Lancement de fichiers

Conception d'un robot amphibie à pattes bio-inspirées

A. Prince BENGUET

1 Lancement de fichiers Matlab

Se diriger vers le répertoire cas_pivots_en_moins : qui contient des et fichiers Matlab ainsi que des fichiers .step qui sont utilisés dans le modèle 3D du robot en simulation sur Matlab.

Procédure de lancement :

— Ouvrir Matlab ; importer le dossier **"cas_pivots_en_moins"**, lequel doit constituer l'espace de travail.

— **CHARGER DANS LE WORKSPACE**, TOUS les dossiers et fichiers se trouvant dans le dossier **"_Control_Zmp_Com_InverseKin"** ainsi que le dossier **"PalmesFlexibles"**.

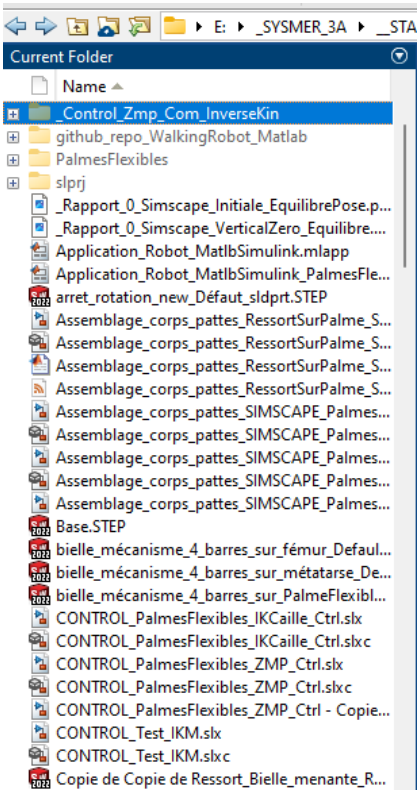


FIGURE 1 – WORKSPACE : "cas_pivots_en_moins"

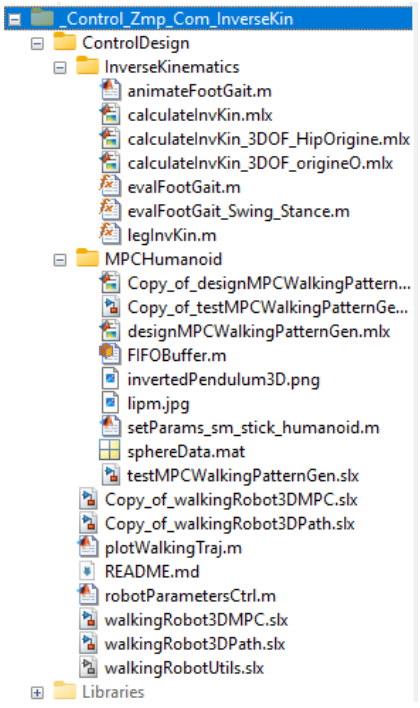
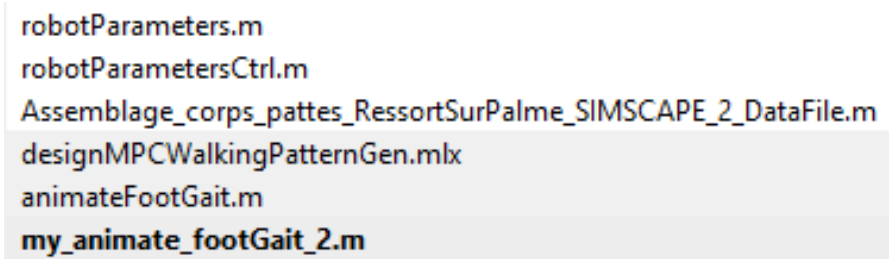


FIGURE 2 – Fichiers chargés dans le Workspace : clic droit sur un dossier contenant des fichiers et faire "Add to path"



h

FIGURE 3 – Fichiers à lancer avant simulation.

