ENSIBS

Modélisation d'un pioneer3DX et Simulation

[Sous-titre du document]

Hadrien MERCIER
26/06/2020

Table des matières

Introduction

Ce rapport porte à la fois sur modélisation du robot pioneer3DX sous Gazebo et Rviz, sur la modélisation d'un monde pour le robot sous Gazebo, et sur la simulation de pilotage de la base mobile grâce à des capteurs ajoutés sur le robot.

I- Présentation du projet

1- Utiliser ROS et Gazebo

ROS (Robot Operating Systeme) est un outil informatique flexible pour l'écriture de logiciels de robotique. Il comprend des outils, des bibliothèques qui permettent la création d'un comportement complexe d'un robot. ROS peut utiliser pour cela une grande variété de capteurs et est compatible à plusieurs robots et à une grande variété de plates-formes robotiques.

https://www.ros.org/about-ros/

Gazebo va permettre de créer une simulation de notre robot et de son environnement puis grâce à ROS, on pourra décrire le comportement du pioneer3DX dans son environnement.

http://gazebosim.org/

2- Description du pioneer3DX réelle.

Le robot à modéliser est un pioneer3DX sur lequel, on est venu ajouter des capteurs et d'autres éléments : Une caméra 3D orbbec Astra pro, un RPLidar A2 360° de SLAMTEC, 2 Carte Odroid XU4 et un écran VU7+.

A- Informations sur les différents éléments du robot

Pioneer3DX :

Le pioneer3DX est un robot léger à deux roues et deux moteurs. Le robot est constitué d'un SONAR avant, d'une batterie, d'encodeurs de roues, d'un microcontrôleur avec le firmware ARCOS et le progiciel de robotique SDK Pioneer. Ce sont des robots fiable, durable, personnalisable et évolutif, très populaire dans le monde de l'enseignement.

https://www.generationrobots.com/fr/402395-robot-mobile-pioneer-3-dx.html

Camera 3D orbbec Astra pro :

Les cameras Astra d'Orbbec sont de puissante et fiable camera 3D avec couleur VGA, un suivi de profondeur à longue portée et hautement compatible avec des applications existantes.

https://orbbec3d.com/product-astra-pro/

RPLidar A2 360° :

Le RPLidar A2 permet de scanner l'environnement à 360° grâce à un balayage laser. Il peut scanner jusqu'à 18m à une fréquence d'échantillonnage jusqu'à 8000Hz.

https://www.slamtec.com/en/Lidar/A2

Carte Ordroid XU4 :

Plus économe, plus petite et surtout plus puissante les ODROID-XU4 sont une nouvelle génération d'appareil informatique. Elles permettent notamment d'exécuter Ubuntu 16.04.

https://www.kubii.fr/odroid/2101-carte-odroid-xu4-avec-heat-sink-kubii-3272496009844.html

Ecran VU7+ :

http://www.eu.diigiit.com/fr/odroid-vu7-plus

B- Position des différents éléments sur la basse robotique (pioneer3DX)

Sur la photo suivante on peut voir la disposition des différents éléments sur le pioneer3DX.

