

Annexe (ROS)

Introduction

L'évolution de la robotique a permis la réalisation de systèmes robotiques sophistiqués capables d'effectuer des tâches complexes avec une grande précision. Dans ce contexte, la simulation joue un rôle clé, offrant un environnement sûr et économique pour tester et valider des conceptions avant leur fabrication.

Ce projet porte sur la conception et la simulation d'un bras robotique à l'aide de **SolidWorks** pour la modélisation 3D et de **ROS (Robot Operating System)** pour la simulation et le contrôle.

La conception mécanique du bras robotique est réalisée dans SolidWorks, un logiciel de CAO (Conception Assistée par Ordinateur) offrant des outils avancés pour modéliser des systèmes mécaniques complexes. Une fois la conception finalisée, le modèle 3D est exporté vers ROS, une plateforme de développement open source, qui fournit une infrastructure robuste pour le contrôle, la planification, la navigation et la simulation des robots.

L'objectif principal de ce projet est de :

- 1. Concevoir un bras robotique en 3D** avec SolidWorks, en respectant les spécifications mécaniques.
- 2. Importer le modèle dans ROS**, en utilisant des outils tels que URDF (Unified Robot Description Format) pour décrire les propriétés cinématiques et dynamiques du robot.
- 3. Simuler le bras robotique dans Gazebo**, un simulateur compatible avec ROS, afin de valider ses mouvements et d'optimiser ses performances.
- 4. Tester et contrôler** les mouvements du bras en utilisant des algorithmes intégrés à ROS.

I. Installation de ROS Noetic et création d'un espace de travail Catkin

1. Prérequis système

- **Système d'exploitation** : ROS Noetic est compatible uniquement avec **Ubuntu 20.04**. Assurez-vous que votre système d'exploitation est correct.
- **Connexion Internet** : Nécessaire pour télécharger les packages ROS.

2. Installation de ROS Noetic

Configuration des sources de ROS : Ouvrez un terminal et ajoutez les dépôts ROS à votre gestionnaire de paquets.

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

Ajout de la clé GPG :

```
sudo apt update
sudo apt install curl
curl -s https://raw.githubusercontent.com/ros/rosdistro/master/ros.asc | sudo apt-key add -
```

Mise à jour et installation de ROS Noetic :

```
sudo apt update
sudo apt install ros-noetic-desktop-full
```

Initialisation de rosdep : rosdep permet d'installer les dépendances des packages ROS.

```
sudo rosdep init
rosdep update
```

Ajout des variables d'environnement : Ajoutez la configuration ROS à votre fichier `.bashrc`.

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

Installation de catkin et autres outils utiles :

```
sudo apt install python3-rosinstall python3-rosinstall-generator python3-wstool build-essential
```

3. Création de l'espace de travail Catkin

Créer un répertoire pour l'espace de travail :

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

Initialiser l'espace de travail : Accédez au répertoire `catkin_ws` et initialisez-le.

```
cd ~/catkin_ws/
catkin_make
```

Configurer les variables d'environnement : Ajoutez la configuration de l'espace de travail Catkin dans le fichier `.bashrc`.

```
echo "source ~/catkin_ws/devel/setup.bash" >> ~/.bashrc  
source ~/.bashrc
```

Vérifiez que l'espace de travail est correctement configuré :

```
echo $ROS_PACKAGE_PATH
```

4. Test de ROS Noetic

Lancer le Master ROS (roscore) :

```
roscore
```

Si le processus démarre correctement, ROS est installé avec succès.