— Il y a des fichiers (se trouvant dans différents dossiers) à lancer **avant de débuter la simulation** : voir figure 3 :

1. DANS LE répertoire **"cas_pivots_en_moins"** (workspace) :
   — "robotParameters.m" PUIS lancer "robotParametersCtrl.m" (VOIR POINT 2.c)
   — "my_animate_footGait_2.m" (calculs des trajectoires à tester : une phase de trajectoire patte gauche/patte droite se faisant sur gaitPeriod)
   — "Assemblage_corps_pattes_RessortSurPalme_SIMSCAPE_2_DataFile.m" (détails sur le fichiers 3D, après import depuis solidworks)
   — OUVRIR SANS LANCER LA SIMULATION dudit fichier : "CONTROL_PalmesFlexibles_IKCaille_Ctrl.slx" (car ce fichier de simulation sera lancé correctement depuis une application Matlab GUI :"Application_Robot_MatlbSimulink_PalmesFlexibles.mlapp" se trouvant dans le Workspace)
2. DANS LE répertoire **"cas_pivots_en_moins_Control_Zmp_Com_InverseKin"** :
   (a) Précisément DANS **"ControlDesign\InverseKinematics"** : lancer "animateFootGait.m" (calculs de certaines trajectoires)
   (b) Précisément DANS **"ControlDesign\MPCHumanoid"** : lancer "designMPCWalkingPatternGen.mlx" (permettra de charger des grandeurs utiles au contrôle via MPC, LIP, ZMP).
   (c) lancer "robotParametersCtrl.m" (des paramètres)
3. Un point reste à vérifier : **IL PEUT Y AVOIR DES ERREURS** sur les noms des DES FICHIERS .STEP du modèle 3D, utilisés sous simscape multibody. figure 4.

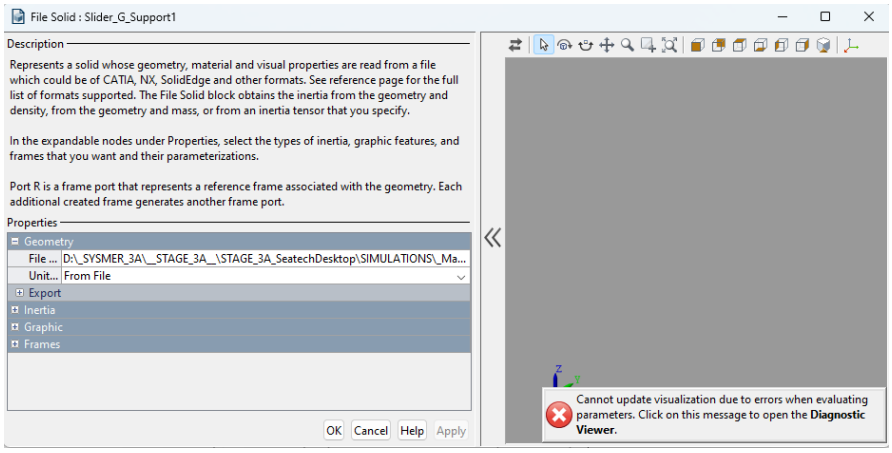
h 

FIGURE 4 – Nom d'un modèle 3D ; bloc Simulink **"File Solid".**

En effet, dans ce nom de fichier, il y a "D", pour la dénomination du disque "D" (ma clé USB) ; cette dénomination est fonction de la machine dans laquelle la clé est insérée. Ainsi, si la clé USB porte la dénomination de disque "E" dans une machine donnée il y aura erreur sur le nom du fichier (car le "D" ne sera pas changer). Il faudra alors remplacer manuellement E par D dans chaque fichier ayant ce problème.

SOLUTIONS :
Toute personne récupérant les fichiers sur drive devra changer manuellement les noms de fichiers : il faut alors ne garder que LE NOM DU FICHIER, et supprimer les chemins (adaptés à l'emplacement des fichiers sur ma clé USB) car les fichiers 3D .step sont déjà dans le même Workspace (**"cas_pivots_en_moins"**) que le fichier "CONTROL_PalmesFlexibles_IKCaille_Ctrl.slx" dans lequel ils sont lancés :

Le faire manuellement. Pour cela, Ouvrir le fichier de simulation ("CONTROL_PalmesFlexibles_IKCaille_Ctrl.slx" puis chercher le bloc simulink contenant les modèles 3D.

