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



# Table des matières

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

# 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 :

# III) Présentation de ROS1) 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 dis "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 reçois 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 créer ensuite une connexion entre les deux nodes. •
- Les deux nodes peuvent communiquer entre eux. •

Les messages qui sont communiqués entres 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 :

### 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 détallerai 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.



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-Indutrial 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'environment :

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-rosinstallgenerator 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 <u>https://github.com/ros-</u>
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.

	Movelti Setup Assistant 🖉 🖨 🎯
Start	Movelt! Setup Assistant
Self-Collisions	These tools will assist you in creating a Semantic Robot Description Format (SRDF) file, various yaml configuration and many roslaunch files for utilizing all aspects of MoveltI functionality.
Virtual Joints	Create new or edit existing? All settings for Movelt are stored in the Movelt configuration package.
Planning Groups	Here you have the option to create a new configuration package or load an existing one. Note: changes to a Movelti configuration package outside this Setup Assistant are likely to be overwritten by this tool.
Robot Poses	Create New Movelt Edit Existing Movelt Configuration Package
End Effectors	
Passive Joints	
ROS Control	
Simulation	Movelt!
3D Perception	Setup Assistant 2.0
Author Information	
Configuration Files	



Entrer la localisation du fichier de description du robot.

				Move	It! Setup Assistar	nt		
Start	ο	ptimize						
Self-Collisions	Thi: dec	s searches for p creasing motion ver in collision	pairs of robot li n planning time in collision in th	inks that can s a. These pairs a he robot's def	afely be disabled I are disabled when ault position, or w	from collision checking, they are always in collis	sion,	
Virtual Joints	eac to c	check for self c	kinematic chai collision.	n. Sampling d	ensity specifies ho	w many random robot (	positions	
Planning Groups	Si	ampling Densit	ty: Low 💳 🖸			High	10000	
Robot Poses		Mi	n. collisions for	"always"-coll	iding pairs: 95%	Cenerate Collision	Matrix	
End Effectors		Link A 🔻	Link B	Disabled	eason to Disab			
	1	base_link	shoulder_link	✓	Adjacent Li			
			writt 1 link	7	Adjacopt Li			Carl Contract
Passive Joints	2	forearm_link	WIISC_I_UIIK	•	Aujacent Li			
ROS Control	2	forearm_link	wrist_2_link	V	Never in Co			X
ROS Control	2 3 4	forearm_link forearm_link shoulder_link	wrist_2_link upper_arm	✓ ✓	Adjacent Li Adjacent Li			<b>X</b>
Passive Joints ROS Control Simulation	2 3 4 5	forearm_link forearm_link shoulder_link shoulder_link	wrist_2_link upper_arm wrist_1_link	V V V	Adjacent Li         Never in Co         Adjacent Li         Never in Co			×.
ROS Control Simulation 3D Perception	2 3 4 5 6	forearm_link forearm_link shoulder_link shoulder_link shoulder_link	wrist_2_link upper_arm wrist_1_link wrist_2_link	V V V V	Adjacent Li Adjacent Li Never in Co Never in Co			*
Passive Joints ROS Control Simulation 3D Perception	2 3 4 5 6 7	forearm_link forearm_link shoulder_link shoulder_link upper_arm	wrist_2_link upper_arm wrist_1_link wrist_2_link forearm_link	× × × × ×	Adjacent Li Adjacent Li Never in Co Never in Co Adjacent Li			X
Passive Joints ROS Control Simulation 3D Perception Author Information	2 3 4 5 6 7 8	forearm_link forearm_link shoulder_link shoulder_link shoulder_link upper_arm wrist_1_link	wrist_2_link upper_arm wrist_1_link wrist_2_link forearm_link ee_link	× × × × × × × ×	Adjacent Li Never in Co Never in Co Adjacent Li Never in Co			X

Générer matrice de collision.