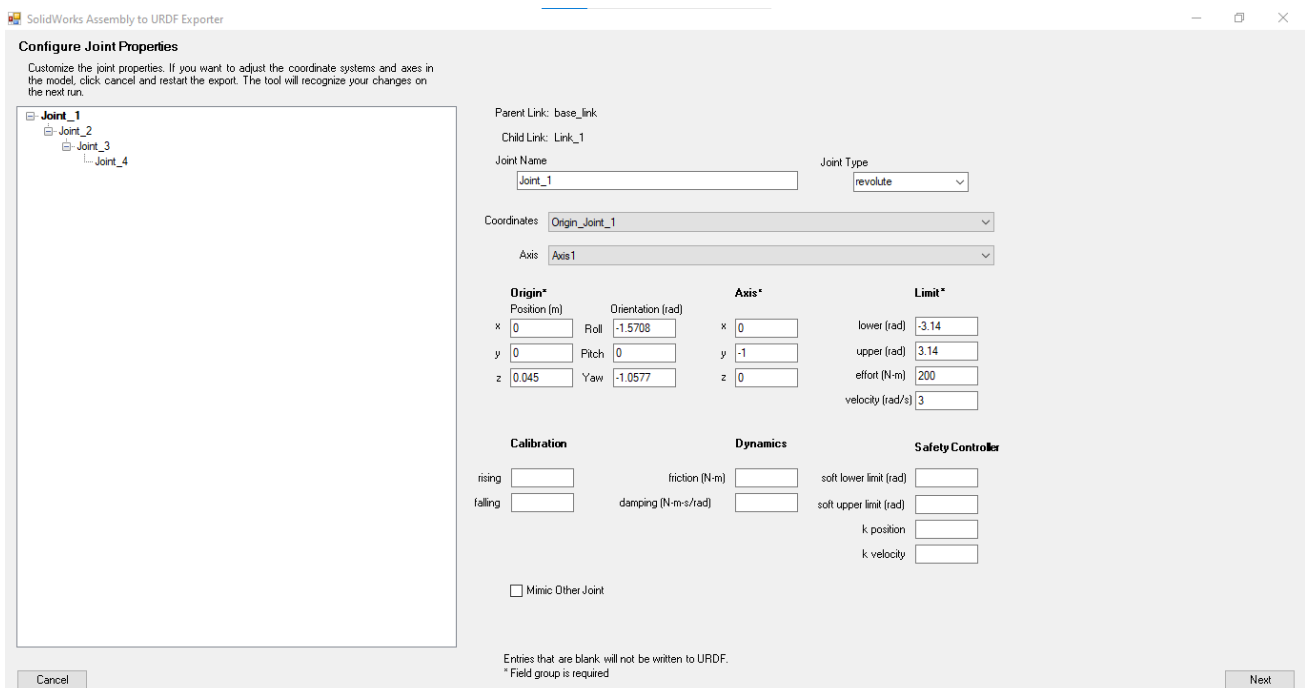
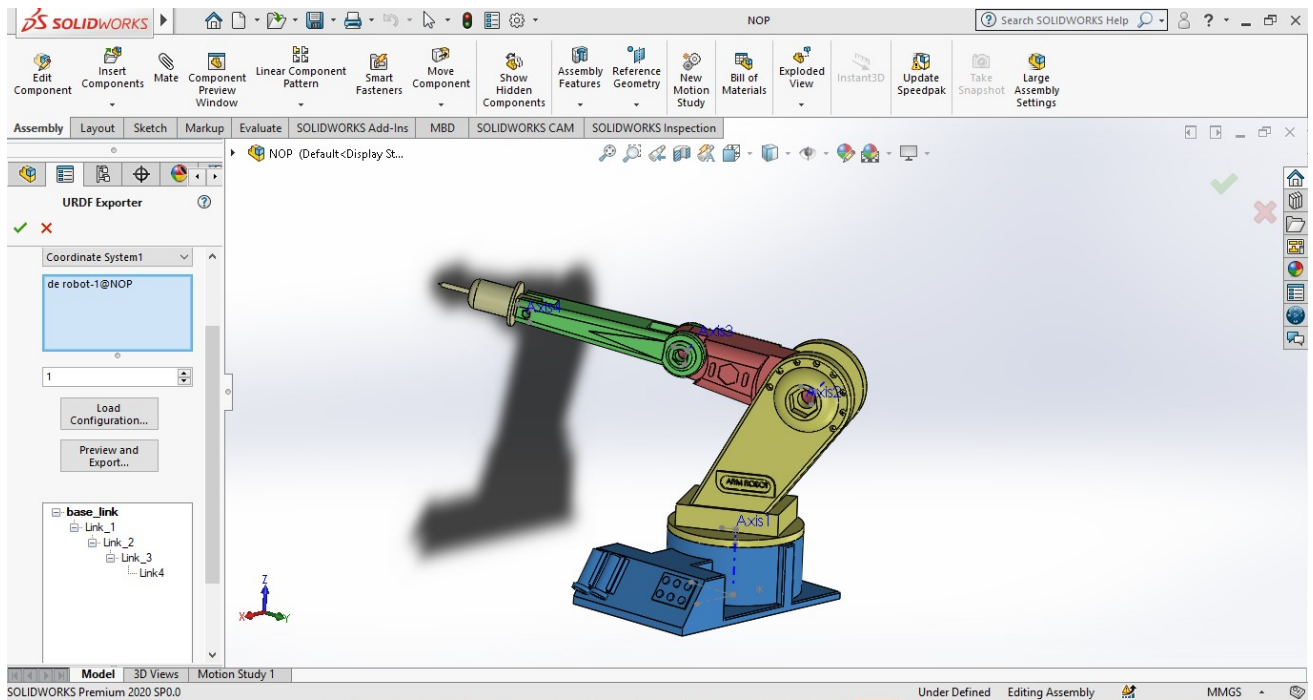
Vérifier la structure de l'espace Catkin : Vous devriez voir les dossiers suivants dans catkin_ws :

```
ls ~/catkin_ws  
build devel src
```

II. Exportation du package URDF depuis SOLIDWORKS

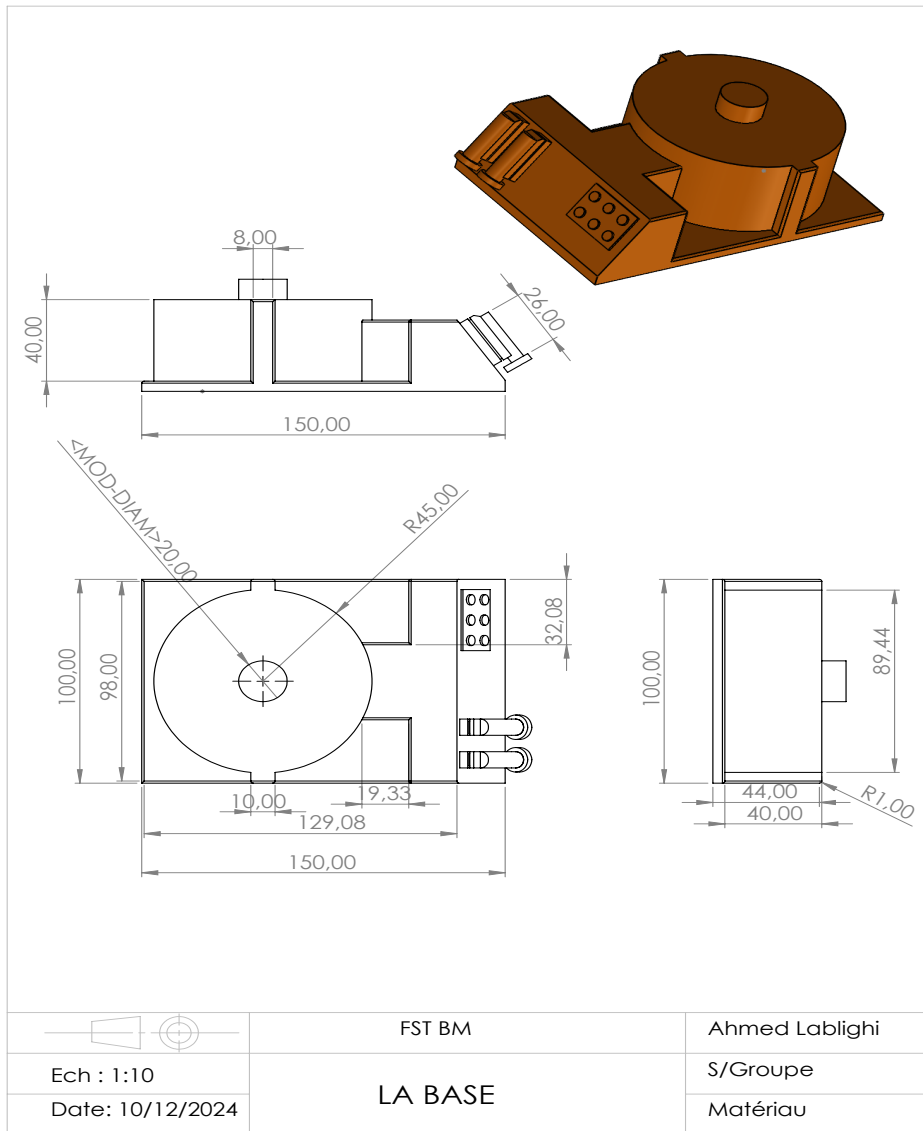
L'exportation d'un modèle URDF (Unified Robot Description Format) à partir de SolidWorks nécessite l'utilisation d'un plugin spécialement conçu pour générer des fichiers compatibles ROS. Voici les étapes détaillées :