4. LA DERNIÈRE ÉTAPE CONSISTE À LANCER L'APPLICATION :
"Application_Robot_MatlbSimulink_PalmesFlexibles.mlapp".

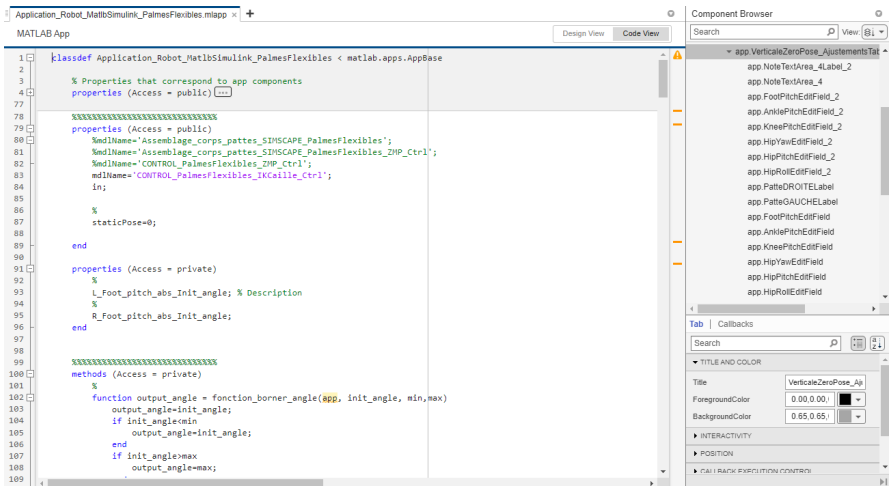
h 

FIGURE 5 – Aller dans l'onglet "Code Viewer" pour voir le code ; Interface graphique, en haut à droite ; **"mdlName"** : stocke le nom du fichier simulink (.slx) qui sera lancé.