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

	Movelt! Setup Assistant	θ
Start	Define Virtual Joints	
Self-Collisions	Create a virtual joint between a robot link and an external frame of reference (considered fixed with respect to the robot).	
Virtual Joints	Virtual Joint Name:	
	FixedBase	
Planning Groups	Child Link:	
	world	
Robot Poses	Parent Frame Name:	
End Effectors	world	
LING LITECOUS	Joint Type:	-50
Passive Joints	fixed 👻	18 N
ROS Control		<b>\$</b>
Simulation		
3D Perception		
Author Information		
Configuration Files	Save Cancel	
		visual collision

	Moveltl Setup Assistant	
Start	Define Planning Groups	
Self-Collisions	Create and edit 'joint model' groups for your robot based on joint collections, link collections, kinematic chains or subgroups. A planning group defines the set of (joint, link) pairs considered for planning and collision checking. Define individual groups for each subset of the robot you want to plan for Note: when adding a link to the group, its parent joint is added too and vice versa.	
Virtual Joints	Current Groups	
Planning Groups		
Robot Poses		
End Effectors		<u>e.</u>
Passive Joints		
ROS Control		~
Simulation		
3D Perception		
Author Information		
Configuration Files		
	Expand All Collapse All	✓ visual 🗌 collision

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

	Movelti Setup Assistant	• • •
Start	Define Planning Groups	
Self-Collisions	Create and edit "joint model' groups for your robot based on joint collections, link collections, kinematic chains or subgroups. A planning group defines the set of (joint, link) pairs considered for planning and collision checking. Define individual groups for each subset of the robot you want to plan for Note: when adding a link to the group, its parent joint is added to on and vice versa.	
Virtual Joints	Create New Planning Group	
Planning Groups	Kinematics	
Robot Poses	Group Name: manipulator Kinematic Solver: kdL_kinematics_plugin/KDLKinematicsPlugin ~	
End Effectors	Kin. Search Resolution: 0.005	
Passive Joints	Kin. Search Timeout (sec): 0.005 Kin. parameters file:	
ROS Control	OMPL Planning	
Simulation	Group Default Planner: None	
3D Perception	Next, Add Components To Group:	
Author Information	Recommended: Add Kin. Chain Add Joints	
and investigate all as	Advanced Options: Add Subgroups Add Links	
Configuration Files	<u>C</u> ancel √ visual □ collis	sion

_	Movelti Setup Assistant	• • •
Start	Define Planning Groups	
Self-Collisions	Create and edit 'joint model' groups for your robot based on joint collections, link collections, kinematic chains or subgroups. A planning group defines the set of 'joint, link' pairs considered for planning and collision checking. Define individual groups for each subset of the robot you want to plan for.Note: when adding a link to the group, its parent joint is added to on and vice versa.	
Virtual Joints	Edit 'manipulator' Kinematic Chain	
Planning Groups	Robot Links	
Robot Poses	base	
End Effectors	v upper_arm_link v forearm_link v wrist 1 link	8.
Passive Joints	▼ wrist_2 link ▼ wrist_3 link	
ROS Control	ee_link tool0	
Simulation		
3D Perception	· · · · · · · · · · · · · · · · · · ·	
Author Information	Base Link base_link Choose Selected	
Configuration Files	Tip Link tool0 Choose Selected	
configuration Files	Expand All Collapse All	collision

	Movelt! Setup Assistant	• •
Start	Define Robot Poses	
Self-Collisions	Create poses for the robot. Poses are defined as sets of joint values for particular planning groups. This is useful for things like home position. The first pose for each robot will be its initial pose in simulation.	
Virtual Joints	Pose Name Group Name	
Planning Groups		
End Effectors		Sec.
Passive Joints		
ROS Control		6
Simulation		
3D Perception		
Author Information		
Configuration Files		
	Show Default Pose Movelt! Delete Selected Add Pose	✓ visual 🗌 collision

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.



