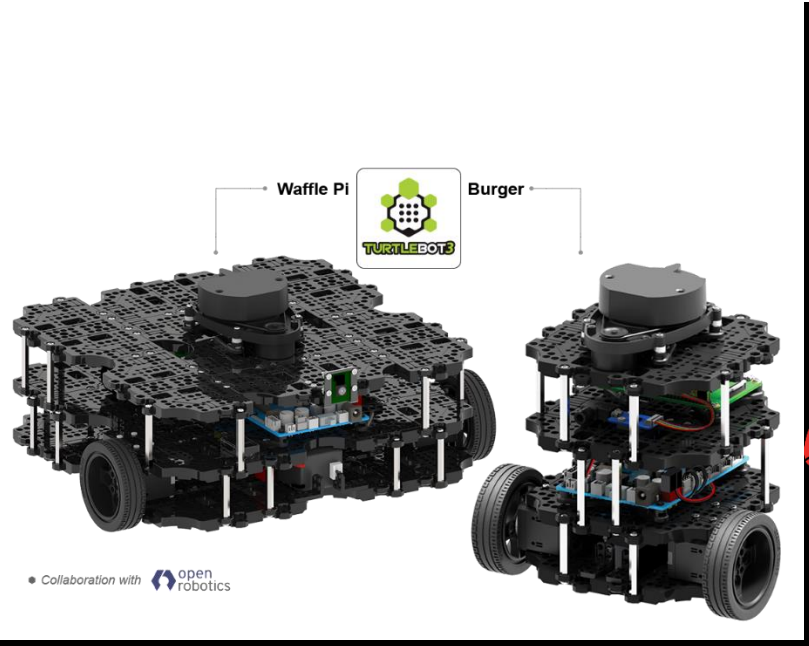

UNIVERSITÉ DE TECHNOLOGIE DE BELFORT-MONTBÉLIARD

TW53 – Robots in the loop

Rapport de projet industriel TW53 – A2018

D'ambelle Cédric – Li Crapi Mathilde – Pouzergues Albin

Département Ingénierie et Management des Systèmes Industriel

Clients
Crombez Nathan
Yan Zhi

Suiveur UTBM
Crombez Nathan
Gete Eric
Yan Zhi

utbm université de technologie Belfort-Montbéliard

Remerciements

Nos remerciements vont principalement à M. Zhi Yan et M. Nathan Crombez qui nous ont apporté une grande aide. Sans eux, de nombreuses parties du projet auraient été très longues à résoudre voire impossible pour nous d’obtenir une solution exploitable.

Ainsi, nos principales pensées se tournent vers eux. Mais n’oublions pas également M. Ouhmad Hassan qui nous a souvent permis de travailler dans la salle de robotique et également M. Sihao Deng qui nous a permis d’avoir accès à un ordinateur sous Linux.

Table des matières

Introduction

Aujourd’hui, l’industrie tend à se moderniser de jour en jour. L’un des objectifs de l’industrie actuelle est de tendre vers une industrie où les produits sont personnalisables avec des moyens de production “smart”. L’un des moyens pour arriver à ces objectifs est d’équiper le service logistique avec des robots collaboratifs.

Notre projet a pour but de pouvoir montrer à un client potentiel, ce que l’ajout d’un ou plusieurs robots apporterait à sa chaîne logistique. Afin de simuler ces AGV, nous avons utilisé des Turtlebots. Un projet antérieur s’était occupé de monter le robot.

Nous travaillerons avec le middleware ROS (Robot Operating System) sous licence ouverte. Il utilise le système d’exploitation Linux et fait appel à diverses bibliothèques.

Pour nous aider à la formation, ROS propose sur son site des tutoriels et sur internet se trouvent des bibliothèques.

I. Cahier des charges

Lors de la conception du projet, le premier élément sur lequel nous nous sommes attelés a été de formaliser les besoins du client et définir les périmètres du projet pour nous assurer que tout le monde était d'accord. Cela nous a permis de cadrer la mission.

1.1 Les acteurs du projet

Tout au long du semestre A18, étudiants, clients et suiveurs ont fait vivre le projet et principalement :

- Cédric D'AMBELLE (étudiant)
- Mathilde LI CRAPI (étudiante)
- Albin POUZERGUES (étudiant)
- Nathan CROMBEZ (suiveur/client)
- Zhi YAN (suiveur/client)
- Éric GETE (suiveur)

1.2 Le client a toujours raison, savoir l'écouter est la clé

Le projet consiste à coordonner plusieurs robots (réels et virtuels) sur le même logiciel de contrôle Ros afin d'optimiser un espace de travail collaboratif de robots mobiles industriels. L'objectif sera de concevoir de manière optimale et d'anticiper les répercussions de l'ajout d'un ou de plusieurs robots dans un espace de travail défini.

Pour ce faire nous travaillerons avec un Turtlebot Waffle Pi et nous travaillerons avec la situation initiale présentée ci-dessous (cf figure 1.2.1). Les deux robots effectuent des chargements et des dépôts sur leurs zones respectives. Les zones ont été positionnées de telle sorte que les chemins empruntés par les robots ne se croisent pas.

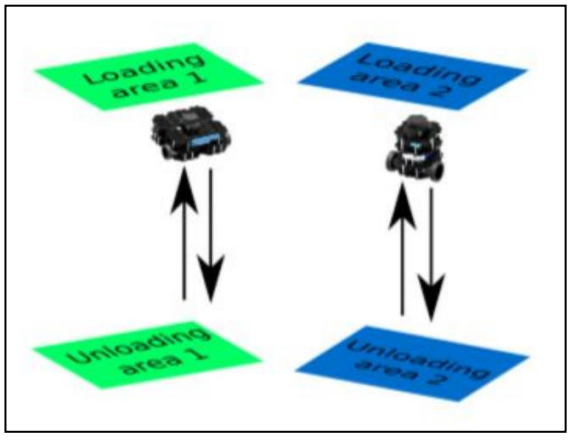

FIG. 1.2.1 - Situation initiale

Afin d'étendre, d'améliorer et de diversifier sa production, l'industriel souhaite ajouter un robot supplémentaire à son espace collaboratif (cf figure 1.2.2). Cet ajout risque de compliquer l'organisation de la zone de travail et par conséquent une étude préliminaire est indispensable afin de conserver le bon fonctionnement de l'espace de travail.

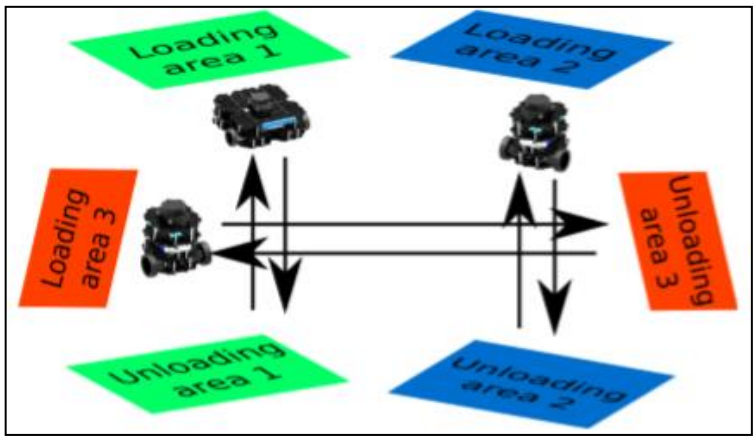

FIG. 1.2.2 - Situation souhaitée

Afin de concevoir de manière optimale et d'anticiper les répercussions de l'ajout d'un ou de plusieurs robots dans l'espace de travail, il serait très intéressant d'être en mesure de « virtualiser » l'ajout de ce ou de ces robots. Cela a ainsi été le but de notre projet.

1.3 La science des projets consiste à prévenir des difficultés de l'exécution, à travers la planification

Dans le cadre de ce projet, nous avons un semestre, soit un peu moins de quatre mois, pour réaliser les objectifs du cahier des charges. Les jalons intermédiaires sont définis sur les figures 1.3.1 et 1.3.2.

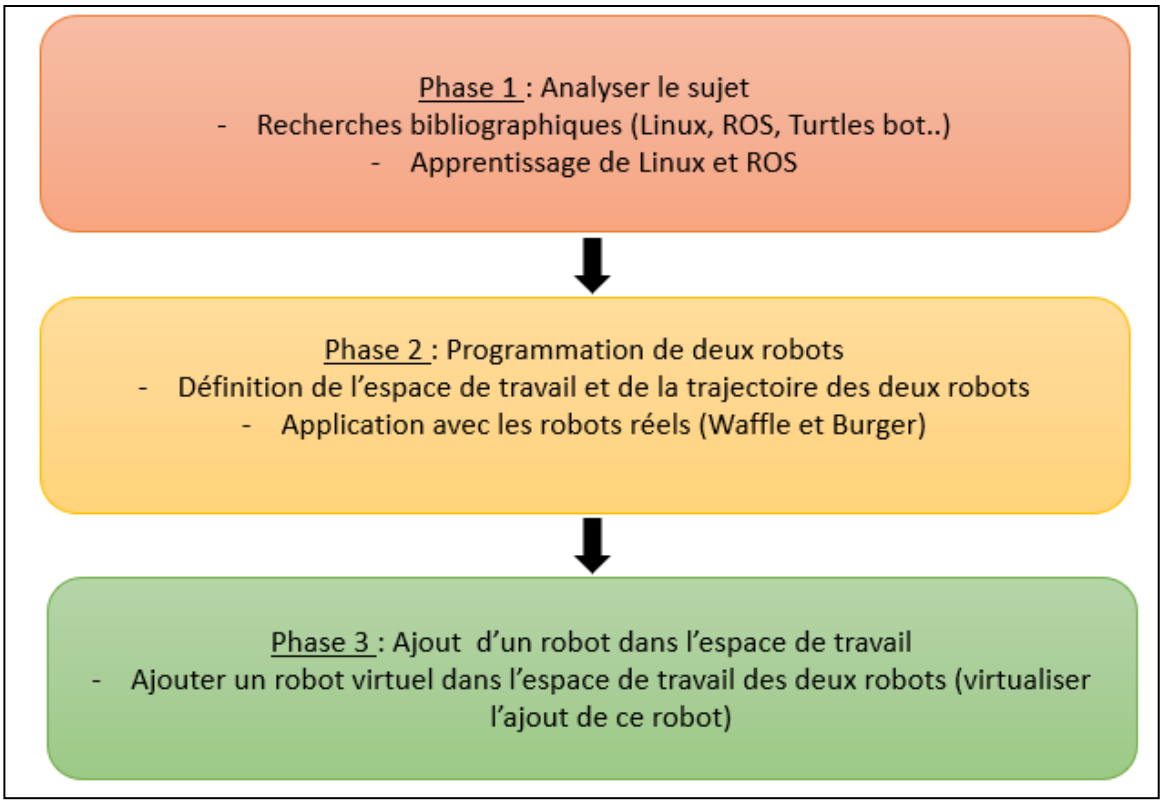

FIG. 1.3.1 – Les jalons intermédiaires

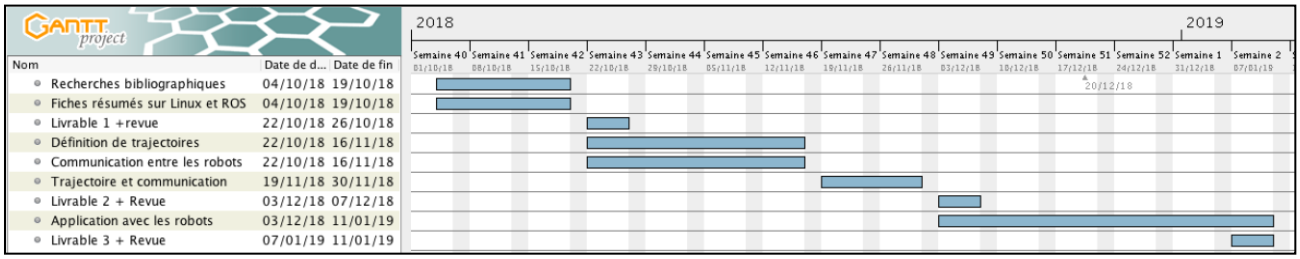

FIG. 1.3.2 – Gantt du projet

Ces plannings nous ont permis de définir les différents livrables que nous souhaitions remettre au client lors des différentes revues de projets, qui permettent à ce dernier de suivre l'état d'avancement du projet. Les livrables ainsi définis sont les suivants :

- Livrables revue 1 : prévus pour la semaine du 22 au 26 octobre 2018. Compte rendu des recherches sur le sujet et de la formation sur Ros : bibliographie, fiches résumées de Linux et Ros (format papier), petite démonstration des connaissances acquises avec Linux et ros.