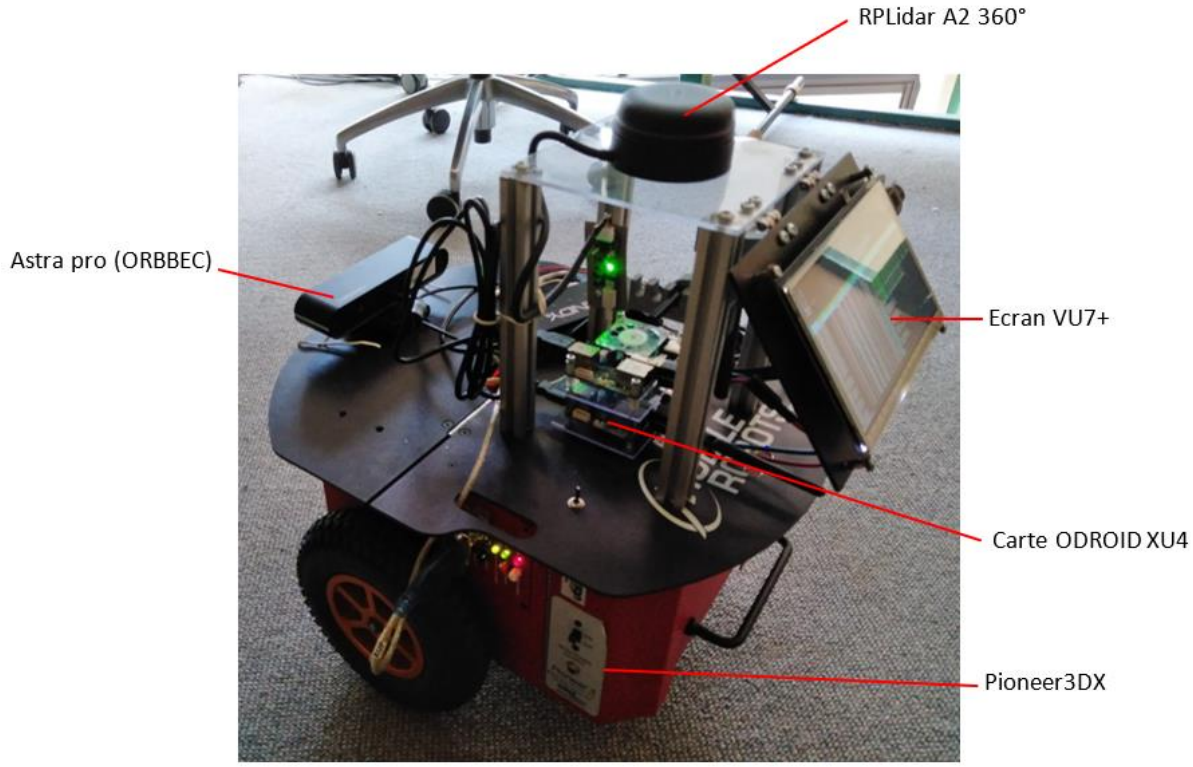

Figure 1 : Photo du pioneer3DX et des éléments ajoutés

Sur le schéma suivant on a les côtes qui permette de visualiser le placement des éléments par rapport au haut du pioneer3DX.

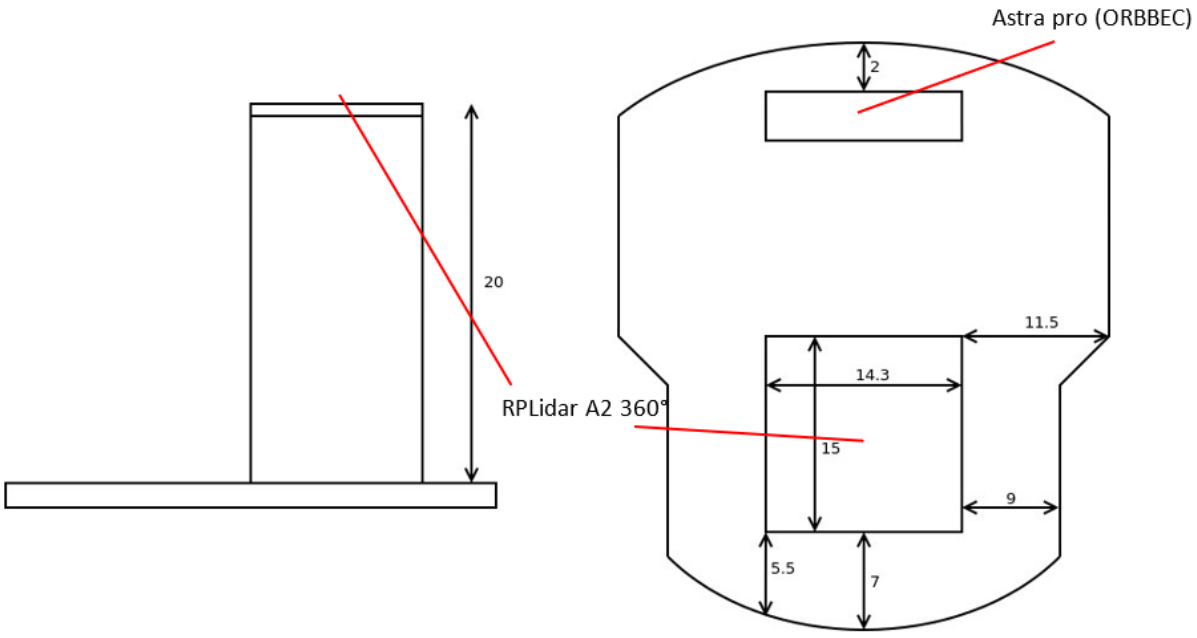

Figure 2 : schéma de positionnement des différents éléments

II- Présentation des dossiers et fichiers

Après avoir installé Ubuntu 16.04 (https://www.youtube.com/watch?v=LOn9tf-Yc4o https://ubuntu.com/download/alternative-downloads) et ROS kinetic (http://wiki.ros.org/kinetic/Installation/Ubuntu), on a téléchargé un projet qui décrit le robot pioneer3DX, un environnement dans lequel il peut se déplacer et même de quoi faire naviguer le robot dans cette environnement (http://jenjenchung.github.io/anthropomorphic/Code/Pioneer3dx%20simulation/ros-kinetic-gazebo7-pioneer.pdf). On a pu modifier les fichiers pour les adapter à notre projet. On va donc vous présenter les différents dossiers et fichiers du projet.