1. Préparer le modèle dans SolidWorks :



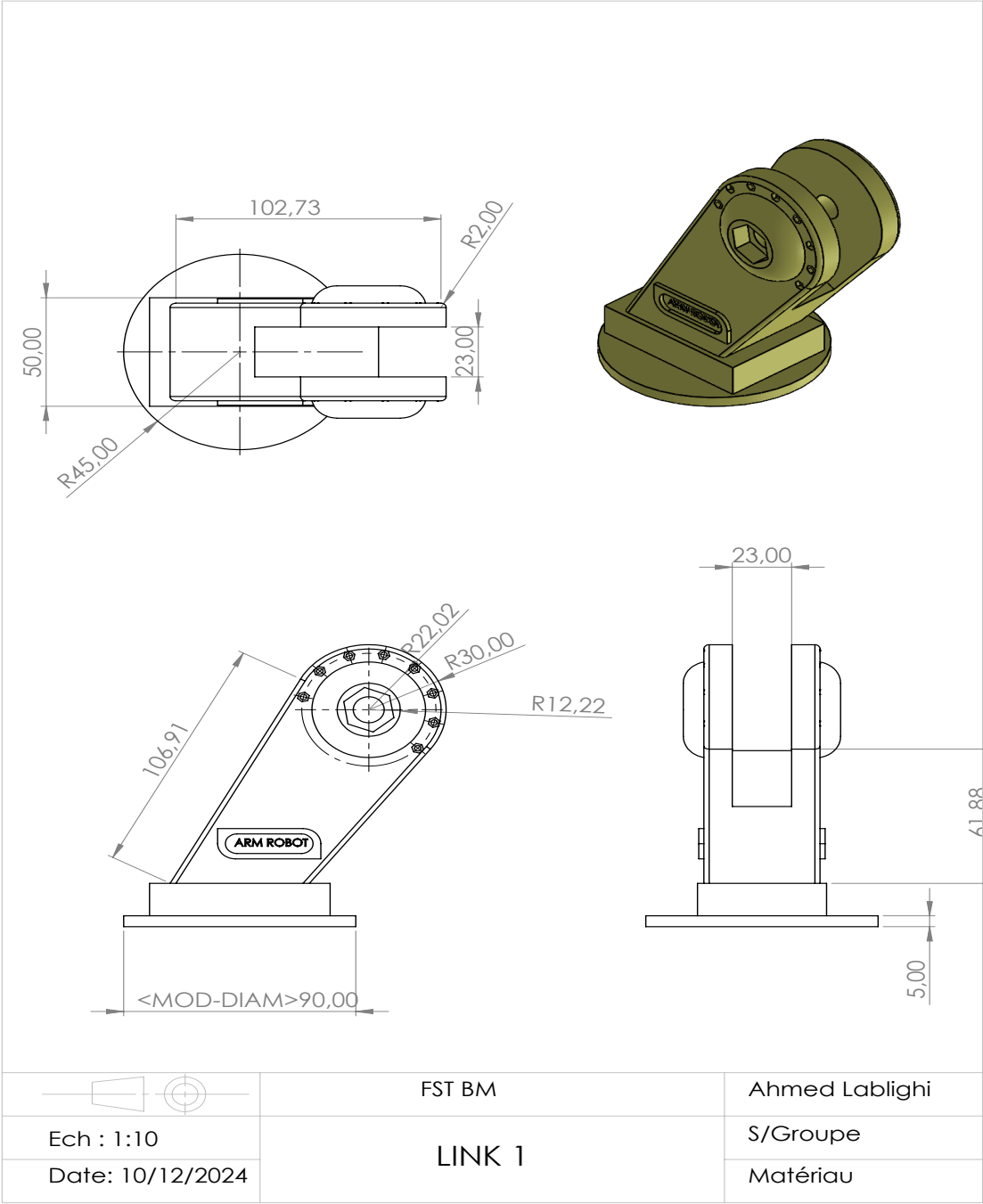
Conception des Composants Individuels :

Base du robot :