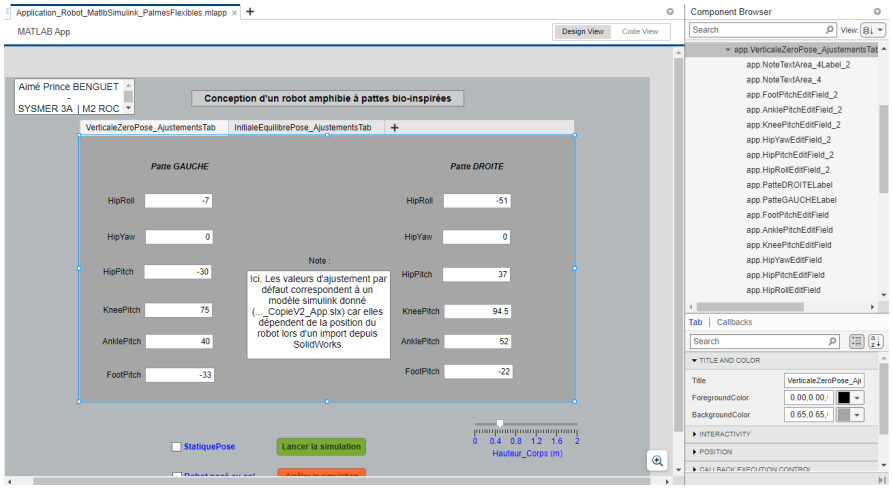
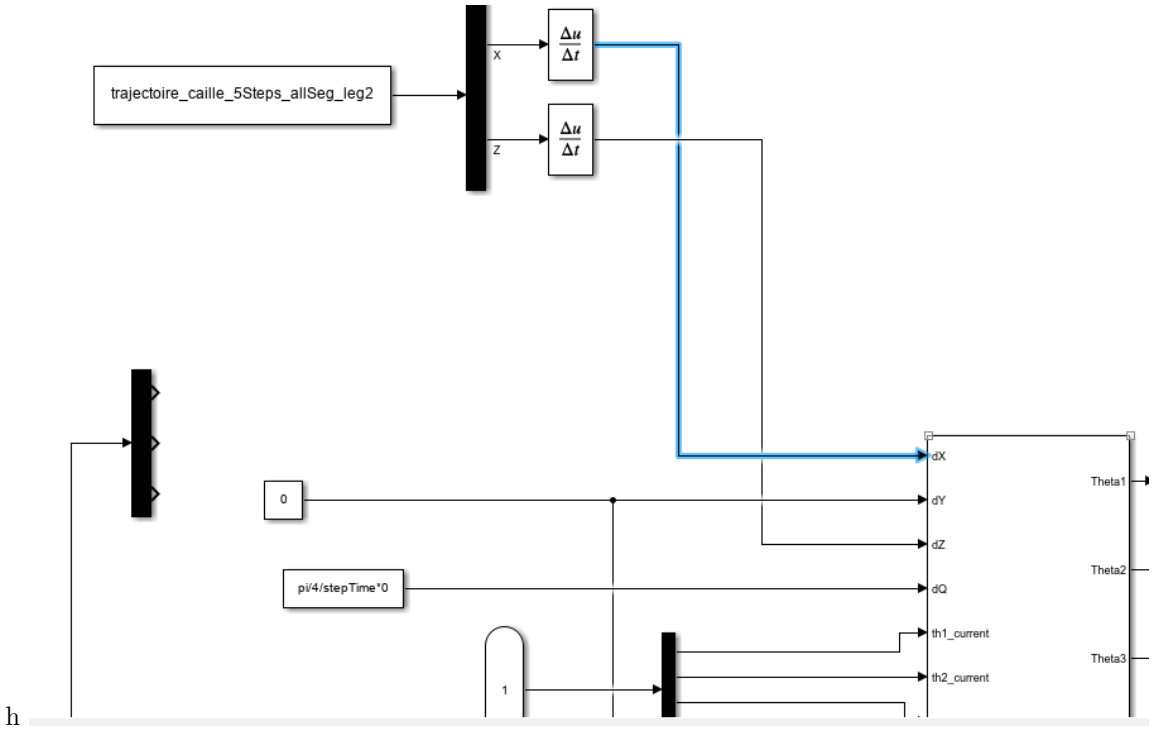
h 

FIGURE 6 – App

REMARQUE :
ON PEUT VOIR QUE L'APPLICATION PEUT LANCER N'IMPORTE QUEL FICHIER DE SIMULATION PARMI CEUX EN COMMENTAIRES ; PAR DÉFAUT, LE FICHIER LANCÉ EST CELUI sous le nom **"mdlName"**, dans **"Properties"**.

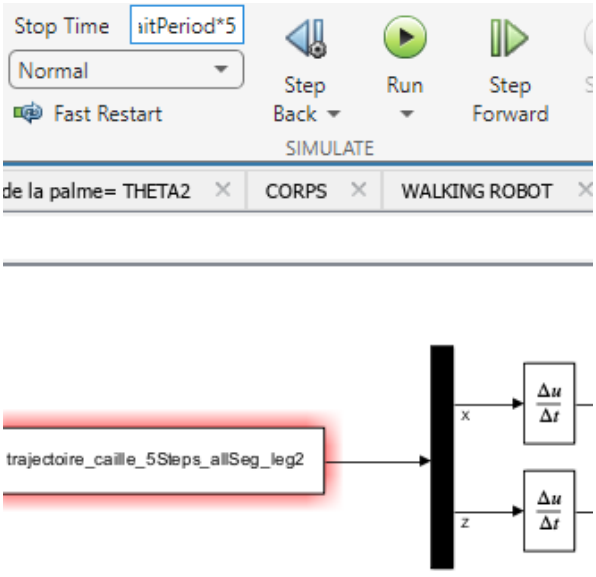
Pour un simple lancement de fichier, aucune modification n'est à faire dans le code ou les fichiers simulink (hormis le problème du renommage de fichiers .step dans les blocs simulink "File Solid").