1- Le fichier « run_pioneer_gazebo »

Nous allons dans un premier temps nous intéresser à ce fichier, car c'est celui-ci qui va lancer les différents fichiers qui permettent de lancer la simulation du pioneer3DX sous Rviz et Gazebo. On retrouve ce premier fichier sous le dossier pioneer_gazebo_ros.

```
#!/bin/bash

my_pid=$$
echo "My process ID is $my_pid"

echo "Launching roscore..."
roscore &
pid=$!
sleep 5s

echo "Launching Gazebo..."
roslaunch pioneer_gazebo frontier_exploration_world.launch &
pid="$pid $!"

sleep 5s

echo "Launching navigation stack..."
roslaunch nav_bundle nav_bundle.launch &
pid="$pid $!"

sleep 3s

echo "Launching controller..."
roslaunch pioneer_ros pioneer_controller_spin_recover.launch &
pid="$pid $!"

echo "Launching Rviz..."
roslaunch pioneer_description frontier_map.launch rviz_name:=pioneer &
pid="$pid $!"


trap "echo Killing all processes.; kill -2 TERM $pid; exit" SIGINT SIGTERM

sleep 24h
```

Figure 3 : Fichier run_pioneer_gazebo

On observe alors ici les différents fichiers qui permettent à la fois de lancer le pioneer3DX et le monde dans lequel le robot évolue sous gazebo, de lancer le robot dans Rviz et même de lancer les programmes qui permettent de contrôler le pioneer3DX.

2- Description du Pioneer sous Gazebo et Rviz.

Sur les dossiers chargés, l'un des dossiers s'appelle « pionneer_description », dans ce dossier on va donc retrouver la description du pioneer3DX. Dans le sous dossier « Meshes » on retrouve les différentes Meshes, partie du robot. Dans le sous dossier « urdf » on va retrouver les fichiers qui décrivent l'assemblage des différentes parties du robot, les fichiers qui donnent des propriétés au partie du robot (couleur, etc.). Le dernier sous dossier « launch », contient les fichiers qui permettent de lancer le robot dans gazebo et Rviz.

A- Dossier « Meshes »

Dans ce dossier on retrouve tous les éléments du robot, les roues, le châssis, le sonar ou encore la caméra Astra et le RPLidar. Tous ces éléments sont en fait des fichiers «. stl », ce sont des meshes ou maillage fait d'éléments triangulaire. Ce dossier contient donc la description des éléments complexes du robot. Complexe car il est possible de créer des formes simples directement avec Gazebo, dans des fichiers du type SDF ou URDF.

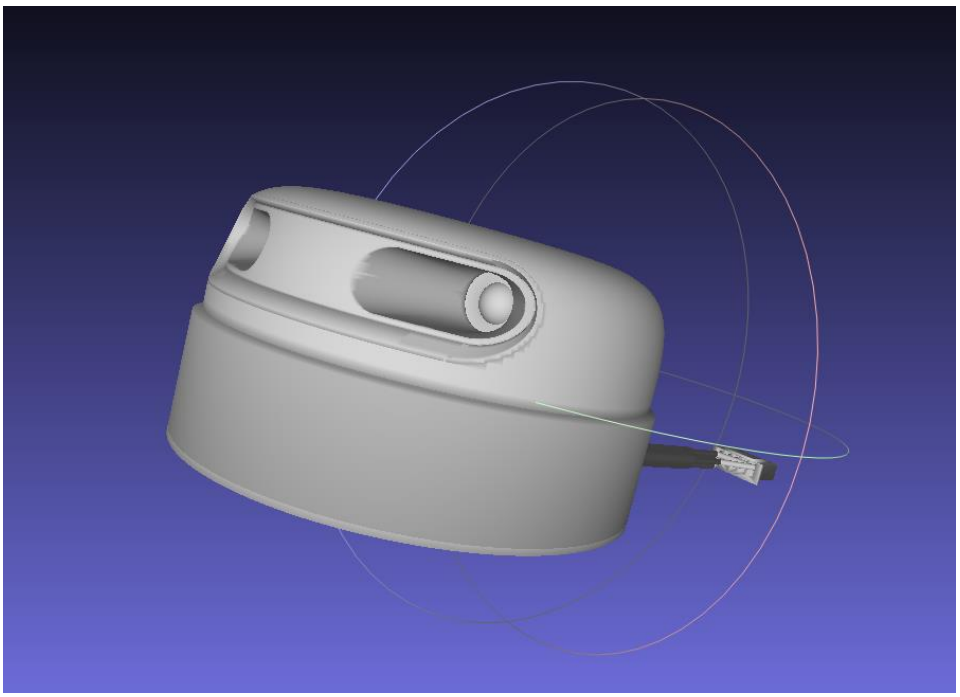

Figure 4 : Fichier «. stl » du RPLidar A2 360°

B- Dossier « URDF »

URDF (The Unified Robotic Description Format) pour ROS et SDF pour Gazebo, sont des formats de fichier XML qui décrivent tous les éléments d'un robot. Ce type de fichier permet de décrire le visuel d'un élément, permet aussi de spécifier les propriété cinématique et dynamique du robot et de ces éléments.

Le format SDF permet en plus, de spécifier la pose du robot directement dans un monde, de spécifier des frictions, et aussi de spécifier des éléments qui ne sont pas des robots comme des lumière, laser etc.

On retrouve alors dans notre dossier URDF différent fichiers :

- ***pioneer3DX_wheel.xacro*** qui décrit et spécifie l'inertie, la géométrie, le visuel, la transmission pour les roues du robot.

- ***pioneer3DX.xacro*** va spécifier la forme générale du robot c-à-d la position dans l'espace d'un élément par rapport à un autre élément. Il va aussi spécifier tous ce qui est visuel et collision.