- Livrables revue 2 : prévus pour la semaine du 3 au 7 décembre 2018. Démonstration de la simulation avec les robots virtuels et réels : définition de l'espace de travail des deux robots, définition de la trajectoire, application aux robots réels (Waffle et Burger).

- Livrables revue finale : prévus pour la semaine du 7 au 11 janvier 2019. Démonstration avec l’ajout d’un robot virtuel dans la zone de travail : travail collaboratif du robot virtuel avec les deux autres robots réels.

II. L'apprentissage, pour aller de l'avant

Comme dit précédemment, notre projet se déroule sous le logiciel de contrôle ROS. Les principaux clients ROS sont compatibles avec des systèmes UNIX. Ubuntu est ainsi le seul système officiellement supporté. Un OS et un environnement nouveau pour nous. Nous avons dû ainsi apprendre à maîtriser ses différents outils.

ROS, Robot Operating System, est un middleware qui a été conçu pour programmer les robots. ROS fournit une importante collection d'outils et de librairies. En complément, des outils comme Github permettent aux utilisateurs du middleware d'avoir accès à de nouvelles librairies constamment et de pouvoir partager leurs avancées. En effet, ROS a été développé de telle sorte à encourager le partage de travail.

2.1 Structure de ROS

2.1.1. Les paquets

La notion de paquet est très importante pour la compréhension de ROS. Les paquets sont essentiels à ROS car ce sont principalement eux qui font l'organisation de celui-ci. Le middleware est organisé sous forme de paquets qui peuvent contenir de nombreux éléments. On peut y retrouver, par exemple, un processus d'exécution (nœud), une librairie, des données, des fichiers de configuration. Dans ROS, le paquet est la plus petite unité de construction qui puisse être créée.

Les paquets ont tendance à tous suivre la même structure. Voici une liste des fichiers qui sont souvent présents dans les paquets :

- **include/package_name** : en-têtes d'inclusion C ++ ;
- **msg/** : dossier contenant des messages ;
- **src/package_name/** : fichiers sources ;
- **srv/** : dossier contenant des services (cf 2.3) ;
- **scripts/** : scripts exécutables ;
- **CMakeLists.txt** : fichier de construction ;
- **package.xml.**

Chaque paquet exprime un besoin. Par exemple, s'il faut calculer la trajectoire d'un robot, un paquet sera créé pour ce besoin.

2.1.2. Le master

Le master de ROS permet aux nœuds de s'enregistrer. Son rôle est de permettre à ceux-ci de se localiser. Grâce au master de ROS, les nœuds sont en capacité de communiquer entre eux (cf 2.3). Sans le master, les nœuds ne sont pas capables de se trouver les uns les autres et donc ne peuvent pas communiquer entre eux.

Le master contient également le « parameter server ». Celui-ci contient des données utilisées comme paramètres.

2.1.3. Les nœuds

Un nœud est un processus qui permet d'effectuer un calcul. Chaque nœud a un but. Par exemple, il y a un nœud qui permet de déplacer un robot jusqu'à un point, un deuxième nœud permettant l'arrêt du robot.

Un système de contrôle du robot comprendra plusieurs nœuds. Il est donc nécessaire que ceux-ci communiquent entre eux. Tous les nœuds en cours d'exécution doivent avoir un nom unique.

Un nœud peut être créé en écrivant un morceau de code en Python par exemple.

2.2 Quelques commandes utiles

Il existe de nombreuses commandes dans ROS, nous avons recensé celles qui nous paraissent les plus importantes lors de l'utilisation de ROS.
Voici quelques commandes liées à la notion de paquet :

- **catkin_create_pkg** : permet de créer un nouveau paquet ;
- **rospack** : permet de donner les informations concernant un paquet ;
- **catkin_make** : permet de compiler un ensemble de paquets ;
- **rosdep** : permet de créer un système de dépendances d'un paquet ;
- **rqt** : permet de visualiser les dépendances d'un paquet sous forme de graphique.

Nous pouvons également trouver des commandes liées au master :
- **roscore** : permet de démarrer ROS et de créer le master.

Lorsque nous lançons cette commande, nous pouvons voir sur la dernière ligne **rosout**. Rosout est ce qui permet d'avoir une interface avec le « cœur » de ROS.

Tout comme pour les paquets, les nœuds ont des commandes qui leur sont associées :
- **rosnode info** : retourne les informations concernant le nœud ;
- **rosnode kill** : permet d'arrêter un nœud ;
- **rosnode list** : permet d'afficher la liste des nœuds actifs ;
- **rosnode ping** : permet de tester la connexion entre les nœuds.

D'autres commandes utiles :
- **rostopic list** : retourne la liste des topics actifs
- **rosrun [package_name][node_name]** : permet de lancer un fichier exécutable

2.3 Communication interne dans ROS

Comme expliqué précédemment, les nœuds sont en lien direct avec le master. Nous allons maintenant regarder comment ils font pour communiquer entre eux.

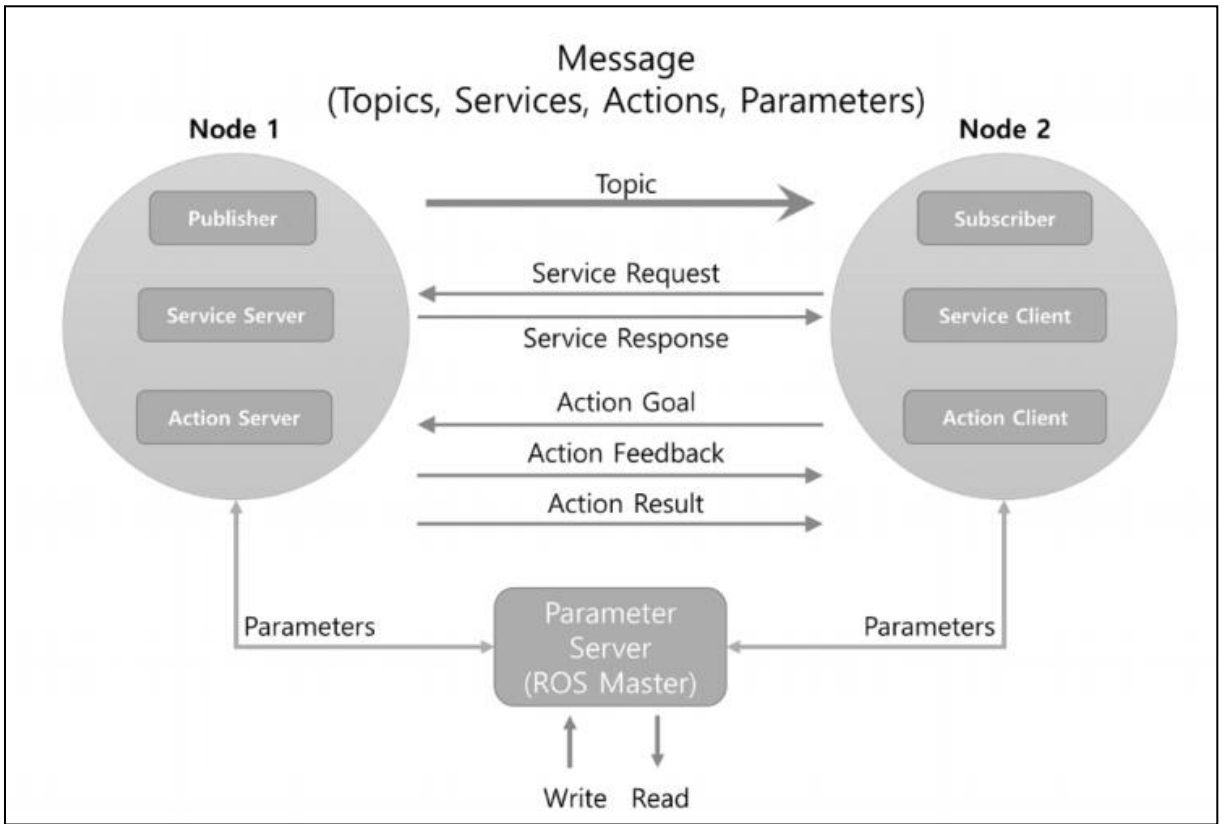

FIG. 2.3.1 – Différents types de messages dans ROS

Les nœuds communiquent entre eux avec des messages. Il existe trois types de messages qui ont chacun leur particularité.

Qu'est-ce qu'un message ? Les messages sont des variables (entiers, flottants, chaîne de caractères, …). Un message peut contenir plusieurs variables. Quels sont les différents types de messages ?

- Topic : (littéralement comme un sujet de conversation). Le *publisher node* enregistre d'abord le *topic* au *master*. Les autres *nodes* ont juste à s'y abonner. Ils recevront les informations du *publisher* par contre ils ne peuvent pas demander d'informations (unidirectionnelles). On utilise ce type de message quand les nodes s'échangent des données tout le temps.

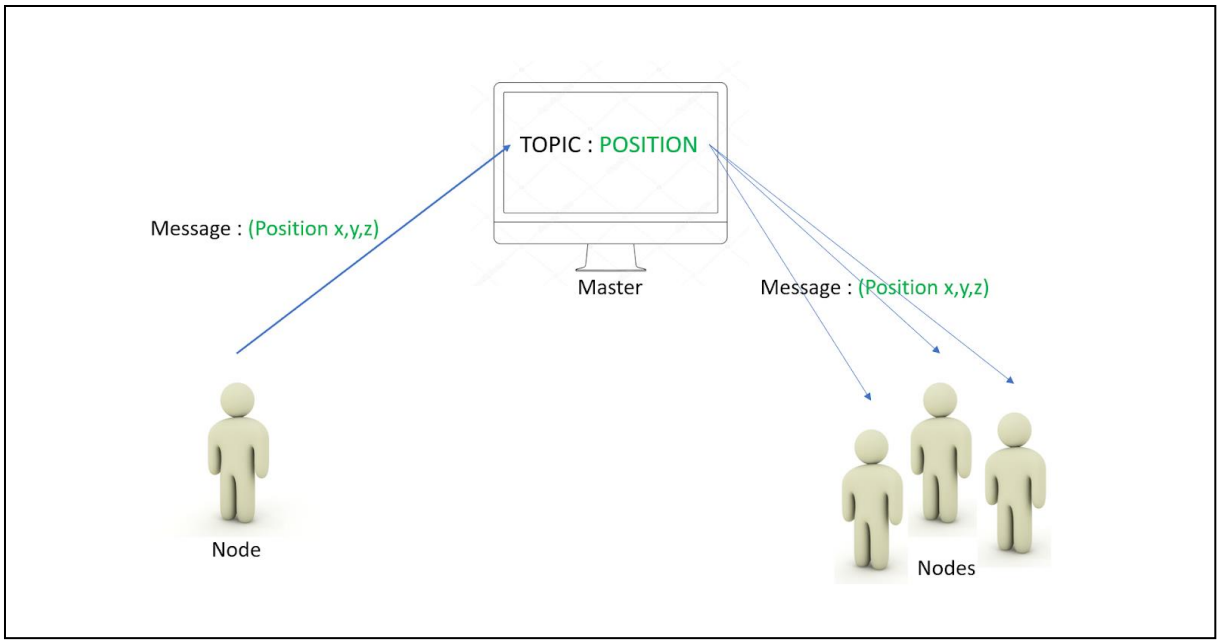

FIG. 2.3.2 – Exemple du type de message Topic

- Service : pour utiliser ce type de messages, il faut un *service server* et un *service client*. Le serveur répond aux demandes ; le client transmet les demandes. Une fois la réponse transmise, le lien est coupé entre les deux *nodes*. Ici, il s'agit d'un lien bidirectionnel. Il est avantageux d'utiliser ce type de message lorsque l'on souhaite connaître l'état du système au moment de la demande.