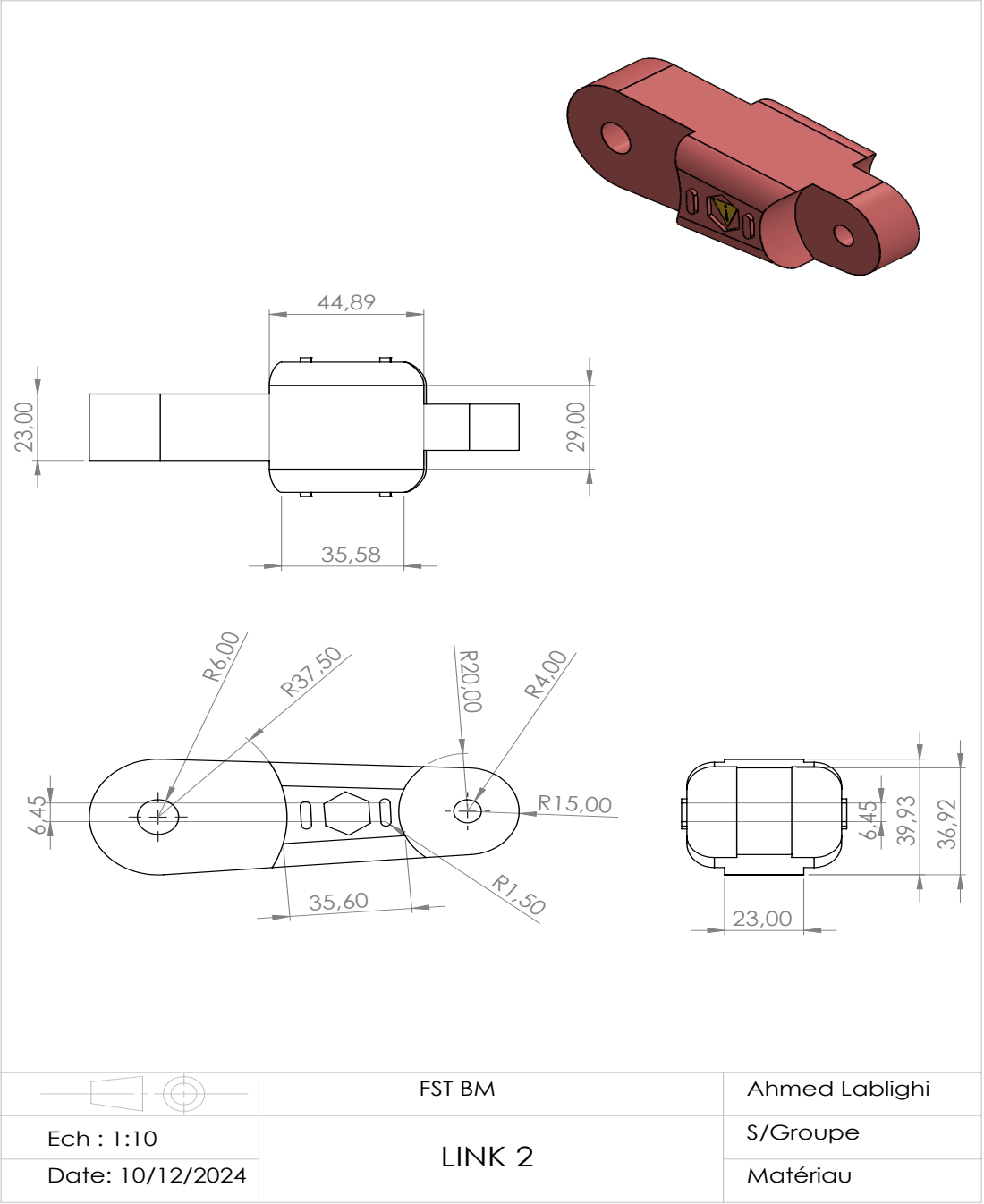


Segments du bras :

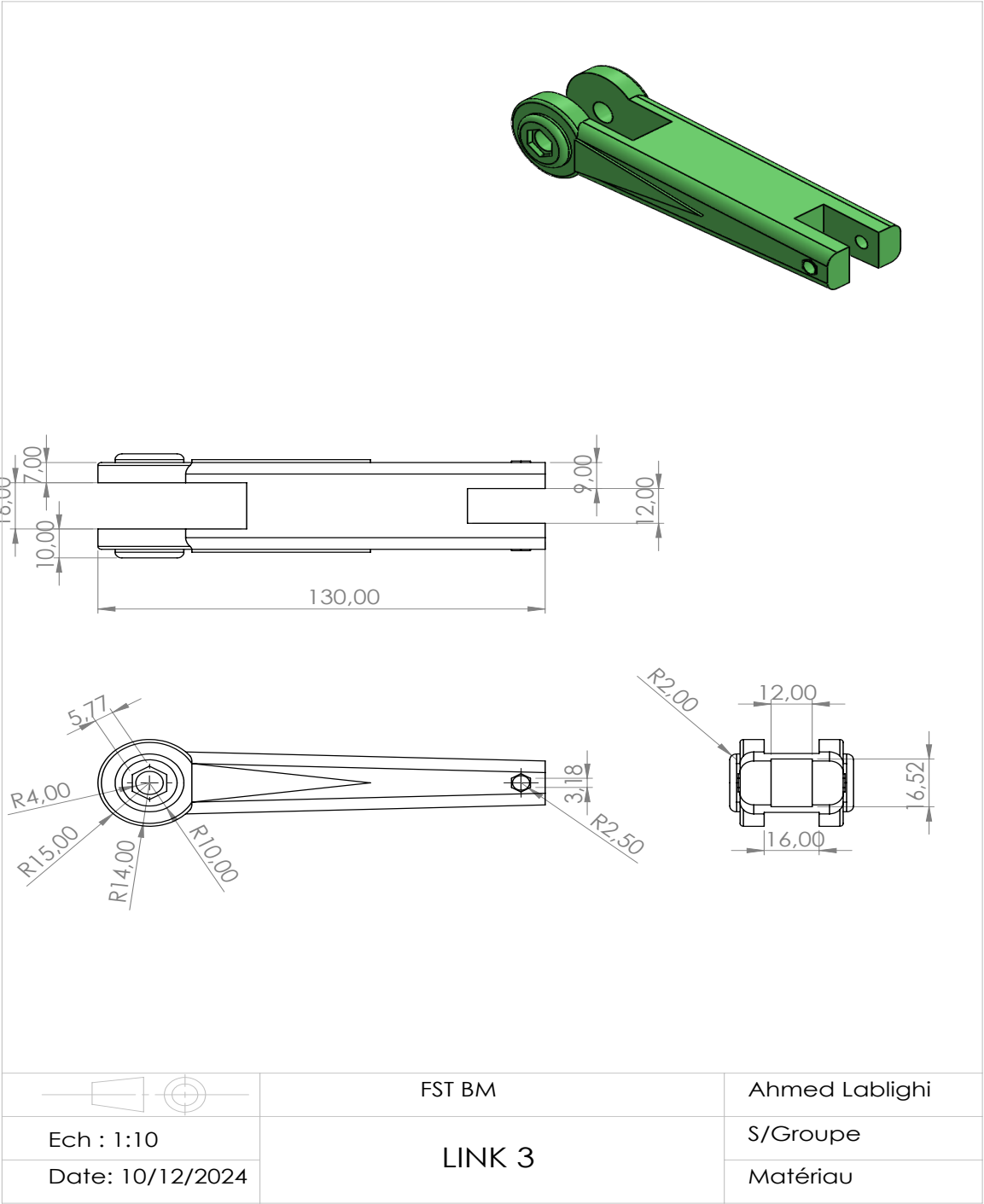
Link 1 :