- ***pioneer.gazebo*** est un fichier SDF qui va permettre des spécifier notamment le RPLidar et la caméra Astra qu'on ne peut pas spécifier dans un fichier URDF. Il permet aussi de spécifier les caractéristiques spécifiques à gazebo comme par exemple la couleur des pièces du robot dans gazebo.

- ***material.xacro*** spécifie juste les couleurs qui sont utilisées pour les différents élément dans pioneer3DX.xacro

Je vous conseille quelques tutoriels pour mieux comprendre comment fonctionnent les fichier URDF et comment les créer.

http://wiki.ros.org/fr/urdf/Tutorials/Create%20your%20own%20urdf%20file.

http://wiki.ros.org/fr/urdf/Tutorials.

Exemple d'une spécification classique d'une pièce d'un robot dans un fichier URDF :

On a ci-dessous un exemple de comment un élément du robot est spécifié. Ici le châssis du pioneer3DX.

```
<!-- Chassis -->
      <gazebo reference="chassis">
              <material value="Gazebo/Red"/>
      </gazebo>
```

Figure 5 : Spécification du châssis dans le fichier pioneer.gazebo

```
<!-- Base link to interface with gmapping and move_base -->
        <link name="base_link"/>

<!-- Chassis -->
  <joint name="chassis_joint" type="fixed">
    <origin xyz="-0.045 0 0.148" rpy="0 0 0"/>
    <parent link="base_link"/>
    <child link="chassis"/>
  </joint>

  <link name="chassis">
    <visual name="chassis_visual">
      <origin xyz="0 0 0" rpy="0 0 0"/>
                        <geometry name="pioneer_geom">
                                <mesh filename="package://pioneer_description/meshes/p3dx_meshes/chassis.stl"/>
                        </geometry>
                        <material name="ChassisRed"/>
    </visual>
    <collision>
                        <origin xyz="0 0 0" rpy="0 0 0"/>
                        <geometry>
                                <mesh filename="package://pioneer_description/meshes/p3dx_meshes/chassis.stl"/>
                                <!--box size="0.43 0.277 0.17"/-->
                        </geometry>
                </collision>
                <inertial>
                  <mass value="5.67"/>
                  <inertia ixx="0.07" ixy="0" ixz="0"
                    iyy="0.08" iyz="0"
                    izz="0.10"
                  />
                </inertial>
  </link>
```

Figure 6 : Spécification du châssis dans le fichier pineer3DX.xacro

```
<material name="ChassisRed">
        <color rgba="0.851 0.0 0.0 1.0"/>
</material>
```

Figure 7 : Spécification du châssis dans le fichier matérials.xacro

On observe alors différente partie dans cette spécification une partie « joint » et une partie « link » qu'on retrouve dans le fichier ***pioneer3DX.xacro***

- Le « joint » va donc lier 2 éléments du robot, on peut aussi dire qu'il va lier 2 « link ». Ces 2 « link » sont le « link » parent et le « link » enfant. Et donc « l'origine », spécifiée dans la section « joint », est l'origine du « link » enfant et ces coordonnées correspondent au déplacement par rapport à l'origine du « link » parent.

Sinon dans la partie « link », on va trouver une section « visual », une section « collision » et une section « inertia »

- Dans les sections « visual » et « collision » on importe ici notre maillage ou fichier « .stl ». Il faut donc faire attention que votre maillage ne soit pas trop lourd, qu'il soit bien en mètre (pas en millimètre) et que son orientation dans l'espace vous convienne. Si ce n'est pas le cas vous pouvez vous référer au document Adapter les fichier stl. Il est

aussi possible de créer des formes comme ici en commenté « box size = ‘0.43 0.277 0.17’ » permet de créer une boite de 43cm de long 27.7cm de large et 17cm de haut.

L’origine dans les sections « visual » et « collision » est bien l’origine du « link » enfant.

- Dans la section « inertia » on spécifie la masse et l’inertie de l’élément.

On retrouve aussi une partie « gazebo » dans le fichier ***pioneer.gazebo***, ici la section gazebo sert seulement à définir la couleur de la pièce dans gazebo

On a encore une partie « material » dans le fichier ***materials.xacro*** qui permet ici aussi de définir la couleur de la pièce.

La plupart des pièces du robot sont assemblées et spécifiées comme le châssis, mais certaine ont besoin de plus spécification c’est le cas de notre RPLidar, de la caméra Astra et des roues.