	Movelti Setup Assistant	5	99
Start	Generate Configuration Files		
Self-Collisions	Create or update the configuration files package needed to run y files to disable them from being generated - this is useful if you h them. Files in orange have been automatically detected as change	our robot with Movelt. Uncheck ave made custom changes to ed.	
Virtual Joints	Configuration Package Save Path		
Planning Groups	Specify the desired directory for the MoveltI configuration pack Overwriting an existing configuration package directory is accep panda_moveit_config	age to be generated. stable. Example: /u/robot/ros/	
Robot Poses	/home/emeric/catkin_ws/src/myur3_moveit_config2	Browse	
End Effectors	Files to be generated: (checked)		
Passive Joints	✓ package.xml     ✓ CMakeLists.txt     ✓ config/     ✓ config/     ✓ config/	Defines a ROS package	
ROS Control	✓ config/ompl_planning.yaml ✓ config/chomp planning.yaml	-	
Simulation	<ul> <li>✓ config/kinematics.yaml</li> <li>✓ config/joint_limits.yaml</li> <li>✓ config/cartesian_limits.yaml</li> </ul>		
3D Perception	<ul> <li>✓ config/fake_controllers.yaml</li> <li>✓ config/ros_controllers.yaml</li> </ul>		
Author Information	✓ config/sensors_3d.yaml ✓ launch/	v )	
Configuration Files		<u>G</u> enerate Package	
		Exit Setup Assistant	sion

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 :

•	🏠 Dossier person	nel catkin_ws	src	myur3_moveit_config2	(	ર	≡	●
ents		Nom			Taille	Dernière	e modif	fication
sier	personnel	in config			11 éléments		26	déc. 2021
au ume	nts	aunch			24 éléments		26	déc. 2021
jes		CMakeLis	ts.txt		309 octets		26	déc. 2021
ique char	e gements	v> package.	kml		1,5 ko		26	déc. 2021
os								

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	
<pre>//arm_controller/follow_joint_trajectory/:ancel /arm_controller/follow_joint_trajectory/feedback /arm_controller/follow_joint_trajectory/feedback /arm_controller/follow_joint_trajectory/result /arm_controller/follow_joint_trajectory/status /arm_controller/state /attached_collision_object /calibested</pre>	
/clicked_point /clicked /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_com_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 ▼ 🖪		
  < <b>xacro:</b> cylinder_inertial radius="0.0 <origin length="0.038" mass="\${base_mass}" rpy="0 0&lt;br&gt;&lt;/&lt;b&gt;xacro:&lt;/b&gt;cylinder inertial&gt;&lt;/td&gt;&lt;td&gt;75" xyz="0.0 0.0 0.0"> 0" /&gt;</origin>		
		Nom de la
<pre><joint name="\${prefix}shoulder pan joi&lt;/td&gt;&lt;td&gt;nt' type=" revolute"=""></joint></pre>	première	
<pre><child \${shoulder_heig<="" 0.0="" link="\${prefix}shoulder_lin &lt;origin xyz=" pre=""></child></pre>	k" /> ht}" гру="0.0 0.0 0.0" />	articulation.
<axis xyz="0 0 1"></axis> <xacro:unless value="\${joint limited&lt;/td&gt;&lt;td&gt;}"></xacro:unless>		
<limit lower="\${-2.0 * pi}" upper="&lt;br"><xacro:if <="" td="" value="\${safety_limits}"><td>"\${2.0 * pi}" effort="330.0" velocity="2.16"/&gt; &gt;</td><td></td></xacro:if></limit>	"\${2.0 * pi}" effort="330.0" velocity="2.16"/> >	
<safety_controller soft_lower_li<br=""><!--<b-->xacro:if&gt;</safety_controller>	mit="\${-2.0 * pi + safety_pos_margin}" soft_up	

Voici la forme du fichier "controllers.yaml" :



• "Joint\_names.yaml" :

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

Ouvrir 🔻 🖪	<b>joint_names.yaml</b> ~/catkin_ws/src/myur3_moveit_config2/config	Enregistrer	
<pre>controller_joint_name: wrist_2_joint, wrist_3_</pre>	[shoulder_pan_joint, shoulder_lift_joint, elbow_joint, wrist _joint]	t_1_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"