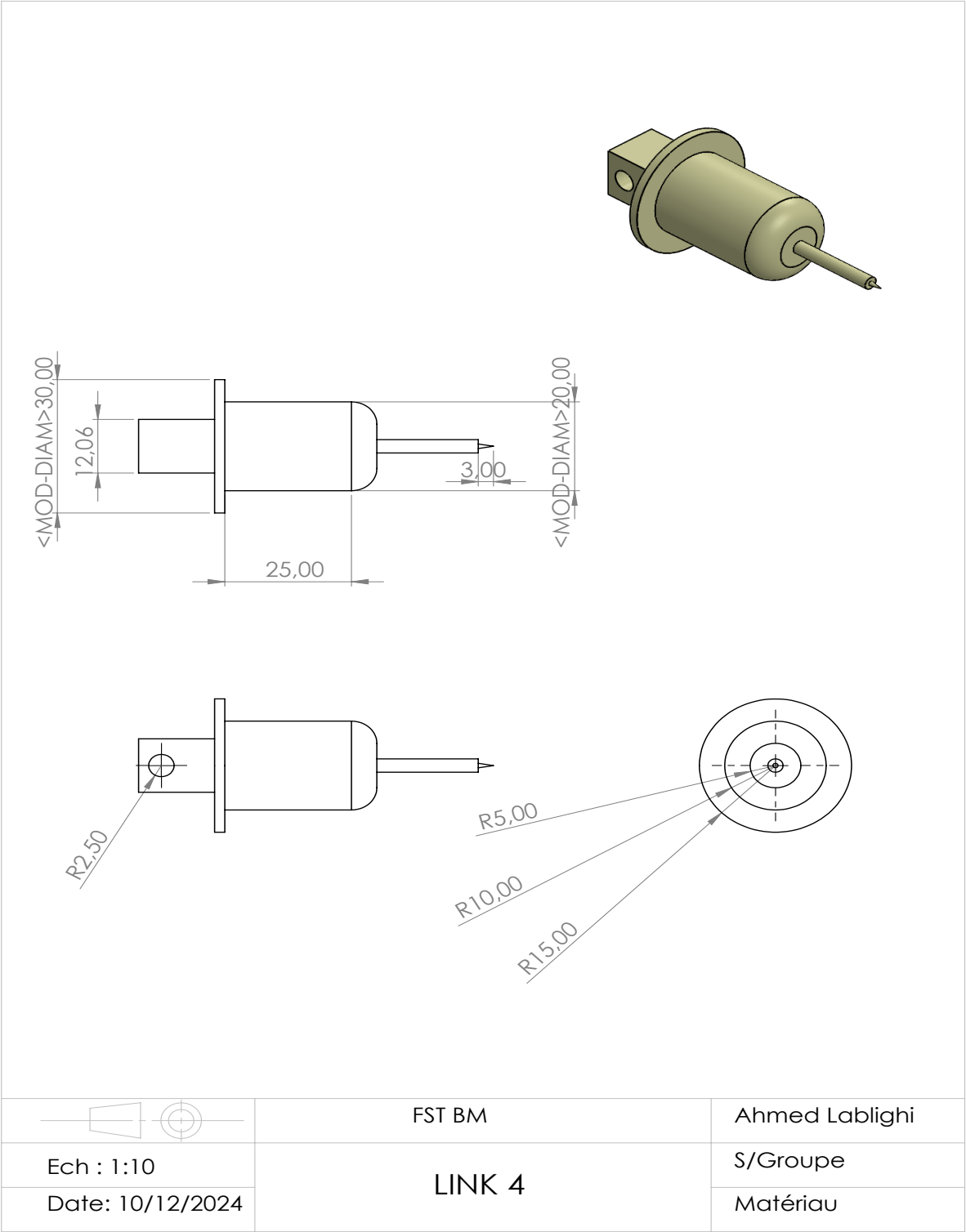
Link 2 :



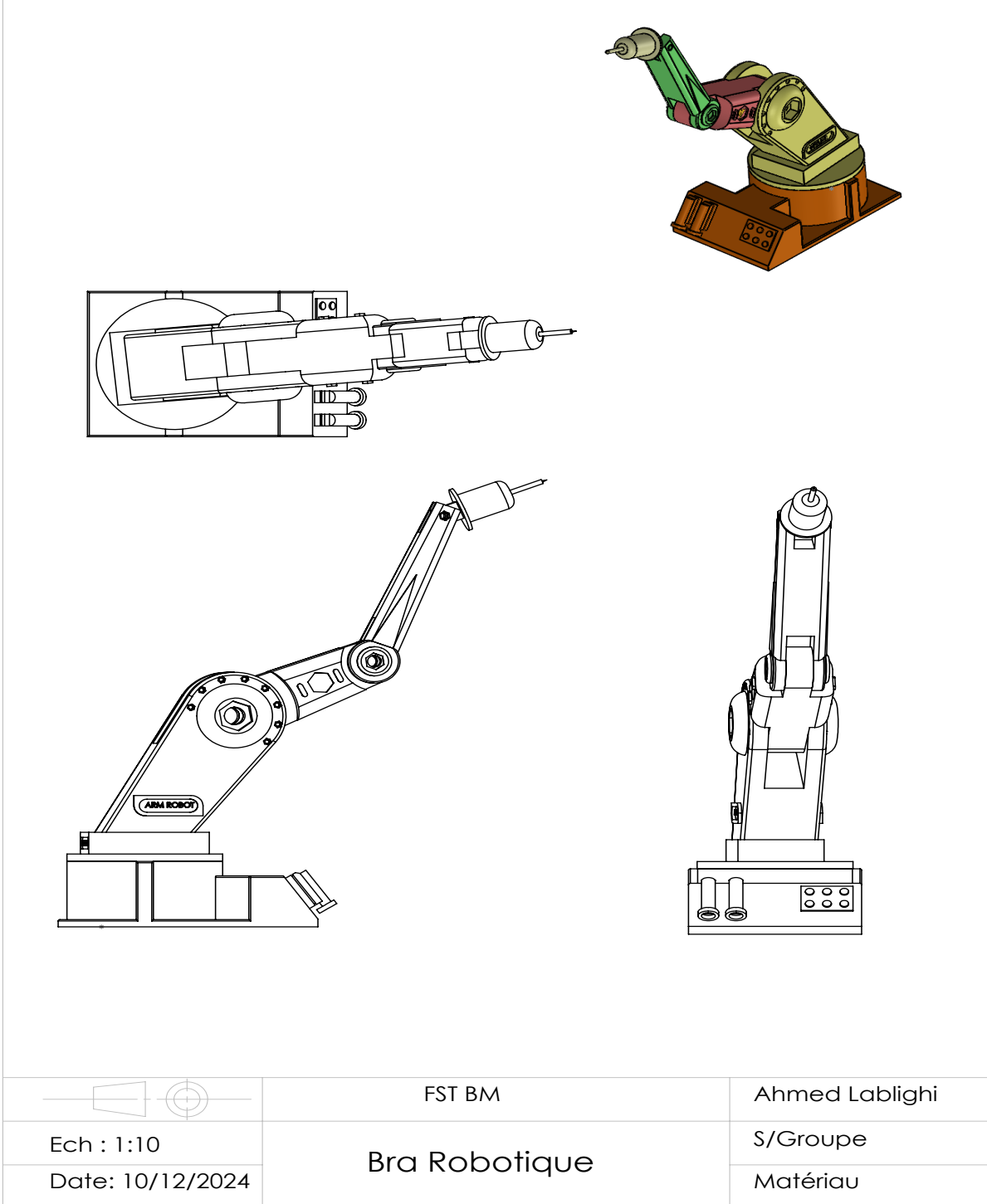
Link 3 :