Spécification dans un fichier URDF relative à certaines pièces :

Les roues doivent tourner on va donc ajouter une spécification, la transmission d’effort entre la roue et un moteur.

```
</collision>
</link>

<transmission type="pr2_mechanism_model/SimpleTransmission" name="${parent}_${suffix}_wheel_trans">
  <actuator name="base_${suffix}_wheel_motor" />
  <joint name="base_${suffix}_wheel_joint" />
  <mechanicalReduction>${reflect * 624/35 * 80/19}</mechanicalReduction>
</transmission>

<joint name="base_${suffix}_wheel_joint" type="continuous">
  <axis xyz="0 1 0"/>
  <anchor xyz="0 0 0"/>
  <limit effort="100" velocity="100" />
  <joint_properties damping="0.0" friction="0.0" />
  <origin xyz="0 ${reflect*0.158} 0.091" rpy="0 0 0"/>
  <parent link="base_link"/>
  <child link="p3dx_${suffix}_wheel"/>
</joint>


<gazebo reference="p3dx_${suffix}_hubcap">
    <material value="Gazebo/Yellow"/>
</gazebo>


<gazebo reference="p3dx_${suffix}_wheel">
    <material value="Gazebo/Black"/>
    <elem key="mu1" value="0.5" />
    <elem key="mu2" value="50.0" />
    <elem key="kp"  value="100000000.0" />
    <elem key="kd"  value="1.0" />
</gazebo>

/xacro:macro>
```

Figure 8 : Spécification du fonctionnement des roues

Ici on aperçoit la section « transmission » qui met en action un moteur avec la roue.

http://wiki.ros.org/urdf/XML/Transmission

Dans la section « joint » on trouve de nouvelles spécifications, « axis » libère la rotation selon l'axe Y, « limit » spécifie les limites du joint en effort et vélocité é, « joint properities » spécifie l'amortissement et les frictions du joint.

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

Le RPLidar A2

Les capteurs ont leurs propres spécifications relatives à leur fonctionnement. Il n'y a pas pour tous les capteurs la référence exacte dans gazebo. Il faut donc choisir, parmi les capteurs disponibles dans ***gazebo_models,*** un capteur qui correspond au notre et qu'on va modifier et adapter aux caractéristiques du notre.

Pour le RPLidar A2 on choisit alors le sensor_plugins de type laser du capteur hokuyo.

Les spécifications relatives au fonctionnement du RPLidar que nous avons décrit sont les suivantes.

```
<!-- RPLIDAR-A2 -->
  <gazebo reference="laser">
              <material value="Gazebo/Black"/>
  </gazebo>

  <gazebo reference="laser">
    <sensor type="ray" name="RPlidar_A2">
      <pose>0 0 0 0 0 0</pose>
      <visualize>false</visualize>
      <update_rate>40.0</update_rate>
      <ray>
       <scan>
          <horizontal>
            <samples>550</samples>
            <resolution>1</resolution>
            <min_angle>-3.1415</min_angle>
            <max_angle>3.1415</max_angle>
          </horizontal>
       </scan>
       <range>
          <min>0.15</min>
          <max>12.0</max>
          <resolution>0.01</resolution>
       </range>
       <noise>
          <type>gaussian</type>
          <!-- Noise parameters based on published spec for Hokuyo laser achieving "+-30mm" accuracy at range < 10m.  A mean of 0.0m and
               stddev of 0.01m will put 99.7% of samples within 0.03m of the true reading. -->
          <mean>0.0</mean>
          <stddev>0.01</stddev>
       </noise>
      </ray>
      <plugin name="gazebo_ros_head_hokuyo_controller" filename="libgazebo_ros_laser.so">
        <topicName>base_scan</topicName>
        <frameName>laser</frameName>
      </plugin>
    </sensor>
  </gazebo>
```

Figure 9 : Spécification du fonctionnement du RPLidar A2

On utilise alors comme référence notre repère de l'élément « laser » qui correspond à la partie

physique du RPLidar. Les lasers sont ici au nombre de 550 réparties sur 360° et détecte les obstacles entre 15cm et 12m. http://gazebosim.org/tutorials?tut=ros_gzplugins&cat=connect_ros#Laser

La caméra Astra

Pour les mêmes raisons que le RPLidar nous allons ici choisir des plugins de model disponibl dans gazebo. Pour le RPLidar A2 on choisit alors le sensor_plugins de type Depth Camera de la caméra kinect disponible sur gazebo.

```
<!-- OrbbecAstra -->
<gazebo reference="OrbbecAstra_link">
  <sensor name="camera" type="depth">
    <update_rate>30</update_rate>
    <camera>
      <horizontal_fov>1.047198</horizontal_fov>
      <image>
        <width>640</width>
        <height>480</height>
        <format>R8G8B8</format>
      </image>
      <clip>
        <near>0.6</near>
        <far>8</far>
      </clip>
    </camera>
    <plugin name="camera_controller" filename="libgazebo_ros_openni_kinect.so">
      <baseline>0.2</baseline>
      <alwaysOn>true</alwaysOn>
      <updateRate>1.0</updateRate>
      <cameraName>camera_ir</cameraName>
      <imageTopicName>/camera/color/image_raw</imageTopicName>
      <cameraInfoTopicName>/camera/color/camera_info</cameraInfoTopicName>
      <depthImageTopicName>/camera/depth/image_raw</depthImageTopicName>
      <depthImageInfoTopicName>/camera/depth/camera_info</depthImageInfoTopicName>
      <pointCloudTopicName>/camera/depth/points</pointCloudTopicName>
      <frameName>OrbbecAstra_link</frameName>
      <pointCloudCutoff>0.6</pointCloudCutoff>
      <pointCloudCutoffMax>8.0</pointCloudCutoffMax>
      <distortionK1>0.00000001</distortionK1>
      <distortionK2>0.00000001</distortionK2>
      <distortionK3>0.00000001</distortionK3>
      <distortionT1>0.00000001</distortionT1>
      <distortionT2>0.00000001</distortionT2>
      <CxPrime>0</CxPrime>
      <Cx>0</Cx>
      <Cy>0</Cy>
      <focalLength>0</focalLength>
      <hackBaseline>0</hackBaseline>
    </plugin>
  </sensor>
</gazebo>
```