Ouvrir 🕶 🖪	ur3_moveit_controller_manager.launch.xml ~/catkin_ws/src/myur3_moveit_config2/launch					
ur3_moveit_controller_manager.launch.xn	nl ×	gazebo.launch	×	myur3_planning		
<launch> <arg default="&lt;br&gt;&lt;arg name=" moveit_controller_manager<br="" name="execution_type">MoveItSimpleControllerManager" /&gt; <param <="" name="moveit_controller_manager" td=""/><th>'interpo " defau ger" val</th><th>blate"/&gt; Jlt="moveit_simple_c Lue="\$(arg moveit_cc</th><th>ontro</th><td>oller_manager/ ller_manager)"/&gt;</td></arg></launch>	'interpo " defau ger" val	blate"/> Jlt="moveit_simple_c Lue="\$(arg moveit_cc	ontro	oller_manager/ ller_manager)"/>		
<param <br="" name="use_controller_manager"/> <param \$(find="" myur3_moveit_<br="" name="trajectory_execution/ex&lt;br&gt;&lt;! loads ros_controllers to the pa&lt;br&gt;&lt;rosparam file="/>	' value= (ecutior aram ser _config2	="false"/> n_duration_monitorir - <b>ver</b> > 2)/config/controller	ng" va s.yar	alue="false"/> nl"/>		

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" :

Ouvrir ▼ 🖪 ~/catki	<b>gazebo.launch</b> n_ws/src/myur3_moveit_config2/launch	Enregistrer 🗧 🗨 📾 😣
ur3_moveit_controller_manager.launch.xml ×	gazebo.launch ×	myur3_planning_execution.launch ×
<pre><?xml version="1.0"?> <launch></launch></pre>	f true, limits joint rango tarts gazebo in paused moo s gazebo gui" /> ault="hardware_interface/"	e [-PI, PI] on all joints." /> de" /> PositionJointInterface" />
<pre><!-- startup simulated world--> <include file="\$(find gazebo_ros)/launch/&lt;/td&gt;&lt;th&gt;empty_world.launch"> empty.world"/&gt; "/&gt;<td></td></include></pre>		
<pre></pre>	\$(find xacro)/xacro '\$(fin face:=\$(arg transmission_	nd ur_description)/urdf/ hw_interface)"/>
<pre><!-- push robot_description to factory an<br--><node name="spawn_gazebo_model" pkg="gaze&lt;br&gt;robot_description -model robot -z 0.0" pre="" resp<=""></node></pre>	d spawn robot in gazebo - bo_ros" type="spawn_model" awn="false" output="scree	-> " args="-urdf -param n" />
<pre><include file="\$(find ur_gazebo)/launch/c&lt;/pre&gt;&lt;/td&gt;&lt;th&gt;ontroller_utils.launch"></include><td></td></pre>		
<pre><!-- start this controller--> <rosparam file="\$(find ur_gazebo)/control&lt;/td&gt;&lt;th&gt;ler/arm_controller_ur3.ya&lt;br&gt;controller_manager" type="&lt;br">een"/&gt;<td>ml" command="load"/&gt; "controller_manager" args="spawn</td></rosparam></pre>	ml" command="load"/> "controller_manager" args="spawn	

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"

Ouvrir - 🖪	<b>myur3</b> ~/catkin_v	_planning_execution.lau ws/src/myur3_moveit_config2/	I <b>nch</b> ′launch	Enregistrer 🗧 🖨 🔿 🖉
ur3_moveit_controller_manager.launch.xml	×	gazebo.launch	×	myur3_planning_execution.launch ×
<launch></launch>				
<rosparam command="load" file="\$(find&lt;/td&gt;&lt;th&gt;l myur3&lt;/th&gt;&lt;th&gt;3_moveit_config2)/co&lt;/th&gt;&lt;td&gt;onfig&lt;/td&gt;&lt;td&gt;/joint_names.yaml"></rosparam>				
<include file="\$(find myur3_moveit_co&lt;br&gt;&lt;arg name=" load_robot_description"<br=""></include>	onfig2) value=	)/launch/planning_co ="true"/>	ontex	t.launch">
<node name="joint_state_publisher" pk<br=""><param name="/use_gui" source_list"="" value="false&lt;br&gt;&lt;rosparam param="/>[/jo </node>	g="joi e"/> oint_st	int_state_publisher' tates]	'typ	e="joint_state_publisher">
<include file="\$(find myur3_moveit_co&lt;br&gt;&lt;arg name=" publish_monitored_planni<br=""></include>	onfig2) .ng_sce	)/launch/move_group ene" value="true"/>	laun	ch">
<include file="\$(find myur3_moveit_co&lt;br&gt; &lt;br&gt;&lt;/include&gt;&lt;/td&gt;&lt;th&gt;onfig2)&lt;/th&gt;&lt;th&gt;)/launch/moveit_rvi&lt;/th&gt;&lt;td&gt;z.lau&lt;/td&gt;&lt;td&gt;nch"></include>				