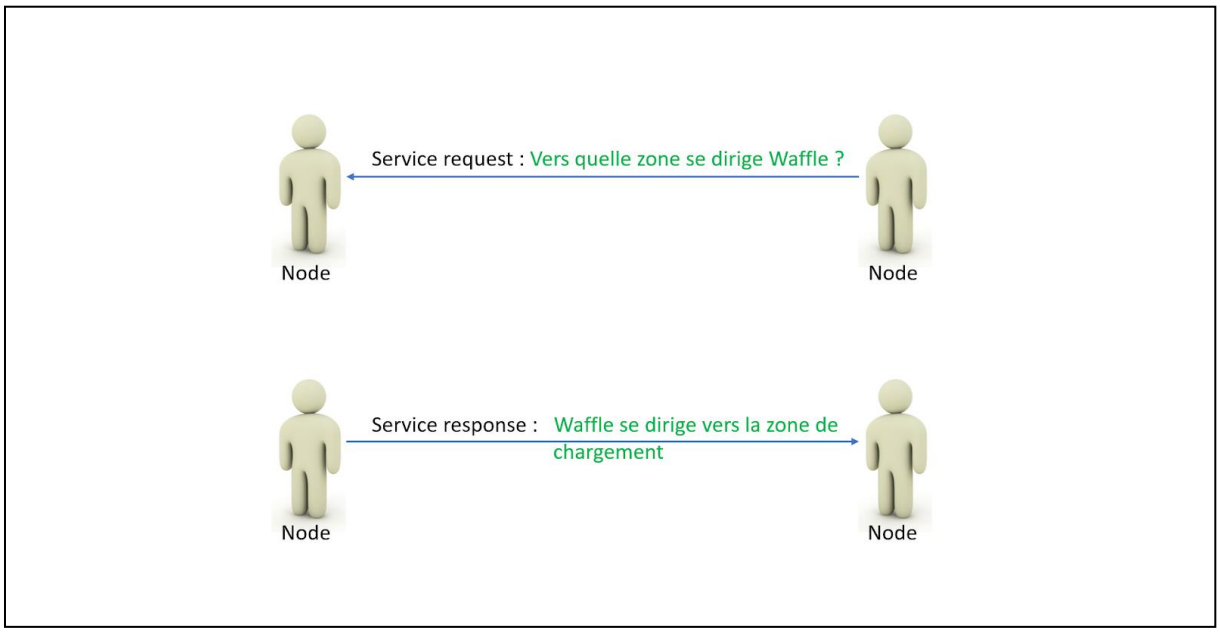

FIG. 2.3.3 – Exemple du type de message Service

- Action : ils fonctionnent comme le service. Le serveur exécute l'action que le client a demandée. Ils sont utilisés lorsque le temps de réponse est long et/ou quand on souhaite avoir un feedback (valeur intermédiaire demandée par exemple).

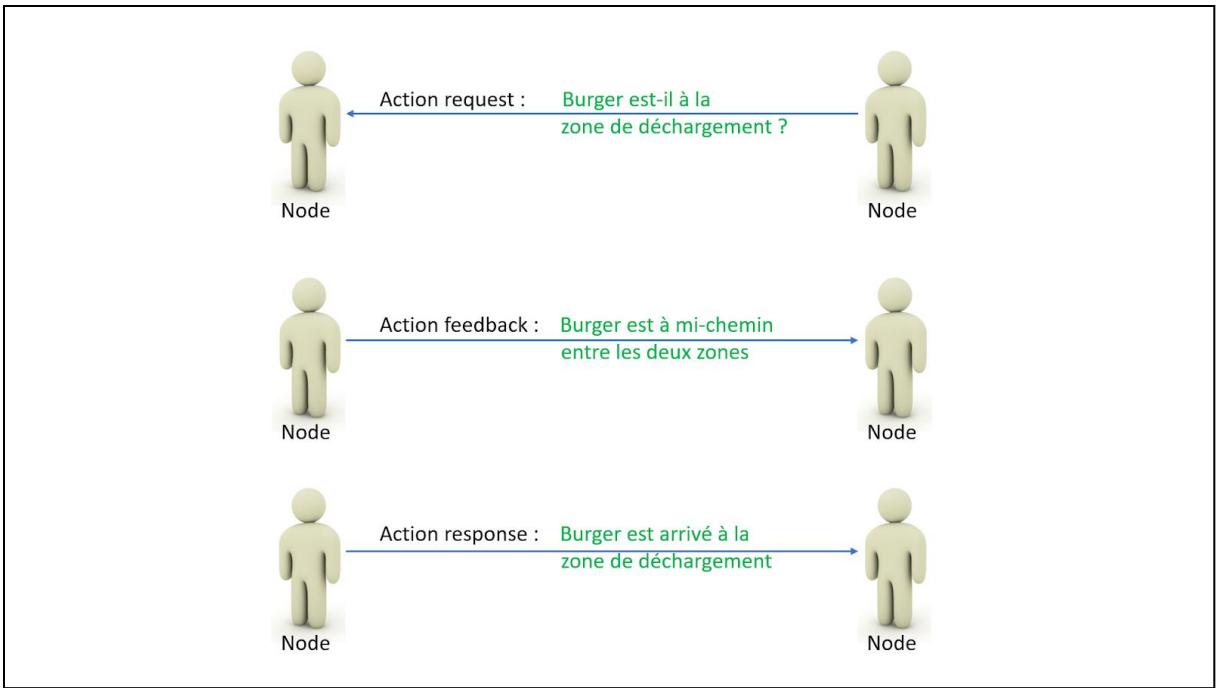

FIG. 2.3.4 – Exemple du type de message Action

Comme on peut le voir sur la figure 2.3.1, il existe aussi un *parameter server*. Nous pouvons utiliser les paramètres comme messages. En effet, les paramètres sont des variables globales (dans le master) qui sont utilisées dans les *nodes*. Ces variables sont stockées dans le *parameter server* du master et peuvent être utilisées comme message entre deux *nodes*.

III. Répondre aux objectifs

Après avoir appris à nous servir de l'invité de commandes d'Ubuntu et nous être familiarisé avec les commandes ROS et son fonctionnement, il était temps pour nous de nous concentrer sur les souhaits à proprement parler du client. La partie la plus motivante et aussi la plus complexe.

3.1 Le Turtlebot côté hardware

Un de nos principaux outils de travail a été le Turtlebot. Nous avons travaillé avec la dernière génération de robots, les Turtlebot3 et il existe trois modèles dans cette famille. Le Burger, le Waffle et le Waffle Pi (figure 3.1.1).

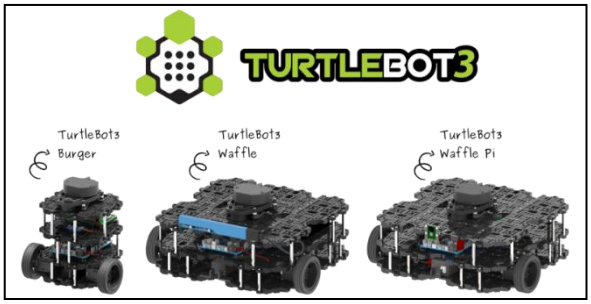

FIG. 3.1.1 – Différents modèles de Turtlebot3
source : http://emanual.robotis.com

Que ce soit en simulé ou en réel nous avons principalement eu à faire au Turtlebot Waffle Pi qui est composé d'une belle armada de capteurs. Nous avons par exemple sur le robot un LIDAR 2D 360°. Ce dernier est utilisé notamment pour SLAM et la navigation. Nous avions également une caméra Raspbery Pi, un microcontrôleur Raspbery Pi ainsi que deux servomoteurs.

Ces éléments nous ont permis par la suite de récupérer des positions ou encore permis au robot de se repérer dans son environnement, par exemple l'odom avec les transformées adéquates émanant des servomoteurs. Les données récupérées par ces capteurs furent très utiles.

3.2 Le démarrage

A ce stade l'idée était simplement de démarrer/lancer les différents éléments qui nous seraient utiles dans le projet.

3.2.1. La connexion au robot réel

Le robot et l'ordinateur communiquent par Wi-Fi. Nous devons avoir un accès déporté par rapport au robot pour quelques raisons simples. Tout d'abord, le processeur du robot ne peut pas gérer la multitude de calculs nécessaires lors de notre projet. Un autre point est que nous avons besoin de simuler des robots donc ne pas être présent physiquement sur le robot prend tout son sens.

Pour la connexion du robot, il a été nécessaire d'installer un routeur Wi-Fi. Ce dernier est notre bus d'instrumentation afin d'assurer la connexion entre l'ordinateur et notre Turtlebot Waffle Pi.

Nous devions nous connecter à ce réseau local et il nous fallait une adresse IP fixe. Ce dernier est nécessaire car nous voulions la conserver lorsque nous nous connections au réseau (figure 3.2.1.1)

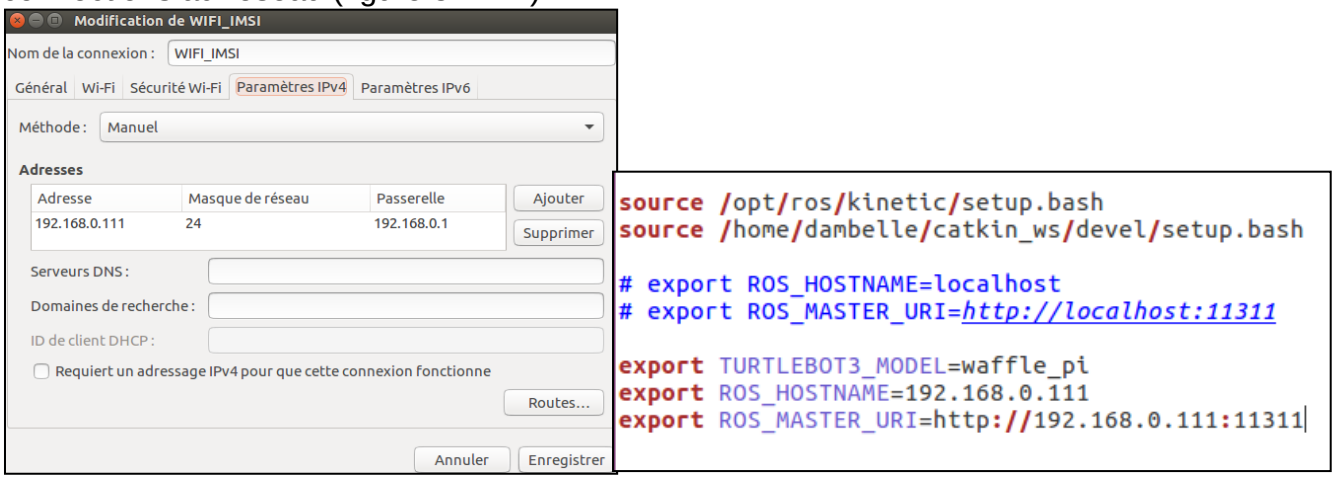

FIG. 3.2.1.1 – A gauche, configuration de l'adresse IP fixe et à droite, variables d'environnement du bash

Pour pouvoir nous connecter au robot, il fallait associer les bonnes valeurs aux variables d'environnement dans le bashrc. Le bashrc est un script shell qui s'exécute à chaque fois qu'un nouveau shell est ouvert. Cela nous permet en autre d'automatiser des commandes qui devraient être lancées à chaque nouveau shell. Sur la figure 3.2.1.1, à droite, on constate donc qu'on initialise le modèle du robot souhaité et l'adresse IP qu'on souhaite initialiser. En commentaire nous avons l'adresse IP, que nous devons « dé-commenter », lorsque l'on se connecte sous un autre réseau que le réseau WIFI IMSI.

Pour accéder au robot Turtlebot3, il suffit de se connecter avec la commande ssh. C'est un protocole de connexion sécurisé permettant à l'ordinateur d'exécuter des commandes à partir du shell pour par exemple interagir avec le robot (figure 3.2.1.2). Le mot de passe du pi@192.168.0.5 est turtlebot.

FIG. 3.2.1.2 – Connexion avec le Turtlebot3

La synchronisation entre le robot et l'ordinateur est également un critère important. Nous avons voulu l'effectuer à partir du protocole NTP mais cela n'a malheureusement pas marché malgré de nombreux essais.

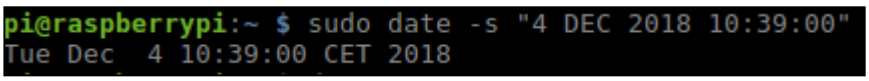

FIG. 3.2.1.3 – Synchronisation manuelle horloge entre ordinateur et robot

Ce dernier est un protocole informatique qui permet de synchroniser l'horloge locale de l'ordinateur avec le robot. Nous devions donc l'effectuer manuellement à chaque connexion sur le robot (figure 3.2.1.3).