Figure 10 : Spécification relative à la caméra Astra pro

On définit alors l'orientation de l'image, sa taille, la largeur et la hauteur de l'écran. De quelle distance à quel distance la caméra peut filmer. http://gazebosim.org/tutorials?tut=ros_gzplugins&cat=connect_ros#OpenniKinect

C- Dossier « launch »

Dans le dossier « launch » on retrouve deux fichiers :

- ***pioneer.Rviz*** qui décrit le robot pour Rviz.

- ***frontier_map.launch*** permet de lancer le robot dans Rviz.

3- Dossier « pioneer_gazebo_ros »

A- Dossier pioneer_gazebo

Dans ce dossier on trouve les sous-dossiers et fichiers intéressant suivant :

- Le sous-dossier ***world*** contient différents mondes dans lequel on peut lancer le pioneer3DX

- Le fichier ***frontier_exploration_world.launch*** se trouvant dans le sous-dossier ***launch*** permet de lancer le robot et un environnement pour le robot sous gazebo.

```
<launch>

  <!-- these are the arguments you can pass this launch file, for example paused:=true -->
  <arg name="paused" default="false"/>
  <arg name="use_sim_time" default="true"/>
  <arg name="gui" default="true"/>
  <arg name="headless" default="false"/>
  <arg name="debug" default="false"/>
  <arg name="model" default="$(find pioneer_description)/urdf/pioneer3dx.xacro"/>

  <!-- We resume the logic in empty_world.launch, changing only the name of the world to be launched -->
  <include file="$(find gazebo_ros)/launch/empty_world.launch">
    <arg name="world_name" value="$(find pioneer_gazebo)/worlds/closed_maze.world"/>
    <arg name="debug" value="$(arg debug)" />
    <arg name="gui" value="$(arg gui)" />
    <arg name="paused" value="$(arg paused)"/>
    <arg name="use_sim_time" value="$(arg use_sim_time)"/>
    <arg name="headless" value="$(arg headless)"/>
  </include>

  <!-- Load the URDF into the ROS Parameter Server -->
  <param name="robot_description" command="$(find xacro)/xacro --inorder $(arg model)"/>

  <!-- Run a python script to the send a service call to gazebo_ros to spawn a URDF robot -->
  <node name="urdf_spawner" pkg="gazebo_ros" type="spawn_model" respawn="false" output="screen"
        args="-x 0 -y 0 -z 0 -Y 0 -urdf -model pioneer -param robot_description"/>

  <!-- ros_control pioneer launch file -->
  <!--include file="$(find pioneer_control)/launch/pioneer_control.launch" /-->

</launch>
```

*Figure 11 : Fichier **frontier_exploration_world.launch***

En modifiant le nom du monde encadrer dans le programme ci-dessus par le nom d'un autre fichier se trouvant dans le dossier ***world*** votre simulation devrait se lancer dans un autre environnement.

Vous pouvez regarder le lien suivant pour mieux comprendre les arguments de ce fichier launch : http://gazebosim.org/tutorials/?tut=ros_roslaunch

B- Dossier pioneer_ros

Dans ce dossier on trouve les sous-dossiers et fichiers intéressant suivant :

- Le fichier du sous-dossier ***launch***, ***pioneer_controller_spin_recover.launch*** permet de lancer le contrôleur du robot.

```
<launch>
        <!-- Robot controller parameters -->
  <node pkg="pioneer_ros" type="pioneer_pid_controller" name="pioneer_ros">
        <rosparam file="$(find pioneer_ros)/pioneer_controller_params.yaml" command="load"/>
        <remap from="pioneer/cmd_vel" to="controller_cmd_vel"/>
  </node>
  <node pkg="pioneer_ros" type="move_base_recover" name="move_base_recover" output="screen"/>
<!--  <node pkg="pioneer_ros" type="turn_robot_around.py" name="turn_robot_around"/>-->
</launch>
```

Figure 12: Fichier pioneer_controller_spin_recover.launch

Ce fichier va notamment lancer deux programmes, ***pioneer_pid_controller*** et ***move_base_recover*** se trouvant dans le sous-dossier ***src.*** Ces programmes n'ont pas été étudié plus dans le détail.

4- Dossier « nav_bundle »

Dans ce dossier on trouve les sous-dossiers et fichiers intéressant suivant :

- ***nav_bundle.launch*** va lancer le stack de navigation

Ce fichier va notamment lancer le fichier ***move_base.launch*** du dossier ***pioneer_2dnav*** , et les programme ***base_link_navigation_client*** et ***map_navigation_client*** *contenu* dans le dossier ***simple_navigation_goals.***

5- Dossier « pioneer_2dnav »

On a ici le package ***move_base*** qui permet la navigation du robot. Il n'a pas été étudié en détail mais le lien suivant en parle : http://wiki.ros.org/move_base

III- Résultat et Application

1- Installer et lancer le projet

Pour apprécier est voir ce que les dossiers précédents font, il faudra suivre les étapes suivantes.