– POUR MODIFIER LE CONTRôLE APPLIQUÉ (Ne pas suivre ces étapes s'il est simplement question de lancer la simulation pour observer mes derniers résultats) :



h

FIGURE 7 – La consigne de trajectoire de l'extrémité de la patte est issue d'un bloc "from workspace" qui récupère (depuis "my_animate_footGait_2.m" étant dans le workspace) un tableau à trois colonnes : temps, pose X (sens longitudinal), pose Z (hauteur). Cette position doit être dérivée ensuite puis insérée en **entrée du bloc** de cinématique inverse de la patte correspondant.



h

FIGURE 8 – La consigne de trajectoire précédente se trouvant dans le bloc "from workspace"(trajectoire_caille_5steps...) est une répétition (sur 4 pas) d'une trajectoire de base se faisant sur **gaitPeriod**. En **débutant par des consignes nulles** pour ne pas que le robot se déplace dès le lancement de la simulation : la consigne de trajectoire se trouve donc dans un tableau de longueur " 5 steps" (où 1 step - patteG / patteD - se fait en "gaitPeriod", et correspond, dans un tableau à une longueur donnée dans "my_animate_footGait_2.m")
Ainsi, le temps de simulation doit être un multiple de "gaitPeriod", donc 5*gaitPeriod. (4 steps de marche + 1 step au début (consignes nulles X et Z)).

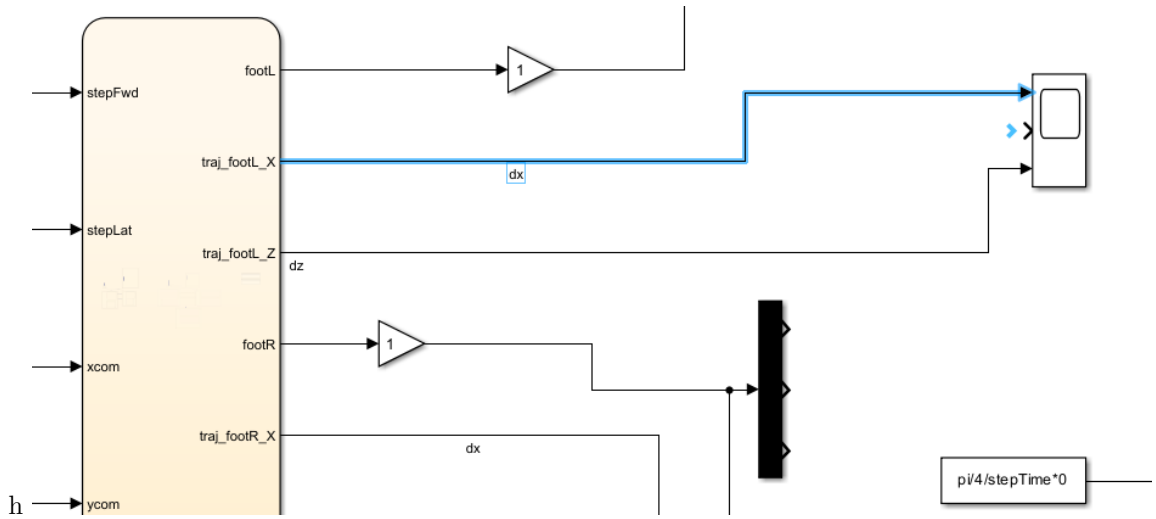


FIGURE 9 – Ici, la consigne de trajectoire de l'extrémité de la patte (traj_footL_X : vitesse suivant X) est issue du contrôle via MPC, ZMP et LIP. Cette consigne doit être envoyé comme **entrée dX** vers le bloc de cinématique inverse de la patte correspondant SI ON SOUHAITE TESTÉ LE CONTRôLE via MPC, ZMP et LIP.
Ici, le temps de simulation sur SIMULINK doit être "gaitPeriod".

2 Lancement de fichiers ROS

Ne pas hésiter à lire le tuto Notes ROS - Vrac

Le Workspace utilisé est "uuv_simulator" : chemin **_BAP/_ROS/uuv_simulator**.