3.2.2. Faire mouvoir le robot réel

Nous avions différents leviers d'actions pour faire bouger le robot. C'est même un peu plus loin dans le projet une question importante à laquelle nous nous sommes confrontés. Ici les moyens sont simples car c'est « manuellement » que nous faisions bouger le robot.

La toute première est la plus intuitive est la manette fournie avec le Turtlebot qui est semblable à celle des consoles de jeux vidéo. C'est notamment grâce à ce dernier que nous avons réalisé le SLAM de la carte.

Ensuite de nombreux autres moyens existent et nous les avons essayés. Il y avait par exemple les « Interactive Marker » sur Rviz (figure 3.2.2.1), l'élément « move to goal position », etc.

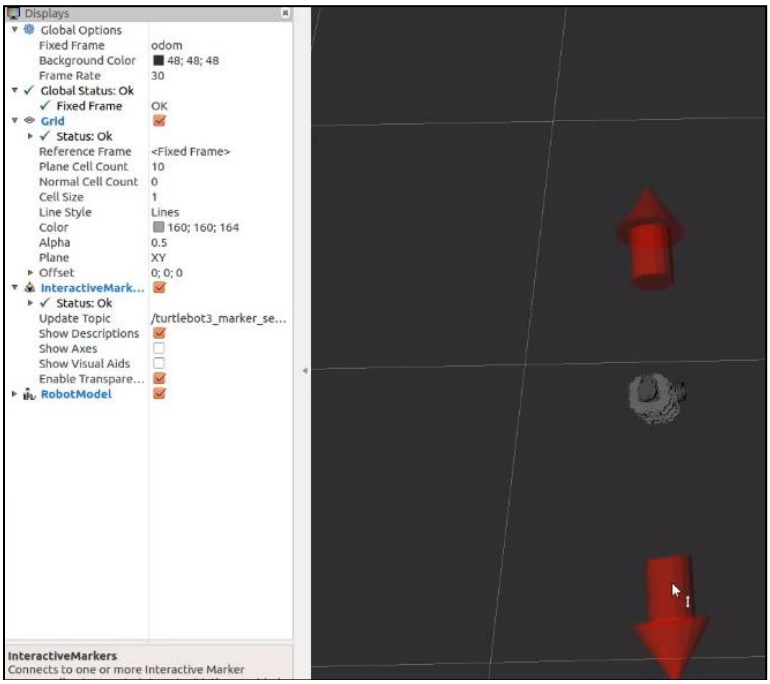

FIG. 3.2.2.1 – Exemple avec les Interactive Marker

De nombreux programmes existent également, nous avions ainsi essayé par exemple de bouger le robot en ligne droite et de lui faire faire un retour avec une confirmation de l'utilisateur (cf Annexe 1 : Programme mouvement Linéaire).

Dans ce programme, nous déplaçons le robot en publiant sur le topic /cmd_vel. Effectivement, nous contrôlons sa vitesse suivant l'axe x. Le robot ne peut donc pas contourner un obstacle si jamais il venait à y en avoir un. Dans ce cas, le robot se déplace selon la distance choisie par le programmeur. En effet, nous ne donnions pas de point à atteindre.

Nous demandions au robot de se déplacer entre deux points séparés par une distance que nous avons définie dans le programme. Nous avons imaginé que cette distance séparait la zone de chargement et la zone de déchargement. L'utilisateur devait dire au robot si le chargement ou déchargement était fini pour que le robot aille à l'autre point.

Nous avons donc changé l'approche en ce qui concerne le déplacement vers un point donné.

3.2.3. Gazebo

Gazebo est un outil de simulation 3D de différents scénarios robotiques possibles. Lors de nos premières manipulations de cet outil nous avions voulu comprendre le fonctionnement. Nous avons donc lancé un launch permettant d'avoir trois Turtlebot3 dans un environnement prédéfini, dans notre cas une maison. Même si cela est hors du cadre de Gazebo, nous voulions comprendre l'interaction des différents topic et node relatif au lancement de plusieurs robots (figure 3.2.3.1).

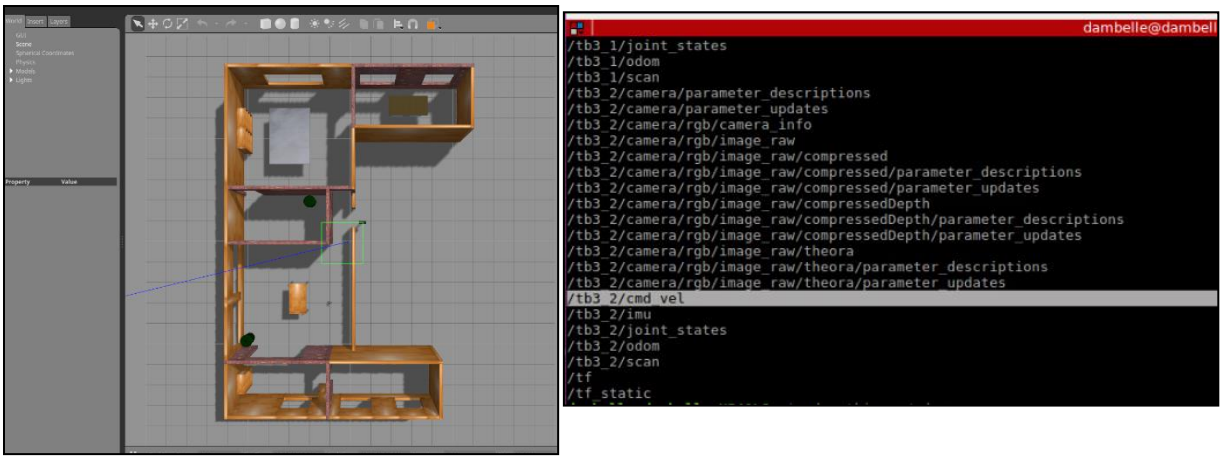

FIG. 3.2.3.1 – A gauche, la simulation sous Gazebo et à droite, quelques topics relatifs aux robots

Ces launch déjà créés facilitent grandement le travail et notamment l'apprentissage. En effet avec leurs aides, la compréhension de certaines notions/éléments a été grandement facilitée.

3.3 Aller vers l'objectif

Molière disait « le chemin est long du projet à la chose ». Cette citation décrit parfaitement notre situation. En effet dans la pratique et d'un œil extérieur, le projet peut paraître simple et rapide. Mais il nous a fallu de nombreuses heures de travail pour surmonter les petites embûches qui étaient mises sur notre route. Il fut complexe de réaliser tout ce qui nous était proposé.

3.3.1. Construction de la carte physique et SLAM.

Tout d'abord qu'est-ce que SLAM ? Ce sigle signifie en anglais Simultaneous Localization And Mapping. C'est un package qui permet au robot de construire la carte de son environnement et de s'y localiser. Plus précisément nous avons utilisé dans ce package l'algorithme Gmapping. La navigation ne peut être faite sans SLAM. Ils sont dépendants l'un de l'autre. Ainsi, le robot va construire une carte avec l'aide de son capteur LIDAR 360° 2D.

Pour effectuer le SLAM nous avons été obligés de piloter le robot avec l'aide de la télécommande car le SLAM est souvent considéré comme le paradoxe de la poule et de l'œuf : une carte est nécessaire pour définir la localisation, la localisation est nécessaire pour construire une carte. Nous l'avons ainsi aidé pour établir cette carte.

Pour le premier essai, il était question de construire une carte prototype (figure 3.3.1.1).

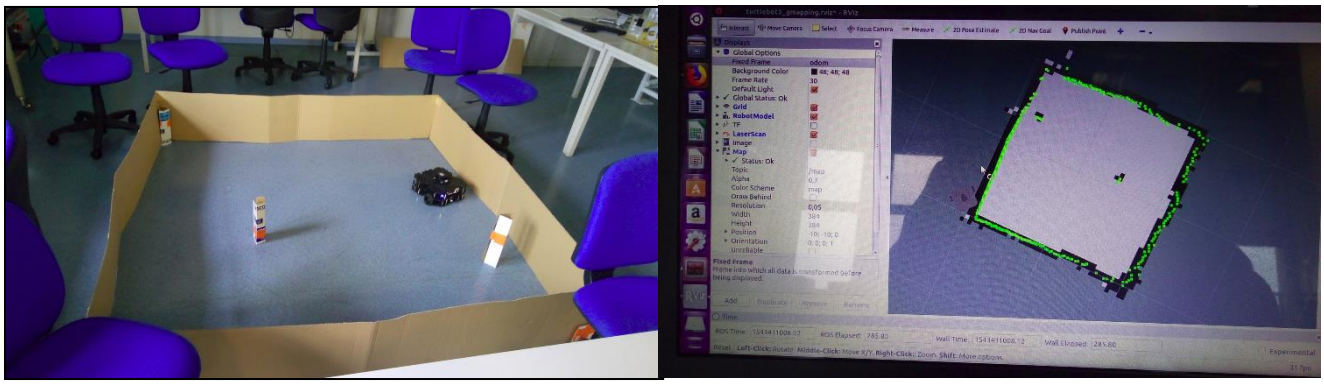

FIG. 3.3.1.1 – A gauche, le prototype de carte réel et à droite, le SLAM de cette carte

Il nous fallait une carte suffisamment discriminante qui permette au robot de ne pas se perdre par la suite. Mais qu'est-ce que cela signifie ? Si l'environnement présente de nombreuses surfaces régulières ces dernières peuvent sembler similaires pour le robot. Ainsi, le robot n'a pas suffisamment d'informations pour se localiser. Pour cela, nous avons ajouté des obstacles.

Cependant, la carte ci-dessus ne pouvait pas correspondre au standard que nous nous étions fixé. Nous souhaitions une carte qui soit répétable, c'est-à-dire une carte qui ait très peu de variabilités lors d'une « remise en place » au niveau de ces formes, surfaces et distances. La carte ci-dessus ne correspond pas ainsi pas à ces critères. On constate également sur la figure 3.3.1.1 que le robot est perdu dans la carte.

Nous avons ensuite effectué une deuxième carte un peu plus robuste et discriminante que la précédente (figure 3.3.1.2).

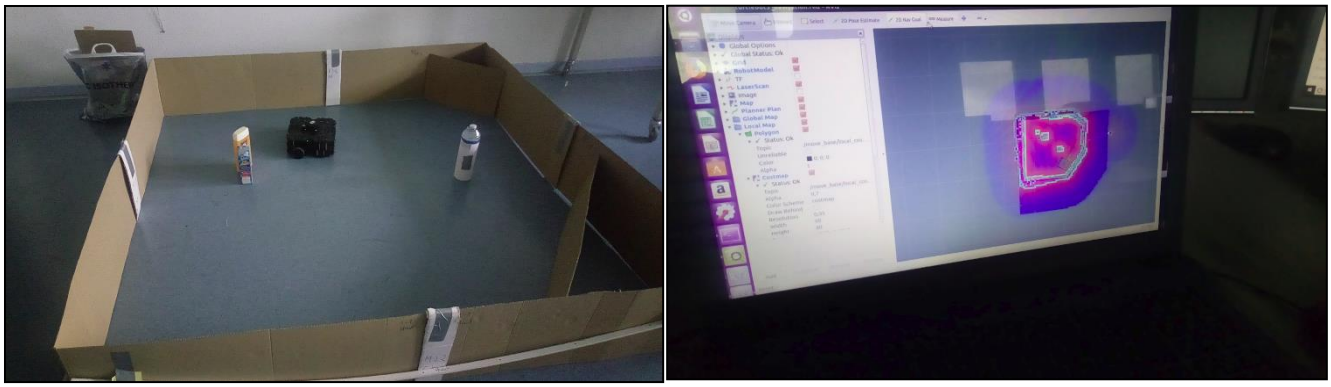

FIG. 3.3.1.2 – A gauche, la carte réelle V2 et à droite, le SLAM de cette carte

Nous n'étions toujours pas satisfaits de cette dernière car malgré les renforts apportés sur le côté la carte avait toujours tendance à bouger et en plus de cela nous nous sommes aperçus qu'elle était trop petite pour que le robot puisse bouger sans trop d'encombres.

Ainsi, l'avant-dernière carte, la V3 (figure 3.3.1.3) est née du fruit de l'expérience acquis.

FIG. 3.3.1.3 – La carte V3

Cependant, la carte n'a pas été retenue car nous avons rencontré de nombreuses problématiques liées à la modélisation de cette dernière. Après avoir mis en place physiquement les éléments relatifs à la carte (murs, obstacles, …) nous avons réalisé une cartographie de cette dernière. De là, nous avons pu obtenir une carte dont deux fichiers émanant de la cartographie, avec la commande rosrun map_server map_server map.yaml. Un fichier png utile notamment pour Rviz et un fichier yaml contenant les métadonnées à propos de la carte (données servants à définir ou décrire la carte dans notre cas).