Link 4 :











3.1.3 Assemblage du Modèle :



2. Présentation du package ROS exporté depuis SOLIDOWRKS :



 config	10/01/2025 17:37	Dossier de fichiers	
 launch	10/01/2025 17:37	Dossier de fichiers	
 meshes	10/01/2025 17:37	Dossier de fichiers	
 textures	10/01/2025 17:37	Dossier de fichiers	
 urdf	10/01/2025 17:37	Dossier de fichiers	
 CMakeLists	10/01/2025 17:37	Document texte	1 Ko
 export	10/01/2025 17:37	Document texte	98 Ko
 package	10/01/2025 17:37	Microsoft Edge H...	1 Ko

Dossier et fichiers disponibles dans le package exporté :

configuration: Ce dossier contient un fichier « joint_names_YourPackageName.yaml » contenant les noms des joints disponibles dans le fichier URDF qui est généré.

lancement: Ce dossier contient deux fichiers. « display.launch » qui est utilisé pour ouvrir votre robot dans Rviz. « gazebo.launch » est utilisé pour lancer votre robot dans GAZEBO.

maillages: Ce dossier contient les fichiers « .stl » de tous les liens de votre robot.

textures: ce dossier sera vide.

URDF: Ce dossier contient un fichier URDF de votre robot. Ce fichier est une description de votre robot contenant des informations sur les liens, les articulations, les positions et les plages de mouvement des articulations, leurs propriétés de masse, leurs propriétés inertielles, etc. Ce dossier contient également un fichier your_package_name.csv contenant des informations sur les liens de votre robot.

CMakeLists.txt : Ce fichier est l'entrée du système de construction CMake pour la création de packages logiciels [1]. Il décrit comment construire le code et où l'installer. Vous ne devez pas renommer ou modifier la séquence de code dans ce fichier.

exporter: Contient les journaux générés lors de la création du package dans SOLIDWORKS

paquet.xml: Ce fichier définit les propriétés du paquet, telles que le nom du paquet, les numéros de version, les auteurs, les mainteneurs et les dépendances sur d'autres paquets catkin. Les dépendances de votre paquet système sont déclarées dans package.xml. Si elles sont manquantes ou incorrectes, votre paquet peut ou non fonctionner

III. Ajoutez le package URDF à votre espace de travail et ajoutez les dépendances :

4.1 Copiez le package ROS dans votre espace de travail catkin :

Copiez le package ROS créé à l'aide de SOLIDWORKS dans le dossier « src » de votre espace de travail catkin.

Le nom de mon package est robot_arm_urdf. Je l'ai copié dans le chemin suivant :

```
~/moveit_ws/src/bras_robot_urdf
```

4.2 Modifier le fichier CmakeLists.txt

Le fichier CMakeLists.txt par défaut est très basique et ne contient pas d'autres dépendances importantes nécessaires à notre simulation. Modifiez le script dans ce fichier comme indiqué ci-dessous :

- le fichier avant les modifications :

```
cmake_minimum_required(VERSION 2.8.3)

project(robot_arm)

find_package(catkin REQUIRED)

catkin_package()

find_package(roslaunch) foreach(dir config launch meshes urdf)

install(DIRECTORY ${dir}/

DESTINATION ${CATKIN_PACKAGE_SHARE_DESTINATION}/${dir})

endforeach(dir)
```

- le fichier après les modifications :

```
cmake_minimum_required(VERSION 2.8.3)

project(robot_arm_urdf)

find_package(catkin REQUIRED COMPONENTS
  message_generation
  roscpp
  rospy
  std_msgs
  geometry_msgs
  urdf
  xacro
  message_generation
)

catkin_package(
  CATKIN_DEPENDS
  geometry_msgs
  roscpp
  rospy
  std_msgs
)

find_package(roslaunch)

foreach(dir config launch meshes urdf)

install(DIRECTORY${dir}/DESTINATION
  ${CATKIN_PACKAGE_SHARE_DESTINATION}/${dir})

endforeach(dir)
```

4.3 Modifier le fichier package.xml :

Le package.xml par défaut comporte très peu de dépendances ajoutées. Nous devons ajouter davantage de dépendances pour nos besoins de simulation.