2.1 Lancement de fichiers utilisant moveit pour le contrôle :

Dans un terminal (pur utiliser les services, etc) :

```
>> roscore
```

Dans un autre :

```
>> cd uuv_simulator

>> catkin_make

>> source devel/setup.bash

>> roslaunch my_moveit_robot_sim2 full_robot_sim.launch
```

Ce fichier .launch contient les lignes suivantes, qui incluent (balise "include") d'autres fichiers, notamment "my_ocean_waves.launch" (je lance le robot bipède dans un monde marin ; à lire pour comprendre les fichiers appelés).

```
<!-- -->
<launch>
    <!-- Launch Your robot arms launch file which loads the robot in Gazebo and spawns
         the
    controllers -->
    <include file =  $(find uuv_gazebo_worlds)/launch/my_ocean_waves.launch  />
```

```
    <!-- Launch Moveit Move Group Node -->
    <include file =  $(find my_moveit_robot_sim2)/launch/move_group.launch  />

    <!-- Run Rviz and load the default configuration to see the state of the
        move_group node -->
    <arg name= use_rviz  default= true  />
    <include file= $(find my_moveit_robot_sim2)/launch/moveit_rviz.launch  if= $(arg
        use_rviz) >
        <arg name= rviz_config  value= $(find my_moveit_robot_sim2)/launch/moveit.rviz
            />
    </include>

    <!--<node name= rqt_reconfigure  pkg= rqt_gui  type= rqt_gui />-->
</launch>
```

Dans le fichier `full_robot_sim.launch` du package `my_moveit_robot_sim2`, une commande `include` est utilisée pour inclure un fichier `.launch` provenant d'un autre package dans le même workspace. Voici la ligne concernée :

```
<include file =  $(find uuv_gazebo_worlds)/launch/my_ocean_waves.launch  />
```

Le package `uuv_gazebo_worlds`, qui contient le fichier `my_ocean_waves.launch`, est situé dans le même workspace que le package `my_moveit_robot_sim2`, à partir duquel cette inclusion est réalisée. Cette inclusion permet de charger le fichier `my_ocean_waves.launch` depuis le package `uuv_gazebo_worlds` dans le contexte du package `my_moveit_robot_sim2`.

2.2 Lancement de fichiers sans moveit pour tests d'hydrodynamiques :

```
>> roscore
```

Dans un autre :

```
>> cd uuv_simulator

>> catkin_make

>> source devel/setup.bash

>> roslaunch uuv_gazebo_worlds ocean_waves.launch
```

La dernière ligne permet de lancer simplement un monde marin. Il faut ensuite **Insérer un robot à l'intérieur**.

Exemple, DANS UN AUTRE TERMINAL, LANCER LE BLUEROV2 :

```
>> roslaunch bluerov2_description upload_bluerov2.launch
```

Dans notre cas, lancer le robot bipède :

```
>> roslaunch bluerov2_description upload_Ros2RobotBipede.launch
```

Dans un autre terminal, appeler les services suivants pour avoir l'effet du courant marin :

```
 rosservice call /hydrodynamics/set_current_velocity  velocity: 0.2
 rosservice call /hydrodynamics/set_current_horz_angle  angle: -0.8
 rosservice call /hydrodynamics/set_current_vert_angle  angle: 0.2
```

Ainsi, on peut lancer le robot bipède et voir son interaction avec le monde marin. Il faut cependant lire les fichiers appelés dans ces fichiers de lancement (.launch) pour comprendre les détails.