Cette étape avait pour finalité de modéliser notre environnement en 3D sous Gazebo. Après de nombreux essais, nous nous sommes aperçus qu'il était très complexe de modéliser des éléments sous ce logiciel. Notamment pour les obstacles.

En effet, les seuls éléments qu'on puisse représenter sont des murs et le placement relatif par rapport à d'autres objets n'est pas très intuitif et précis.

Nous avons donc essayé de modéliser notre environnement par Catia. Gazebo peut travailler avec deux formats de fichiers : SDF et l'URDF. Pour information, le fichier URDF très utilisé dans ROS étant l'abréviation de Universal Robotic Description Format. C'est un fichier XML qui décrit les éléments de l'objet associés au fichier. Ces fichiers sont les plus utilisés sous ROS car ils sont plus faciles à utiliser avec les packages ROS.

Ainsi, l'idée était la suivante :

- Utiliser CATIA pour pouvoir obtenir un fichier de type STL (figure 3.3.1.4).

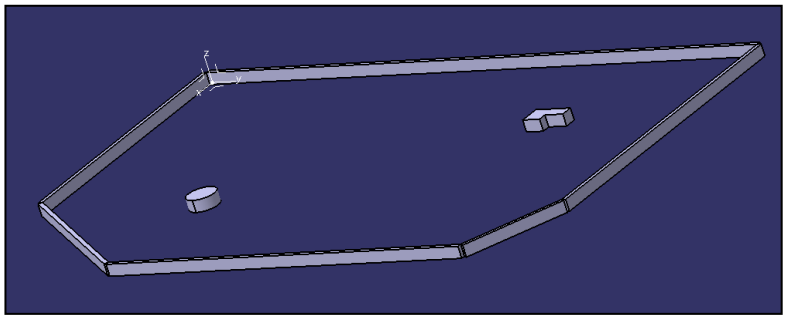

FIG. 3.3.1.4 – La carte V3 sous Catia

- A partir de ce fichier passait par Blender (figure 3.3.1.5) pour pouvoir éventuellement ajuster l'origine, la taille, ajouter des textures et matériaux et surtout pour pouvoir exporter le fichier au format DAE (collada).

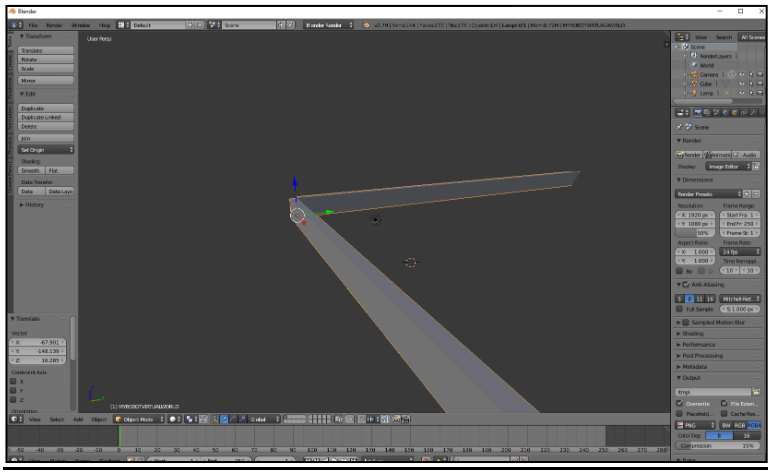

FIG. 3.3.1.5 – La carte V3 sous Blender

- Ensuite importer ce fichier sous Gazebo et l'enregistrer sous le format adéquate (XML extension .world) pour pouvoir créer un launch par la suite. Malencontreusement, cette dernière étape n'a pas fonctionné.

Par la suite, une autre carte à émaner et s'est avérée être la carte finale (figure 3.3.1.6).

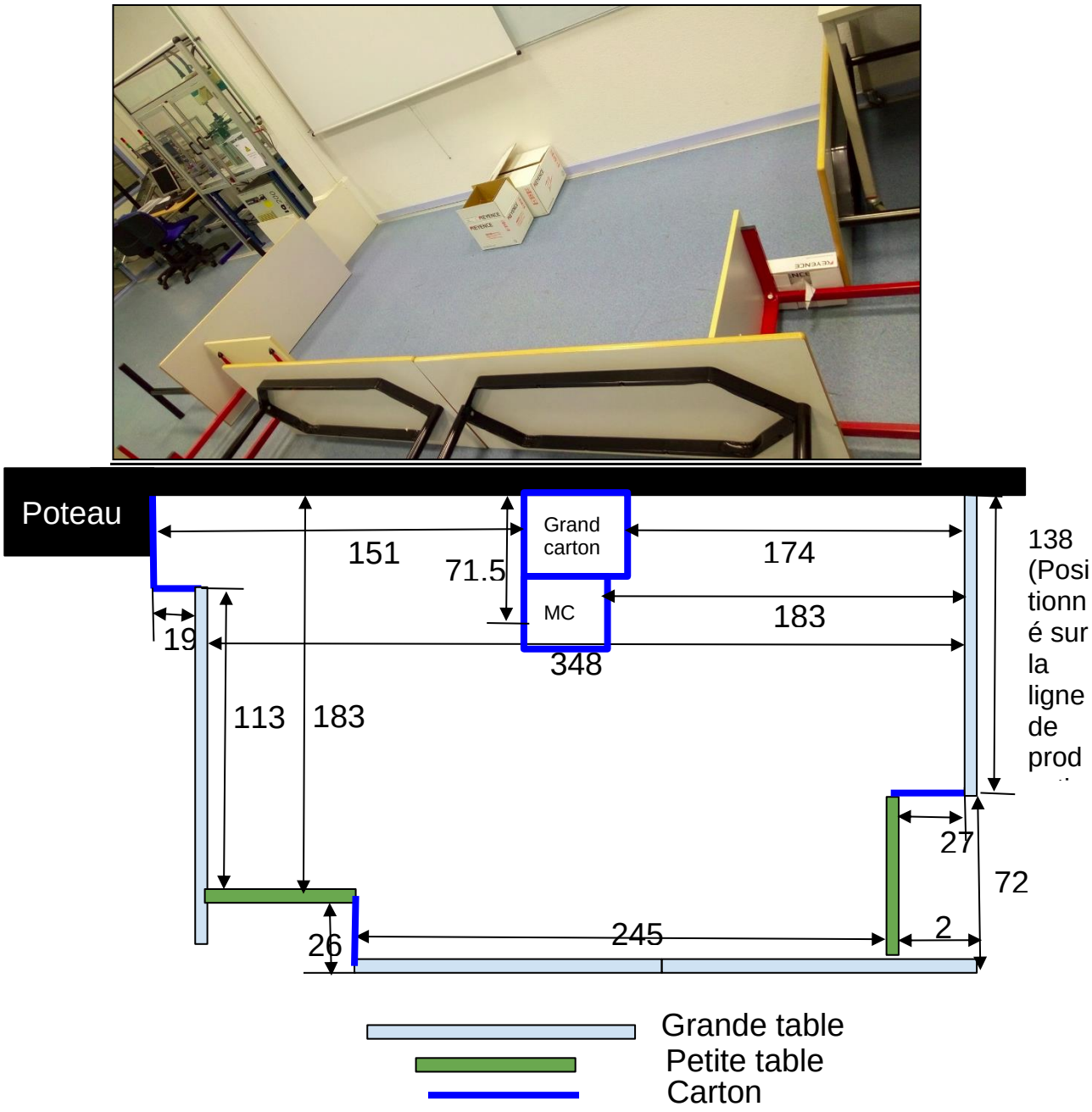

FIG. 3.3.1.6 – En haut, la carte V6 et en bas, le plan de la carte V6

De là, notre travail de SLAM et de modélisation a été effectué. Sur cette carte, nous avons effectué directement la modélisation sous Gazebo.

3.3.2. Modélisation de la carte finale et création des fichiers

Même si le processus de construction de carte a été amorcé dans la sous-partie précédente, il est important de pouvoir comprendre la méthode pour pouvoir modéliser son environnement avec les fichiers adéquats pour pouvoir le faire fonctionner dans un fichier de type launch par la suite. Ce fichier nous permettra de lancer notre robot dans son environnement, ce qui correspondra au robot virtualisé.

A partir de l'image extraite de la cartographie et du LIDAR (figure 3.3.2.1) nous avons pu directement travailler à partir du building editor de Gazebo.

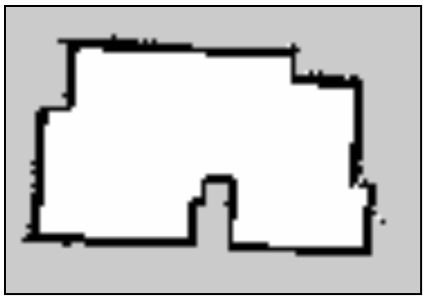

FIG. 3.3.2.1 – La map 2D extraite du LIDAR 2D.

A partir de ce modèle, le respect des dimensions réelles est respecté. En générant le fichier à partir de ce building editor un fichier xml est produit. Le model.sdf (fichier utilisé plus tard pour créer le world) et le model.config (qui est un fichier nécessaire à Gazebo pour trouver le modèle dans sa base de données). Vous les trouverez en annexe 2.

Par la suite la création du world ne consiste simplement, pour notre cas à nous, qu'a un simple enregistrement dans le bon format. Le fichier d'extension .world est créé, ainsi que notre carte virtuelle (figure 3.3.2.2).

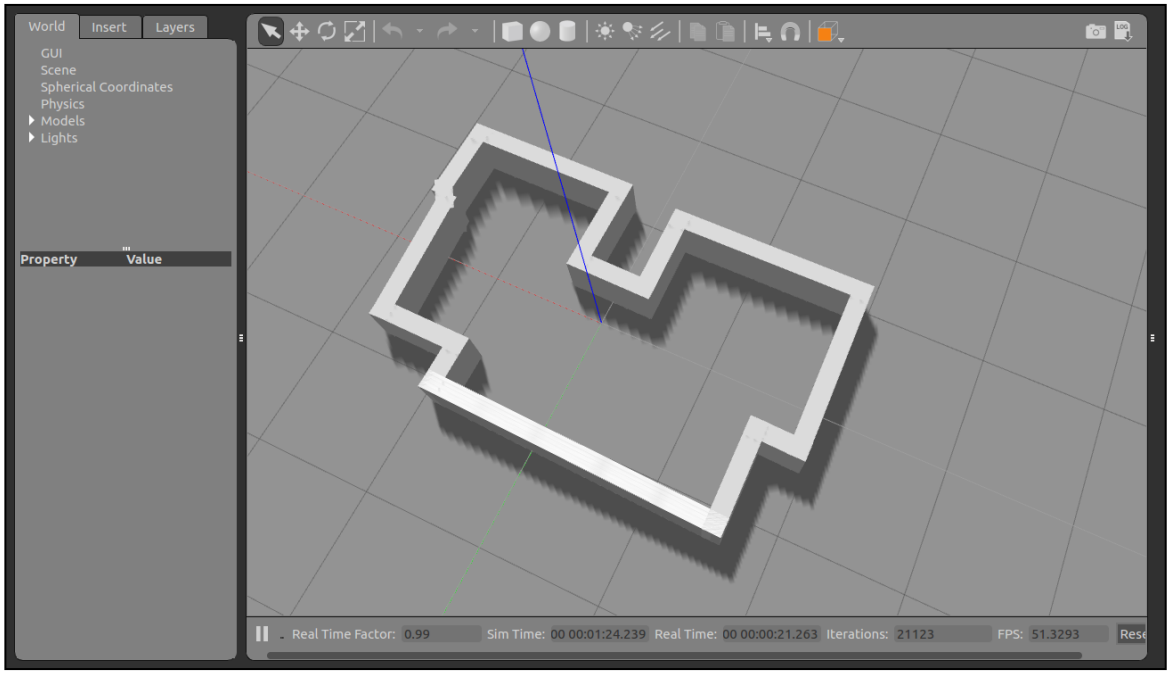

FIG. 3.3.2.2 – Carte virtuelle

Vous trouverez en Annexe 3 le fichier du world.