- Le fichier avant les modifications :

```
<name>robot_arm_urdf</name>
<version>1.0.0</version>
<description>
<p>URDF Description package for robot_arm_urdf</p>
<p>This package contains configuration data, 3D models and launch files for robot_arm
robot</p>
</description>
<author>TODO</author>
<maintainer email="TODO@email.com" />
<license>BSD</license>
<buildtool_depend>catkin</buildtool_depend>
<depend>roslaunch</depend>
<depend>robot_state_publisher</depend>
<depend>rviz</depend>
<depend>joint_state_publisher</depend>
<depend>gazebo</depend>
<export>
<architecture_independent />
</export>
</package>
```

- le fichier après les modification :

```
<package format="2">
  <name>robot_arm_urdf</name>
  <version>1.0.0</version>
  <description>
    <p>URDF Description package for robot_arm_urdf</p>
    <p>This package contains configuration data, 3D models and launch files for robot_arm_
robot</p>
  </description>
  <author>TODO</author>
  <maintainer email="TODO@gmail.com" />
  <license>BSD</license>
  <buildtool_depend>catkin</buildtool_depend>
  <build_depend>message_generation</build_depend>
  <build_depend>roscpp</build_depend>
  <build_depend>rospy</build_depend>
  <build_depend>std_msgs</build_depend>
  <build_depend>geometry_msgs</build_depend>
  <build_depend>urdf</build_depend>
  <build_depend>xacro</build_depend>

  <depend>roslaunch</depend>
  <depend>robot_state_publisher</depend>
  <depend>rviz</depend>
  <depend>joint_state_publisher</depend>
  <depend>joint_state_publisher_gui</depend>
  <depend>gazebo</depend>
  <depend>moveit_simple_controller_manager</depend>
  <build_export_depend>roscpp</build_export_depend>
  <build_export_depend>rospy</build_export_depend>
  <build_export_depend>std_msgs</build_export_depend>
```



```

<build_export_depend>geometry_msgs</build_export_depend>
<build_export_depend>urdf</build_export_depend>
<build_export_depend>xacro</build_export_depend>
<exec_depend>roscpp</exec_depend>
<exec_depend>rospy</exec_depend>
<exec_depend>std_msgs</exec_depend>
<exec_depend>geometry_msgs</exec_depend>
<exec_depend>urdf</exec_depend>
<exec_depend>xacro</exec_depend>
<exec_depend>message_runtime</exec_depend>

<export>
<architecture_independent />
</export>
</package>

```

4.4 Modifiez le fichier URDF pour le rendre adapté à la simulation :

Nous devons ajouter un lien « world » et y fixer le « base_link ». Ajoutez l'extrait de code ci-dessous dans votre URDF entre les `<nom du robot="robot_arm_urdf">` et le `<linkname="base_link">` balises comme indiqué ci-dessous :

```

<robot
  nom="robot_arm_urdf">
  <link name="monde"/>
  joint>
  <joint name="base_joint" type="fixe">
  <parent link="monde"/>
  <lien enfant="base_link"/>
  <origine rpy="0 0 0" xyz="0.0 0.0 0.17"/> </
  <lien
  nom="base_link">

```

Ajoutez des balises de transmission pour chaque joint disponible dans l'URDF :

```
<nom de la transmission="lien_n_trans">
<type>transmission_interface/SimpleTransmission</type>
<joint name="joint_n">
<hardwareInterface>hardware_interface/PositionJointInterface</hardwareInterface> </
joint>
<nom de l'actionneur="lien_n_moteur">
<hardwareInterface>hardware_interface/PositionJointInterface</hardwareInterface>
<r ductionm canique>1</r ductionm canique> </
actionneur>
</transmission>
```

Ajoutez un contr leur de gazebo :

```
<gazebo>
<plugin name="control" filename="libgazebo_ros_control.so">
<robotNamespace></robotNamespace>
</plugin>
</gazebo>
```

Ajoutez des balises d'auto-collision :

```
<gazebo reference="link_n">
<selfCollide>true</selfCollide>
</gazebo>
```

4.5 Écrivez un fichier .yaml pour définir les contrôleurs ROS à utiliser pour différentes articulations et la publication des états des articulations :

Vous devez créer un fichier .yaml dans le dossier « config » de votre package ROS. Ce fichier contiendra un contrôleur d'articulation pour : les articulations du bras robotique, les articulations de l'effecteur final, la publication des états d'articulation et tout autre contrôleur selon les besoins.

```
#Instead of using TAB for indentation, use two spaces at the place of one TAB
#Controller to control robot arm joints
robot_arm_controller:
  type: "position_controllers/JointTrajectoryController"
  joints: [joint_1, joint_2, joint_3, joint_4, joint_5]
#Controller to control end effector joints
hand_ee_controller:
  type: "position_controllers/JointTrajectoryController"
  joints: [joint_6, joint_7]
#Controller to continuously publish joint states/positions
joint_state_controller:
  type: "joint_state_controller/JointStateController"
  publish_rate: 50
```

4.6 Créez un fichier .launch pour générer/ouvrir votre bras robotique dans Gazebo :

```
<launch>
<arg name="arg_x" default="0.00" />
<arg name="arg_y" default="0.00" />
<arg name="arg_z" default="0.00" />
<arg name="arg_R" default="0.00" />
<arg name="arg_P" default="0.00" />
<arg name="arg_Y" default="0.00" />
<!--Urdf file path-->
<param name="robot_description" textfile="$(find
Your_package_name)/urdf/urdf_file_name.urdf"/>
```

```

<!--spawn a empty gazebo world--><include file="$(find gazebo_ros)/launch/empty_world.launch"
/>

<node name="tf_footprint_base" pkg="tf" type="static_transform_publisher" args="0 0 0 0 0
base_link base_footprint 40" />

<!--spawn model-->

<node name="spawn_urdf" pkg="gazebo_ros" type="spawn_model" args="-x $(arg arg_x) -y $(arg
arg_y) -z $(arg arg_z) -Y $(arg arg_Y) -param robot_description -urdf -model
Your_Robot_name_defined_in_urdf_file -J joint_1 0.0 -J joint_2 0.0 -J joint_3 0.0 -J joint_4 0.0 -J
joint_5 0.0 -
J joint_6 0.0 -J joint_7 0.0" />

<!--Load and launch the joint trajectory controller-->

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

<node name="controller_spawner" pkg="controller_manager" type="spawner" respawn="false"
output="screen" args="joint_state_controller robot_arm_controller hand_ee_controller"/>

<!-- Robot State Publisher for TF of each joint: publishes all the current states of the joint, then
RViz
can visualize -->

<node name="robot_state_publisher" pkg="robot_state_publisher" type="robot_state_publisher"
respawn="false" output="screen"/>

</launch>

```

4.7 Construisez votre espace de travail pour chatons :

Ouvrez un terminal.