- Installer Ubuntu 16.04 et ROS kinetic

- Créer et configurer dans votre dossier personnel un dossier catkin_ws

Dans un terminal lancer une par une les commandes suivantes. (Attention de changer user_name dans la commande 6)

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

2- cd~/catkin_ws/src

3- catkin_init_workspace

4- cd ~/catkin_ws

5- catkin_make

6- echo "source /home/user_name/catkin_ws/devel/setup.bash" >> ~/.bashrc

7- source ~/.bashrc
```

- Il te faudra peut-être installer des paquets additionnels

```
8- sudo apt-get install ros-kinetic-navigation ros-kinetic-slam- gmapping
ros- kinetic-ros-control ros-kinetic-ros-controllers ros- kinetic-rviz
```

Navigation package fournit des capacités de navigation par appui dans RVIz. Il prend les informations des capteurs pour atteindre un objectif. http://wiki.ros.org/navigation

Slam gmapping package utilise les lasers pour générer une carte 2D de l'environnement. http://wiki.ros.org/slam_gmapping

ROS control package contient des interfaces de contrôleur et permet la simulation d'actionneur d'un modèle robot. http://wiki.ros.org/ros_control

ROS controller package est une bibliothèque de plugin de contrôleur disponible. http://wiki.ros.org/ros_controllers

RViz package permet la visualisation 3D pour ROS. http://wiki.ros.org/rviz

- télécharger les différents dossiers du projet dans le dossier ***src*** de votre dossier **catkin_ws**.

```
10- cd ~/catkin_ws/sr

11- git clone https://gitcdr.univ-ubs.fr/ericsenn/jumeau_pioneer3DX.git
```

- Placer les 5 dossiers du dossier Projet directement dans le dossier src : ***nav_bundle, pioneer_description, pioneer_gazebo_ros, pioneer_2dnav, simple_navigation_goals.***

- Refaire le paquet catkin_ws

```
16- cd ~/catkin_ws

17- catkin_make
```

- On peut maintenant lancer notre Simulation en entrant ces 2 dernières lignes de commande dans un terminal