Le launch de la carte virtuelle a pu être établi. Avant tout, il faut savoir que de nombreux fichiers launch ont été créés pour pouvoir amener les nodes nécessaires du robot virtuel. Les launch sont des fichiers XML qui permettent d'automatiser le fait de pouvoir amener un ou plusieurs launch avec les paramètres initialisés de la manière spécifiée dans le programme.

Ouvrir Enregistrer

model.sdf × model.config × worldTB3TW53.launch × mapTB3TW53.world ×

```
<launch>
  <arg name="model" default="$(env TURTLEBOT3_MODEL)" doc="model type [burger, waffle, waffle_pi]"/>
  <arg name="virtual_tb3"  default="tb3_0"/>
  <arg name="virtual_tb3x_pos" default="0.0"/>
  <arg name="virtual_tb3y_pos" default="0.0"/>
  <arg name="virtual_tb3z_pos" default="0.0"/>

  <include file="$(find gazebo_ros)/launch/empty_world.launch">
    <arg name="world_name" value="$(find turtlebot3_gazebo)/worlds/mapTB3TW53.world"/>
    <arg name="paused" value="false"/>
    <arg name="use_sim_time" value="true"/>
    <arg name="gui" value="true"/>
    <arg name="headless" value="false"/>
    <arg name="debug" value="false"/>
  </include>

  <param name="virtual_robot_description" command="$(find xacro)/xacro --inorder $(find turtlebot3_description)/urdf/
turtlebot3_waffle_pi.urdf.xacro" />

  <!--node pkg="gazebo_ros" type="spawn_model" name="spawn_urdf" args="-urdf -model turtlebot3_$(arg model) -x $(arg x_pos) -y $(arg y_pos) -z
$(arg z_pos) -param robot_description" /-->

  <node pkg="gazebo_ros" type="spawn_model" name="spawn_urdf" args="-urdf -model $(arg virtual_tb3) -x $(arg virtual_tb3x_pos) -y $(arg
virtual_tb3y_pos) -z $(arg virtual_tb3z_pos) -param robot_description" />
</launch>
```

XML Largeur des tabulations : 8 Lig 22, Col 10 INS

FIG. 3.3.2.3 – Launch file du robot dans son environnement

La figure 3.3.2.3 est le premier launch qui permet d'emmener un robot virtuel dans son environnement virtuel. Plus tard, dans le rapport vous trouverez le launch file final qui permet de ramener le robot dans son environnement.

3.3.3. La navigation

La navigation permet au robot de bouger d'un endroit à un autre endroit spécifié dans un environnement spécifique. La navigation bouge le robot de la position actuelle à la position désigné sur la carte en utilisant ces capteurs (odometry, LIDAR …). Dans cette optique, une carte contenant les informations géométriques de l'environnement est nécessaire. Le SLAM a donc permis d'obtenir la carte par les informations reçues du LIDAR.

La première navigation effectuée a été faite dans la carte réelle après avoir fait le SLAM.

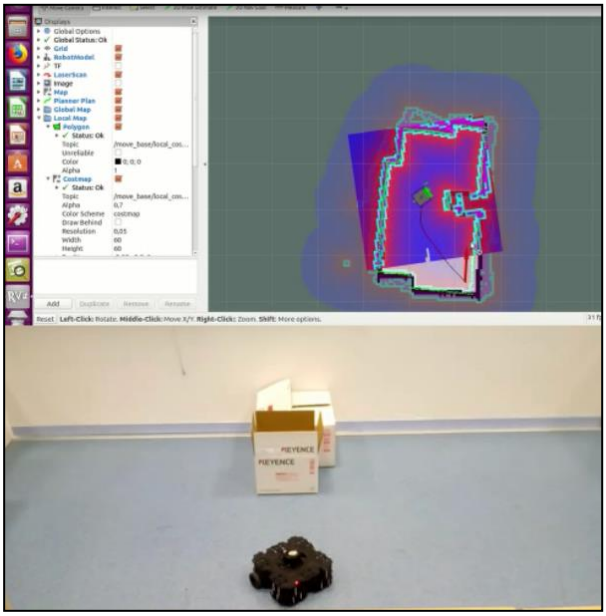

FIG. 3.3.3.1 – Navigation robot réel dans la carte finale V4

Cela nous a permis de nous poser des questions importantes comme le fonctionnement du package navigation.

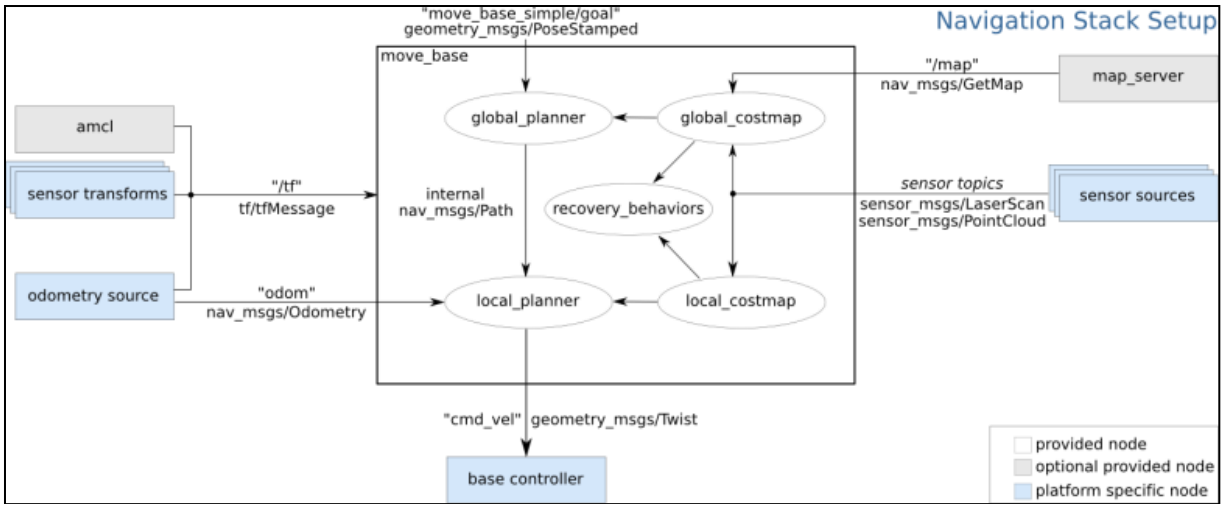

FIG. 3.3.3.2 – Paramètres du package Navigation
source : http://wiki.ros.org/navigation/Tutorials/RobotSetup

La figure 3.3.3.2 nous montre les différents paramètres se référant au Navigation Stack et également ceux sur lesquels nous devons intervenir afin de l'adapter à notre robot. Plus tard, dans le rapport nous constaterons que de nombreux paramètres ont dû être modifiés afin de pouvoir s'adapter à notre projet dû au fait que nous avons plusieurs robots. Cependant pour notre cas ici présent, nous avions juste à lancer le package pour comprendre comment il s'articulait.

Dans ce même package nous pouvons également ajuster des paramètres tels que :

- L'inflation radius. C'est un paramètre qui permet de modifier la zone de tolérance autours des obstacles. Le robot ne peut donc pas rentrer dans ces dernières (figure 3.3.3.3).

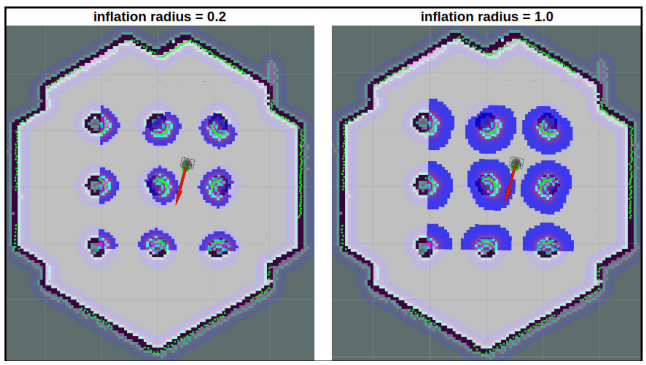

FIG. 3.3.3.3 – Comparaison de valeurs de l'inflation radius
source : http://emanual.robotis.com

- Et d'autres paramètres tels que la vitesse linéaire maximum/minimum, la vitesse maximum de rotation, l'accélération, etc.

Nous nous sommes simplement contentés de modifier la valeur de l'inflation radius pour pouvoir le voir sur le robot réel. Nous avons ensuite effectué la navigation dans notre carte virtuelle. Celle-ci fonctionne très bien.

Nous avons essayé par la suite de récupérer la position du robot dans le référentiel du world. Pour cela nous avons récupéré le package de M. Zhi Yan (https://github.com/yzrobot/pose_publisher) qui donne la position et l'orientation du robot dans le référentiel de la carte. Il nous a simplement fallu lancer le programme et faire un écho du topic sur lequel le programme publie. Ce programme fonctionne sur le principe de transformées, les TF.

Un robot a plusieurs transformées entre ses différents repères. Ce sont des modèles géométriques simples, sous forme de matrices. Certaines de ces transformations sont données directement, c'est-à-dire que la transformée entre les deux repères existe. Mais il arrive assez souvent que le robot doive effectuer des transformées inverses.

Dans notre cas, le robot doit connaître sa position par rapport à la carte. Rien physiquement ne le relie directement à cette dernière. Le package amcl qui est un algorithme de localisation probabiliste permet d'utiliser les filtres particulaires (particle filter) afin de connaître la position d'un des repères du robot dans la carte (figure 3.3.3.4).

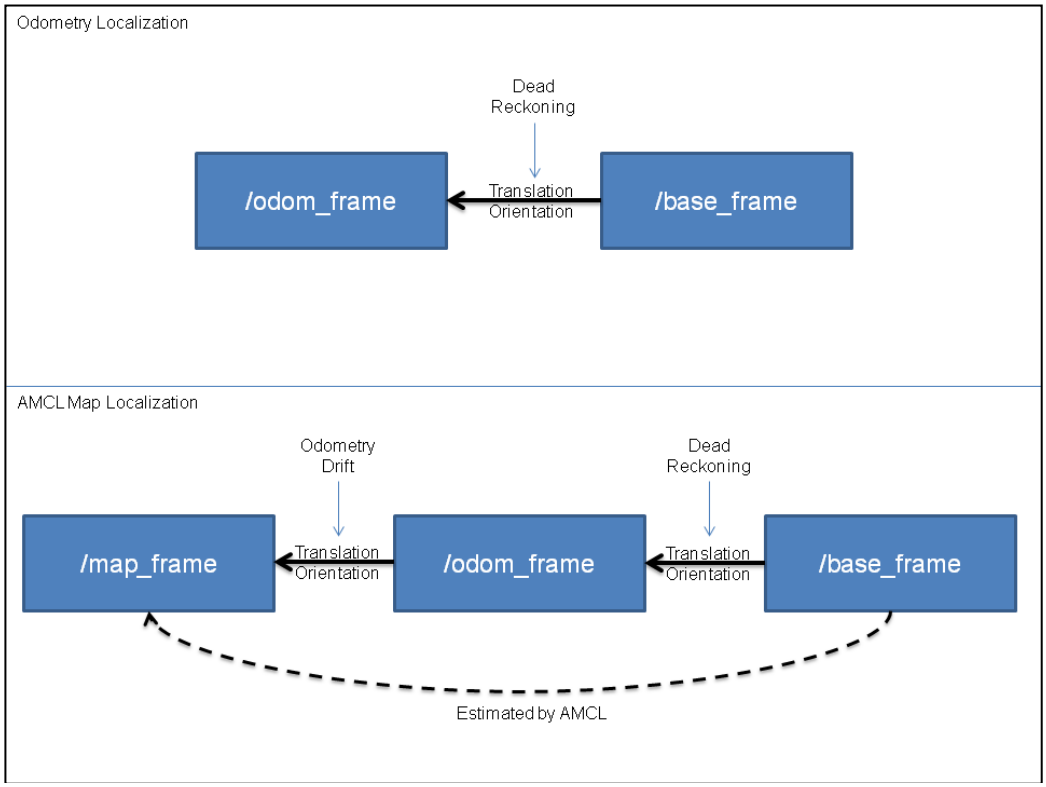

FIG. 3.3.3.4 – Utilité de l'algorithme AMCL
source : http://wiki.ros.org/amcl#Transforms

Le package de M. Zhi Yan nous facilite grandement la tâche car il permet d'obtenir directement la transformée de /mapframe au /posepublisher (qui est un node

qui publie la position relative du robot) en remontant notamment l'ensemble du TF tree (figure 3.3.3.5) avec des transformées inverses.