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 <code>rostopic</code> <code>list</code>, le nom du paramètre.

emeric@Emeric-Humbert: ~	
Fichier Édition Affichage Rechercher Terminal Aide	
<pre>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</pre>	
/calibrated /clicked_point /clock /collision_object /execute_trajectory/cancel /execute_trajectory/feedback /execute_trajectory/goal /execute_trajectory/result /execute_trajectory/result /execute_trajectory/status /gazebo/link_states /gazebo/link_states	
/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 même 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 :

> General	BROEINET IO Davisa	Disabled			
> Safety	• PROFINET TO Device.	Disabica			
> Features	A had a hard a star hard a				
V Fieldbus	The second conduction of the			MARIE	
MODBUS				MANES	
PROFINET		Disable		191122	
EtherNet/IP	Enable	DISADIC		anne -	
	IO Device ridirle.		11111	111111	
> URCaps	Program action upon loss of PROFI Robot IO module Registers mod	INET input connection: dule 1 Registers module	2		
) URCaps	Program action upon loss of <b>PROFI</b> Robot 10 module Registers more than the LED indicates the status of the 10 Controller: • Gray indicates that PROFINET • Yellow indicates that no contr robot • Green indicates that the robo controller	INET input connection due 1 Registers module connection to a PROFINET Functionality is disabled foller is connected to the it is connected to a	2		

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 créer un programme.

> Basic	0	Command	Graphics	Variables	
> Advanced	1 W Rebet Britani	Wagenered and	Graphics	Variables	
> Templates	2 Control by ExternalControl	External C	Control		
V URCaps	and the second provident	Market States			
External Control	a second second	The program of			
Grip Check		As to that, the p	arameters are cu	arrently set as follows:	the remote host.
Gripper Activate	111 A.	Host IP: 192.168 Custom port: 50	1.0.77 002	Had Street	
Gripper		DE CART		ALL STREET	
Vacuum 🕐		P P P	an be altered via	the Installation tab.	
Cam Locate		142214		S. C. C. S.	
Scan Code			31113	A CARLES	
Save Image		CALL AND		ALC: NO	
Apply Visual Offset			<b>ELLA</b>		
Find Visual Offset	and the second s	26.21			
	0	all all		themes and the second	

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 :

Annuler	Filaire	Appliqu	er
étails Identité IP	v4 IPv6 Sécurité		
Méthode IPv4	O Automatique (DHCP)	🔿 Réseau local seulement	
	O Manuel	○ Désactiver	
Adresses			
Adresse	Masque de réseau	Passerelle	
192.168.0.77	255.255.255.0	8	
		8	
DNS		Automatique	)
Séparer les adresses IP	avec des virgules		J
Routes		Automatique	)
Adresse	Masque de réseau	Passerelle Métrique	
		0	

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

			*
Version		About	7
Legal	Hostname:	ur-20215300546	
	IP address:	192/168(0.138 (Not connected to network)	
	Teach Pendant S/N:	20215300546	
		• · · · · · · · · · · · · · · · · · · ·	
		UNIVERSAL ROBOTS	
		www.univeltsal.com	
	Copyr	overed by U.S. Patent No. 8/779/715	

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 :



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/calibratio
n_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 vélocité 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-18-04alongside-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=j6bBxfD\_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-robotfor-ur robot driver

• 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