Accédez à votre espace de travail

```
$cd ~/moveit_ws
```

Sourcez le fichier setup.bash de votre espace de travail catkin.

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

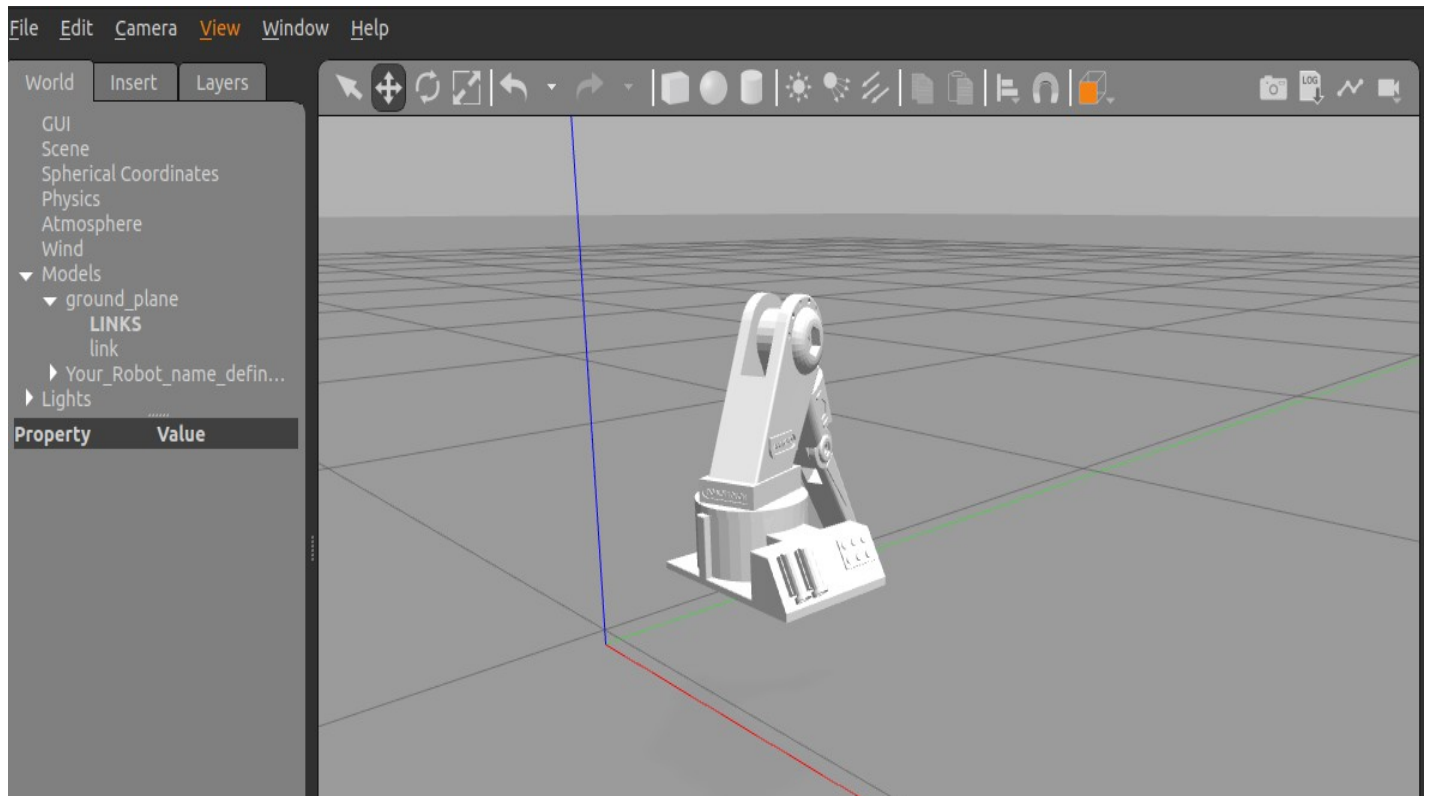
Construire l'espace de travail

```
$construction de chaton
```

Encore une fois, recherchez le fichier setup.bash.

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

4.8 Lancez le fichier de lancement de votre robot pour le charger une première fois dans GAZEBO



IV. Création d'un nouveau package de manipulateur pour contrôler le bras robotique URDF à l'aide de Moveit Setup Assistant :

Moveit est un excellent outil de manipulation de robots open source qui peut être utilisé pour créer des packages, qui peuvent être utilisés pour manipuler n'importe quel robot dans ROS

1. Lancez un nouveau terminal et accédez à votre espace de travail Catkin, où le package URDF est enregistré.

(Mon espace de travail Catkin est créé dans le répertoire personnel avec le nom « moveit_ws »)

```
$cd ~/moveit_ws
```

2. Construisez l'espace de travail si ce n'est pas déjà

fait Ceux qui utilisent la commande build :

```
$construction de chaton
```

Ceux qui utilisent la commande make :

```
$fabrication de chaton
```

3. Recherchez le nouveau fichier setup.bash dans le dossier devel de l'espace de travail.

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

Ou

```
$source ~/moveit_ws/devel/setup.bash
```

4. Lancez l'assistant de configuration Moveit

```
$roslaunch moveit_setup_assistant setup_assistant.launch
```

Avertissement : ne lancez pas l'assistance à l'installation de Moveit avant d'avoir téléchargé le fichier setup.bash de votre espace de travail.

5. Suivez maintenant les étapes indiquées dans la vidéo.

6. Lors de la génération du package à partir de moveit, je l'ai nommé « moveit_robot_arm_sim ».

V. Configuration du package Moveit pour qu'il fonctionne correctement :