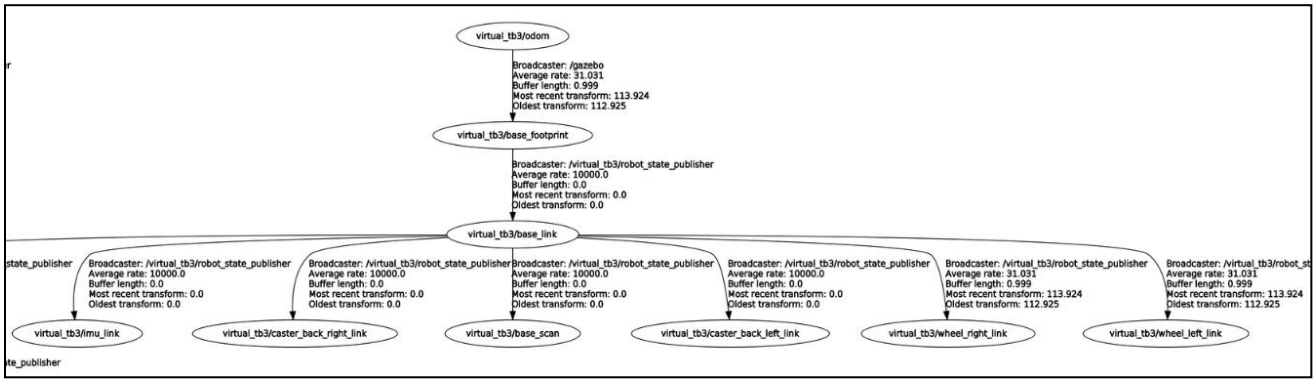

FIG. 3.3.3.5 – Partie du TF tree du robot virtuel

Rqt permettant, en autre, de visualiser le TF tree fut un outil sur lequel nous nous sommes beaucoup appuyés pour notre travail car très utile pour effectuer des débogages.

Toujours grâce au package de navigation, nous avons pu lancer un programme Python (Annexe 5) qui dit au robot d'aller à un endroit en particulier de la carte. Ces informations récupérables par le biais de Rviz peuvent être insérées ensuite dans le programme. Ce nœud fonctionne très bien et peut-être utiliser assez facilement.

3.3.4. Le launch pour le virtual robot

Un launch particulier a dû être créé pour le robot virtuel (Annexe 6). Le principe de Rviz consiste à faire correspondre les bons paramètres et topic au capteur qu'on souhaite visualiser. Cependant si ces mêmes paramètres et topic portent les mêmes noms et reçoivent des informations différentes de deux robots cela crée des conflits.

Cette étape est la plus ardue et elle n'est toujours pas résolue à l'heure actuelle. Ce launch comporte une balise <ns group> qui permet de différencier le robot réel du virtuel (en théorie). Cela évite de retoucher la multitude de fichiers liés à un robot en changeant leurs appellations.

Notre launch doit donc emmener le robot virtuel dans son environnement virtuel mais aussi les paramètres du package navigation. Nous devions l'effectuer dû au fait de la balise. Des nodes tels que l'amcl, le move base, etc doivent pouvoir être différencier du robot réel qui aura ses propres nodes sous sa propre appellation.

Cependant nous avons eu quelques problèmes :

- Le robot virtuel, avec le launch de l'annexe 6, à quelques problèmes de localisation. Il y a un écart important entre la map virtuelle (déjà effectué) et les données reçues du capteur (figure 3.3.4.1). Les AMCL particles (les flèches vertes foncés) ne convergent pas. Après de nombreuses recherches nous n'avons pas trouvé la source du

problème. Il y a beaucoup de « solutions » sur les forums qui ne marchent pas pour nous. Un autre problème est que les particle cloud (point vert clairs) devraient correspondre au fond de carte (contours noirs), et l'écart reste relativement important. C'est un autre problème que nous n'avons pas réussi à résoudre.

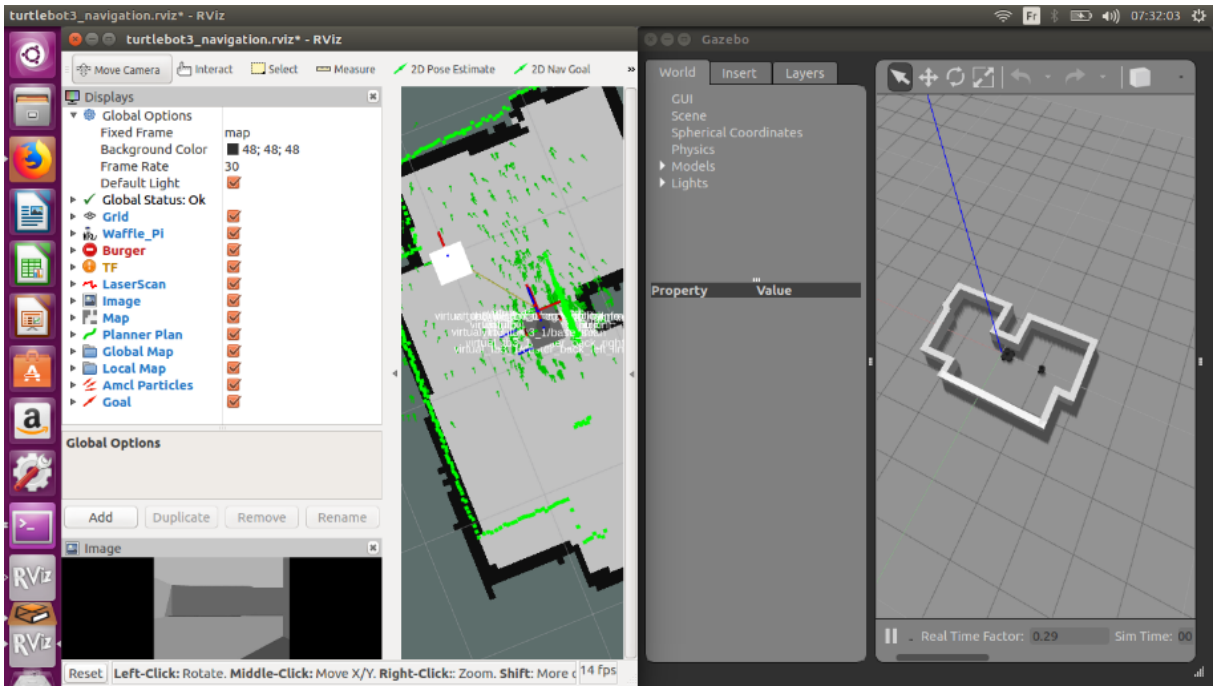

FIG. 3.3.3.6 – Un des problèmes du launch virtual robot

- L'autre problème est que pour faire fonctionner le programme qui spécifie d'aller à un point particulier de la carte ce dernier nécessite que le package navigation (et notamment le move base) soit lancé. Mais le package navigation ne fonctionne pas dû à la balise <ns group> du launch virtual bringup qui change l'appellation de notre robot. La navigation n'arrive plus à reconnaitre notre robot. L'idée aurait été de créer un package navigation spécifique à notre robot mais cela remonte à de l'impossible en si peu de temps. De nombreux fichiers, messages, nodes, topics sont dépendants du package navigation. Le but de ROS étant également de ne plus avoir à faire ce genre de manipulations. Comme le précédent problème il reste non-résolu.

3.3.5. Gérer le croisement des robots

Notre objectif, ici, est d'éviter toute « collision » entre le robot réel et le robot virtuel. Pour cela, nous avons créé un programme (cf annexe 7) qui arrête le robot réel lorsque celui-ci est dans un rayon inférieur à 0,3m de l'autre robot.

La première étape du programme est donc d'obtenir les positions en x et y des robots. Pour cela, des topics existants nous permettent d'obtenir leur position respective. Nous souscrivons au topic Odometry de chacun des robots.

```
odom_sub = rospy.Subscriber('/tb3_0/odom', Odometry, callback)
```

Nous récupérons les valeurs grâce à la commande msg.pose.pose.position.x et nous les stockons dans des variables globales (x_r, y_r, x_v et y_v).

Nous créons ensuite une fonction arret qui a pour but de publier dans le topic /cmd_vel du robot réel. Nous publions dans ce topic à la seule condition que la distance entre les deux robots est inférieure à 0,3m.

def arret():

```
    pub = rospy.Publisher('/tb3_0/cmd_vel', Twist, queue_size=10)
    vel_msg = Twist()

    while not rospy.is_shutdown() :
        if sqrt ((x_r-x_v)**2 + (y_r-y_v)**2) < 10 : #remettre 0.3 (10 pour
les tests)
            #print sqrt ((x_r-x_v)**2 + (y_r-y_v)**2) #test pour voir si on
entre dans le if -> OK
            vel_msg.linear.x = 0
            vel_msg.linear.y = 0
            vel_msg.linear.z = 0
            vel_msg.angular.x = 0
            vel_msg.angular.y = 0
            vel_msg.angular.z = 0
            #print vel_msg #test pour voir si les valeurs dans le twist
sont bien toutes a 0 -> OK
            pub.publish(vel_msg) # Probleme sur cette ligne. Ne publie pas
            rate.sleep()
```

En testant ce programme, nous rencontrons encore des problèmes. Il semblerait que le nœud ne publie pas dans le topic concerné. Nous avons réalisé des tests pour savoir d'où pourrait provenir le problème.

Nous avons commencé par vérifier que les valeurs des positions sont bien récupérées. Ensuite, nous avons vérifié que lorsque la condition est vraie, nous rentrons bien dans la boucle if. La dernière vérification a été de savoir si toutes les valeurs de la variable « vel_msg » étaient bien à 0. Ces vérifications sont revenues positives : le problème ne vient pas de là.

Nous avons également essayé de changer la structure du programme : ne pas mettre la condition dans une fonction, …

Comme le robot ne s'arrête pas lorsqu'il est dans un rayon inférieur à la limite, nous pensons que l'utilisation du « .publish » a mal été gérée.

3.3.6 Faire cohabiter le robot réel et virtuel

Malgré nos petits imprévus du launch bringup virtuel nous arrivons à recevoir les données de nos deux robots (réel et virtuel) sous Rviz. Nous n'arrivons pas à les faire cohabiter simultanément mais nous pouvoir voir l'ensemble de leurs données malgré tout en choisissant l'un puis l'autre.

Si vous souhaiter faire cela il vous faudra :

- Robot réel : Se connecter au robot, lancer le launch turtlebot3_robot. launch du package turtlebot3_bringup. Attendre la fin de la calibration des capteurs.

- Robot virtuel : Lancer son virtual bringup.launch (annexe 6). Lorsque Rviz sera lancer, il faudra ajouter les capteurs que vous souhaitez voir et les associez au topic/parameter du robot souhaité.

Cette procédure permet de voir les robots, par la suite si vous voulez lancer des nodes tels que le programme aller à un point particulier de la carte, faire s’arrêter le robot, etc vous pouvez les lancer indépendamment ou bien créer un launch qui lance l’ensemble des nodes que vous souhaitez.

Conclusion

Dans ce rapport, nous avons expliqué notre démarche concernant ce projet qui consiste en contrôler des robots (réels et virtuels) sous le middleware ROS et de créer un espace collaboratif où les deux robots évolueront.

Notre travail s’est déroulé en trois étapes. La première a été une étape d’apprentissage, la seconde a été de définir l’espace de travail et de commencer à programmer la trajectoire des robots. Enfin, nous nous sommes concentrés sur la partie espace collaboratif.

Nous n’avons malheureusement pas atteint l’objectif du projet. Notre réflexion est à continuer. Nous pensons que nous sommes assez proche de l’objectif fixé et qu’il nous a juste manqué un peu de temps.

Ce projet a été très instructif pour nous. Il nous a permis d’avoir un aperçu sur le monde de la robotique industrielle mobile que nous avons peu abordé en cours. Ce fut très enrichissant car nous avons appris à nous servir d’un système d’exploitation et de ROS que nous ne connaissions pas.

Ce projet se situe dans le cadre d’un projet de dernier semestre d’étude à l’UTBM. En effet, nous avons pu mettre en application certaines notions de SY55 et de MC55. Il nous a placé dans une démarche d’ingénieur : travail en équipe, gestion du temps (livrables, revue de projet).

Comme beaucoup de projet, le nôtre a été enrichissant du point de vue humain. Nous avons collaboré au sein de notre groupe (même s’il y a eu certains problèmes) et nous nous sommes répartis les tâches. Cette répartition a permis un partage de connaissance au sein du groupe.

