous suffit ensuite d'insérer le robot dans Moveit, pour cela, référez vous à la partie simulation de ce rapport.

Vous pouvez à présent contrôler le robot depuis votre ordinateur, prenez garde en le manipulant à ne pas cogner quelque chose ou quelqu'un (pour cela, vous pouvez réduire la vitesse du robot), même si le robot dispose d'un système de contrôle de couple anticollision.



## VIII) Conclusion

En conclusion de ce projet, je voudrais souligner tout ce que celui-ci m'a apporté. Tout d'abord, j'ai pu découvrir de nombreux outils informatiques : Linux, Ubuntu, le langage C++, le langage URDF, ROS, ... J'ai également découvert la puissance de ROS-Industrial et j'ai compris les enjeux qui tournent autour de ce logiciel et son rôle qu'il aura dans les années à venir. Je pense qu'une petite expérience de ce logiciel au cours d'un projet tel que celui-ci est un réel atout dans le monde travail et c'est une expérience à souligner dans son CV.

Ce projet m'aura également appris la débrouille et la recherche de solution le tout en autonomie. En effet, je ne connaissais quasiment aucun outil que j'ai utilisé avant de commencer le projet. De plus, ce n'est pas détaillé dans ce rapport mais j'ai rencontré énormément de problèmes. Rare sont les étapes du projet qui se sont déroulées sans soucis, soucis qu'il a fallu régler pour aboutir au résultat final.

C'est dans ce contexte que je veux remercier les professeurs qui ont encadrés ce projet : M. Yan et M. Deng, qui ont été là pour répondre à mes questions et m'aider à l'avancement du projet.

## IX) Bibliographie

- Installation Ubuntu dual boot :

<https://www.itzgeek.com/how-tos/linux/ubuntu-how-tos/how-to-install-ubuntu-18-04-alongside-with-windows-10-or-8-in-dual-boot.html>

- Tutoriel ROS :

<http://wiki.ros.org/ROS/Tutorials>

- Tutoriel ROS-Industrial et Move-it :

<https://industrial-training-master.readthedocs.io/en/melodic/>

- Configuration Simulation ROS :

<https://www.youtube.com/watch?v=ayp87SjrwPc&t=931s>

[https://www.youtube.com/watch?v=j6bBxD\\_bYs&t=226s](https://www.youtube.com/watch?v=j6bBxD_bYs&t=226s)

- Manipuler un vrai robot :

<https://www.youtube.com/watch?v=BS6pFmr7 IA&list=LL&index=3>

[https://github.com/UniversalRobots/Universal\\_Robots\\_ROS\\_Driver#setting-up-a-ur-robot-for-ur-robot-driver](https://github.com/UniversalRobots/Universal_Robots_ROS_Driver#setting-up-a-ur-robot-for-ur-robot-driver)

- Universal robot

[https://github.com/ros-industrial/universal\\_robot](https://github.com/ros-industrial/universal_robot)

- Manipuler un vrai robot pour autres versions :

[http://wiki.ros.org/universal\\_robot/Tutorials/Getting%20Started%20with%20a%20Universal%20Robot%20and%20ROS-Industrial](http://wiki.ros.org/universal_robot/Tutorials/Getting%20Started%20with%20a%20Universal%20Robot%20and%20ROS-Industrial)