Quelques packages intéressants :
Dans le WorkSpace : uuv_simulator

_pkg : uuv_gazebo_worlds (fichiers ocean_waves.launch et my_ocean_waves.launch)
_pkg : Ros2_RobotBipede (Mon package : avec modèle 3D du robot ; voir dossiers urdf,config, scripts)
_pkg : uuv_bluerov2_description (Fichiers modifiés par moi pour **COMPRENDRE LES ASPECTS DE l'HYDRODYNAMIQUE** : Ros2RobotBipede_default.xacro (DANS le dossier "robot" ; on utilise les plugins pour créer une interaction entre des éléments 3D et le monde marin : voir plugin : **voir le code ci-dessous**), base_bipede.xacro (DANS le dossier "urdf" ; définition des paramètres : masses, volumes, etc. pour différents corps), gazebo.launch (DANS le dossier "urdf" ; définition de plugin d'hydrodynamique))

```
<xacro:Ros2_RobotBipede namespace= $(arg namespace) >
    <!-- The underwater object plugin is given as an input block parameter to
    allow the addition of external models of manipulator units -->
    <gazebo>
        <plugin name= uuv_plugin  filename= libuuv_underwater_object_ros_plugin.so >
            <fluid_density>1000.0</fluid_density>
            <flow_velocity_topic>hydrodynamics/current_velocity</flow_velocity_topic>
            <debug>$(arg debug)</debug>
            <!-- Adding the hydrodynamic and hydrostatic parameters for the vehicle --
                >
            <xacro:Ros2_RobotBipede_hydro_model link_name= base_link  volume= ${
                volume_corps}   cob= ${cob_corps}  x_box= ${x_corps_size}  y_box= ${
                y_corps_size}  z_box= {z_corps_size} />
            <!--<xacro:Ros2_RobotBipede_hydro_model link_name= Link_HipPitch_G/
                base_link  volume= ${volume_fem}   cob= ${cob_fem}  x_box= ${x_fem_size
                }  y_box= ${y_fem_size}  z_box= {z_fem_size} />
            <xacro:Ros2_RobotBipede_hydro_model link_name= Link_HipPitch_D/base_link
                volume= ${volume_fem}   cob= ${cob_fem}  x_box= ${x_fem_size}  y_box=
                ${y_fem_size}  z_box= {z_fem_size} />
            -->
        </plugin>
    </gazebo>
</xacro:Ros2_RobotBipede>
```

Bloc inséré dans le fichier "Ros2RobotBipede_default.xacro" se trouvant dans "uuv_simulator\ blue-rov2_description\robot".

```
<xacro:Ros2_RobotBipede_hydro_model link_name= base_link  volume= ${volume_corps}
    cob= ${cob_corps}  x_box= ${x_corps_size}  y_box= ${y_corps_size}  z_box= {
    z_corps_size} />
```

Sur cette ligne, on précise le lien ("link") du robot qui doit subir les effets hydrodynamiques : impossible d'appliquer les effets sur le robot en entier autrement. Ici, il n'y a que le corps portant le nom "base_link" (dans le modèle .urdf OU .xacro) qui subit les effets car les plugins de ce workspace ne permettent pas de faire autrement :**Voir code sélectionné** figure 10 :

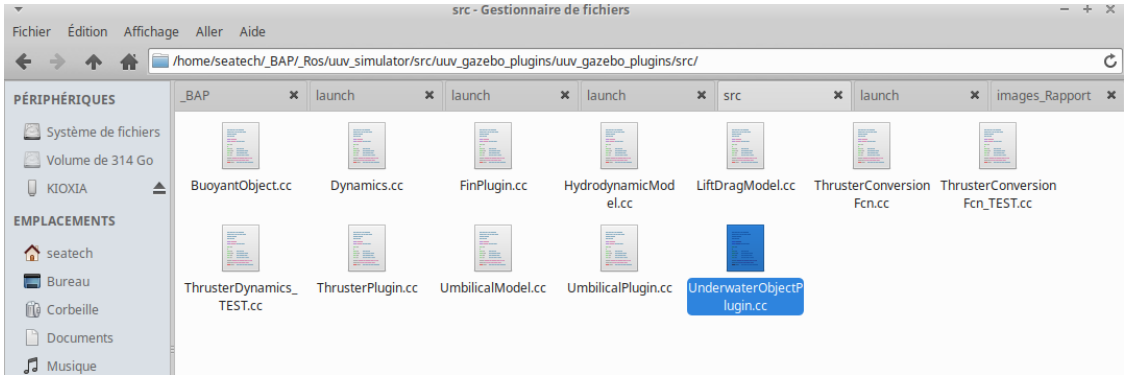


FIGURE 10 – Code c++