```
18- cd ~/catkin_ws/src/pioneer_gazebo_ros

19- ./run\_pioneer\_gazebo
```

Il se peut que la simulation ne se lance pas du premier coup. Dans ce cas fermez tout avec un Ctrl+C, Attendez que tous les processus se soit arrêté. Vous pouvez aussi utiliser la commande `killall roscore` pour arrêter le roscore. Puis finalement relancez la dernière commande `./run\_pioneer\_gazebo`.

2- Simulation

Une fois qu'on a lancé la simulation, les fenêtres Rviz et Gazebo devraient s'ouvrir comme sur les captures suivantes.

On peut observer dans les images suivantes la simulation à la fois dans Rviz et gazebo.

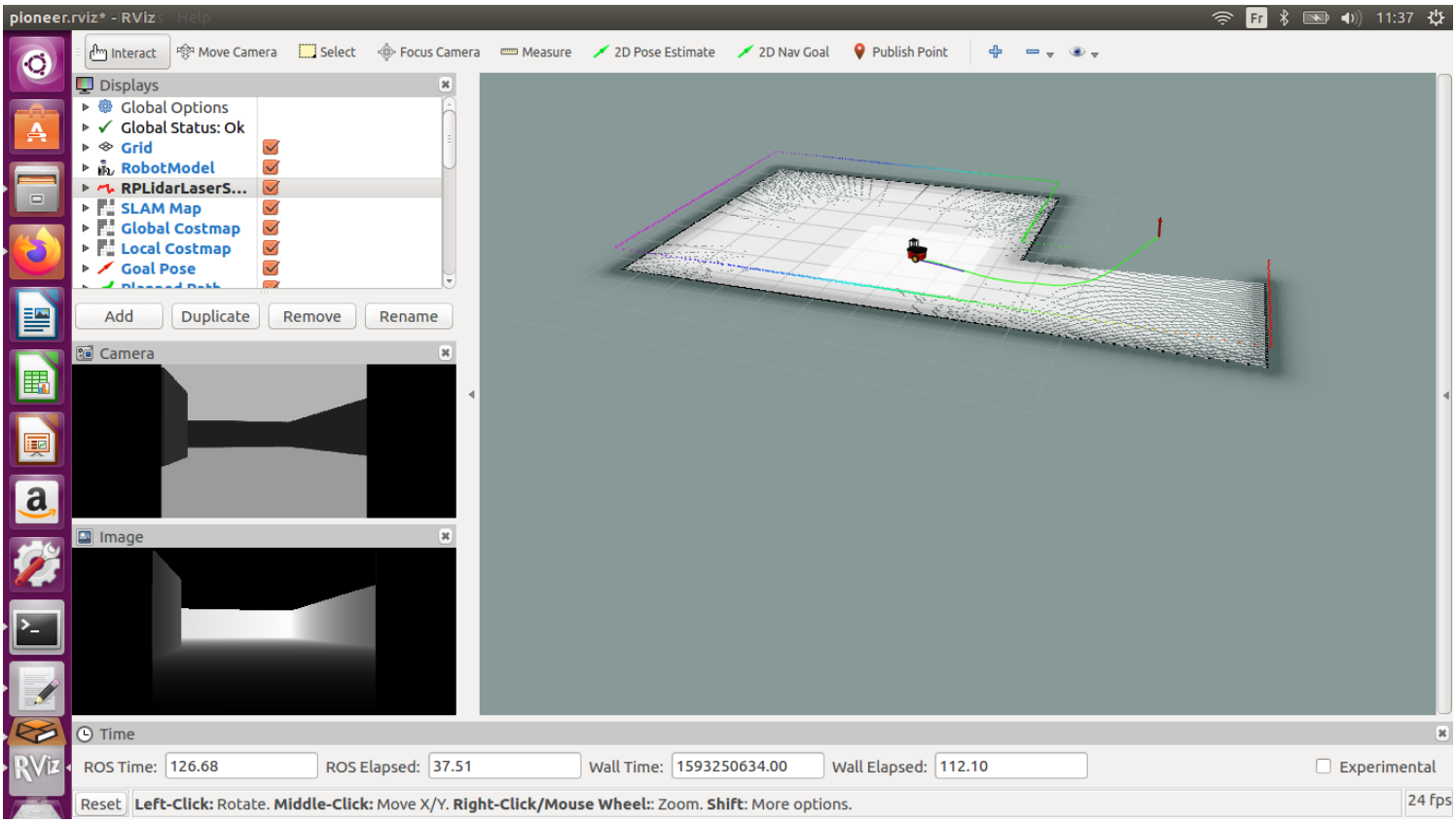

Figure 13 : Visualisation du robot dans RViz.

Dans Rviz on observe le Robot ainsi que les lasers du Rplidar et les images de la caméra, grâce à ces informations le robot va peu à peu cartographier l'environnement.

Pour faire se déplacer le robot, sélectionnez dans la barre outil ***2D Nav Goal*** et cliquez sur le terrain. Le robot devrait alors se déplacer jusqu'à cette position.

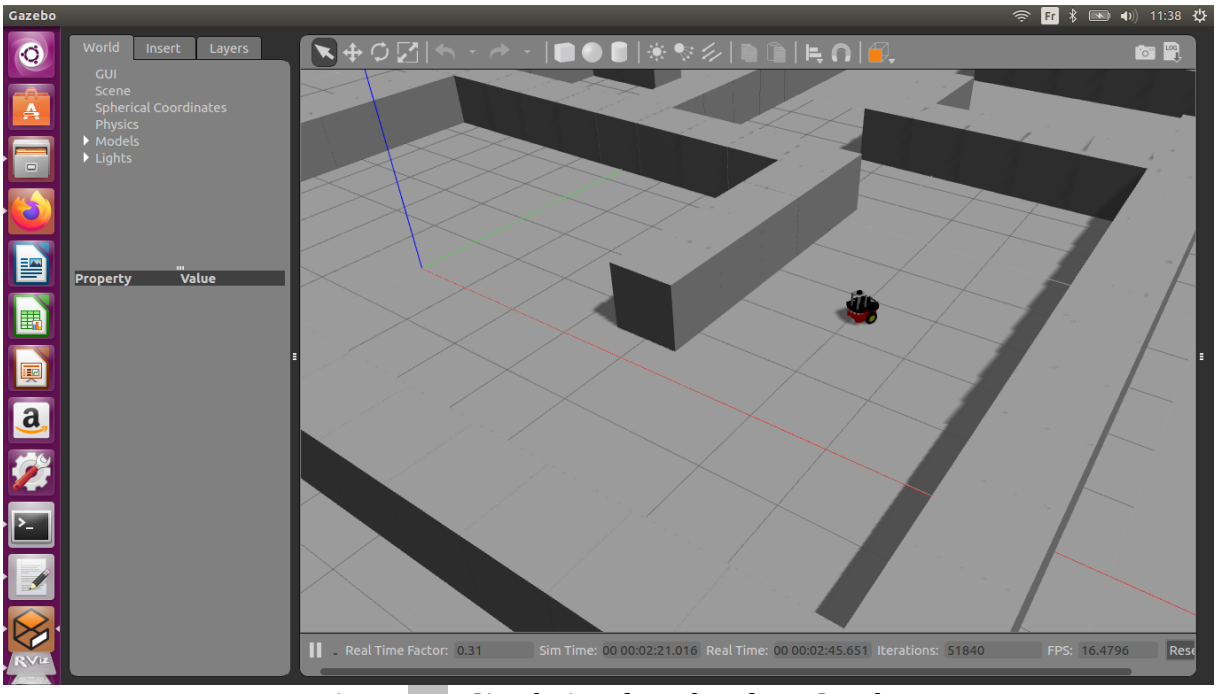

Figure 12 : Simulation du Robot dans Gazebo

Dans Gazebo on va pouvoir aussi observer le robot se déplacer.