Bibliographie

http://wiki.ros.org/

http://www.ee.surrey.ac.uk/Teaching/Unix/

https://github.com/

https://doc.ubuntu-fr.org/xenial

https://answers.ros.org/

http://emanual.robotis.com/docs/en/platform/turtlebot3/overview/

http://www.theconstructsim.com/

Book : ROS_Robot_Programming_English

Internet en général

Annexes

Annexe 1: Programme mouvement Linéaire

```
#!/usr/bin/env python
import rospy
from geometry_msgs.msg import Twist
import time

def move():

    while not rospy.is_shutdown():

        # Starts a new node
        rospy.init_node('robot_cleaner', anonymous=True)
        velocity_publisher = rospy.Publisher('tb3_2/cmd_vel', Twist, queue_size=10)


        #velocity_publisher = rospy.Publisher('/tb3_1/cmd_vel', Twist,
queue_size=10)
        #velocity_publisher = rospy.Publisher('/tb3_3/cmd_vel', Twist,
queue_size=10)
        vel_msg = Twist()

        #Demande de depart

        input("Repondre 1 lorsque le chargement est termine : ")

        #PARTIR
        speed = 1
        distance = 100
        isForward = 0 #True or False

        #Checking if the movement is forward or backwards
        if(isForward):
            vel_msg.linear.x = abs(speed)
        else:
            vel_msg.linear.x = -abs(speed)
        #Since we are moving just in x-axis
        vel_msg.linear.y = 0
        vel_msg.linear.z = 0
        vel_msg.angular.x = 0
        vel_msg.angular.y = 0
        vel_msg.angular.z = 0


        #Setting the current time for distance calculus
        t0 = rospy.Time.now().to_sec()
        current_distance = 0
```

```
  #Loop to move the turtle in an specified distance
  while(current_distance < distance):
      #Publish the velocity
      velocity_publisher.publish(vel_msg)
      #Takes actual time to velocity calculus
      t1=rospy.Time.now().to_sec()
      #Calculates distancePoseStamped
      current_distance= speed*(t1-t0)
  #After the loop, stops the robot
  vel_msg.linear.x = 0
  #Force the robot to stop
  velocity_publisher.publish(vel_msg)



  input("Repondre 1 lorsque le dechargement est termine : ")


  #REVENIR
  speed = 1
  distance = 400
  isForward = 1 #True or False

  #Checking if the movement is forward or backwards
  if(isForward):
      vel_msg.linear.x = abs(speed)
  else:
      vel_msg.linear.x = -abs(speed)
  #Since we are moving just in x-axis
  vel_msg.linear.y = 0
  vel_msg.linear.z = 0
  vel_msg.angular.x = 0
  vel_msg.angular.y = 0
vel_msg.angular.z = 0


  #Setting the current time for distance calculus
  t0 = rospy.Time.now().to_sec()
  current_distance = 0

  #Loop to move the turtle in an specified distance
  while(current_distance < distance):
      #Publish the velocity
      velocity_publisher.publish(vel_msg)
      #Takes actual time to velocity calculus
      t1=rospy.Time.now().to_sec()
      #Calculates distancePoseStamped
      current_distance= speed*(t1-t0)
  #After the loop, stops the robot
```

```
        vel_msg.linear.x = 0
        #Force the robot to stop
        velocity_publisher.publish(vel_msg)

if __name__ == '__main__':
    try:
        #Testing our function
        move()
    except rospy.ROSInterruptException:
        pass
```

Annexe 2 (1) : Modèle du monde virtuel (sdf)

```
<?xml version='1.0'?>
<sdf version='1.6'>
  <model name='worldTB3TW53'>
    <pose frame=''>-0.09725 0.063678 0 0 -0 0</pose>
    <link name='Wall_10'>
      <collision name='Wall_10_Collision'>
        <geometry>
          <box>
            <size>0.5 0.15 0.4</size>
          </box>
        </geometry>
        <pose frame=''>0 0 0.2 0 -0 0</pose>
      </collision>
      <visual name='Wall_10_Visual'>
        <pose frame=''>0 0 0.2 0 -0 0</pose>
        <geometry>
          <box>
            <size>0.5 0.15 0.4</size>
          </box>
        </geometry>
        <material>
          <script>
            <uri>file://media/materials/scripts/gazebo.material</uri>
            <name>Gazebo/Grey</name>
          </script>
          <ambient>1 1 1 1</ambient>
        </material>
      </visual>
      <pose frame=''>-1.62175 0.344822 0 0 -0 3.1216</pose>
    </link>
    <link name='Wall_11'>
      <collision name='Wall_11_Collision'>
        <geometry>
          <box>
            <size>1.54539 0.15 0.4</size>
          </box>
        </geometry>
        <pose frame=''>0 0 0.2 0 -0 0</pose>
      </collision>
      <visual name='Wall_11_Visual'>
        <pose frame=''>0 0 0.2 0 -0 0</pose>
        <geometry>
          <box>
            <size>1.54539 0.15 0.4</size>
          </box>
        </geometry>
        <material>
          <script>
            <uri>file://media/materials/scripts/gazebo.material</uri>
            <name>Gazebo/Grey</name>
          </script>
          <ambient>1 1 1 1</ambient>
        </material>
      </visual>
      <pose frame=''>-1.78875 -0.349178 0 0 0 -1.55933</pose>
    </link>
    <link name='Wall_12'>
      <collision name='Wall_12_Collision'>
        <geometry>
          <box>
            <size>1.75045 0.15 0.4</size>
          </box>
        </geometry>
        <pose frame=''>0 0 0.2 0 -0 0</pose>
      </collision>
      <visual name='Wall_12_Visual'>
        <pose frame=''>0 0 0.2 0 -0 0</pose>
        <geometry>
          <box>
            <size>1.75045 0.15 0.4</size>
          </box>
        </geometry>
```

```
      <material>
        <script>
          <uri>file://media/materials/scripts/gazebo.material</uri>
          <name>Gazebo/Grey</name>
        </script>
        <ambient>1 1 1 1</ambient>
      </material>
    </visual>
    <pose frame=''>-0.98075 -1.04318 0 0 -0 0.004375</pose>
  </link>
  <link name='Wall_13'>
    <collision name='Wall_13_Collision'>
      <geometry>
        <box>
          <size>0.840203 0.15 0.4</size>
        </box>
      </geometry>
      <pose frame=''>0 0 0.2 0 -0 0</pose>
    </collision>
    <visual name='Wall_13_Visual'>
      <pose frame=''>0 0 0.2 0 -0 0</pose>
      <geometry>
        <box>
          <size>0.840203 0.15 0.4</size>
        </box>
      </geometry>
      <material>
        <script>
          <uri>file://media/materials/scripts/gazebo.material</uri>
          <name>Gazebo/Grey</name>
        </script>
        <ambient>1 1 1 1</ambient>
      </material>
    </visual>
    <pose frame=''>-0.18475 -0.694178 0 0 -0 1.58237</pose>
  </link>
  <link name='Wall_14'>
    <collision name='Wall_14_Collision'>
      <geometry>
        <box>
          <size>0.743054 0.15 0.4</size>
        </box>
      </geometry>
      <pose frame=''>0 0 0.2 0 -0 0</pose>
    </collision>
    <visual name='Wall_14_Visual'>
      <pose frame=''>0 0 0.2 0 -0 0</pose>
      <geometry>
        <box>
          <size>0.743054 0.15 0.4</size>
        </box>
      </geometry>
      <material>
        <script>
          <uri>file://media/materials/scripts/gazebo.material</uri>
          <name>Gazebo/Grey</name>
        </script>
        <ambient>1 1 1 1</ambient>
      </material>
    </visual>
    <pose frame=''>0.10775 -0.352678 0 0 0 -0.01349</pose>
  </link>
  <link name='Wall_15'>
    <collision name='Wall_15_Collision'>
      <geometry>
        <box>
          <size>0.894043 0.15 0.4</size>
        </box>
      </geometry>
      <pose frame=''>0 0 0.2 0 -0 0</pose>
    </collision>
    <visual name='Wall_15_Visual'>
      <pose frame=''>0 0 0.2 0 -0 0</pose>
      <geometry>
        <box>
          <size>0.894043 0.15 0.4</size>
```

```
      </box>
     </geometry>
     <material>
      <script>
       <uri>file://media/materials/scripts/gazebo.material</uri>
       <name>Gazebo/Grey</name>
      </script>
      <ambient>1 1 1 1</ambient>
     </material>
    </visual>
    <pose frame=''>0.40025 -0.728678 0 0 0 -1.58155</pose>
   </link>
   <link name='Wall_16'>
    <collision name='Wall_16_Collision'>
     <geometry>
      <box>
       <size>1.52311 0.15 0.4</size>
      </box>
     </geometry>
     <pose frame=''>0 0 0.2 0 -0 0</pose>
    </collision>
    <visual name='Wall_16_Visual'>
     <pose frame=''>0 0 0.2 0 -0 0</pose>
     <geometry>
      <box>
       <size>1.52311 0.15 0.4</size>
      </box>
     </geometry>
     <material>
      <script>
       <uri>file://media/materials/scripts/gazebo.material</uri>
       <name>Gazebo/Grey</name>
      </script>
      <ambient>1 1 1 1</ambient>
     </material>
    </visual>
    <pose frame=''>1.08275 -1.10068 0 0 -0 0</pose>
   </link>
   <link name='Wall_19'>
    <collision name='Wall_19_Collision'>
     <geometry>
      <box>
       <size>1.20032 0.15 0.4</size>
      </box>
     </geometry>
     <pose frame=''>0 0 0.2 0 -0 0</pose>
    </collision>
    <visual name='Wall_19_Visual'>
     <pose frame=''>0 0 0.2 0 -0 0</pose>
     <geometry>
      <box>
       <size>1.20032 0.15 0.4</size>
      </box>
     </geometry>
     <material>
      <script>
       <uri>file://media/materials/scripts/gazebo.material</uri>
       <name>Gazebo/Grey</name>
      </script>
      <ambient>1 1 1 1</ambient>
     </material>
    </visual>
    <pose frame=''>1.64875 0.149322 0 0 -0 1.63844</pose>
   </link>
   <link name='Wall_21'>
    <collision name='Wall_21_Collision'>
     <geometry>
      <box>
       <size>0.75 0.15 0.4</size>
      </box>
     </geometry>
     <pose frame=''>0 0 0.2 0 -0 0</pose>
    </collision>
    <visual name='Wall_21_Visual'>
     <pose frame=''>0 0 0.2 0 -0 0</pose>
     <geometry>
```

```
      <box>
        <size>0.75 0.15 0.4</size>
      </box>
    </geometry>
    <material>
      <script>
        <uri>file://media/materials/scripts/gazebo.material</uri>
        <name>Gazebo/Grey</name>
      </script>
      <ambient>1 1 1 1</ambient>
    </material>
  </visual>
  <pose frame=''>1.76925 -0.800678 0 0 -0 1.5708</pose>
</link>
<link name='Wall_22'>
  <collision name='Wall_22_Collision'>
    <geometry>
      <box>
        <size>0.301384 0.15 0.4</size>
      </box>
    </geometry>
    <pose frame=''>0 0 0.2 0 -0 0</pose>
  </collision>
  <visual name='Wall_22_Visual'>
    <pose frame=''>0 0 0.2 0 -0 0</pose>
    <geometry>
      <box>
        <size>0.301384 0.15 0.4</size>
      </box>
    </geometry>
    <material>
      <script>
        <uri>file://media/materials/scripts/gazebo.material</uri>
        <name>Gazebo/Grey</name>
      </script>
      <ambient>1 1 1 1</ambient>
    </material>
  </visual>
  <pose frame=''>1.72675 -0.437678 0 0 -0 2.16427</pose>
</link>
<link name='Wall_4'>
  <collision name='Wall_4_Collision'>
    <geometry>
      <box>
        <size>0.5 0.15 0.4</size>
      </box>
    </geometry>
    <pose frame=''>0 0 0.2 0 -0 0</pose>
  </collision>
  <visual name='Wall_4_Visual'>
    <pose frame=''>0 0 0.2 0 -0 0</pose>
    <geometry>
      <box>
        <size>0.5 0.15 0.4</size>
      </box>
    </geometry>
    <material>
      <script>
        <uri>file://media/materials/scripts/gazebo.material</uri>
        <name>Gazebo/Grey</name>
      </script>
      <ambient>1 1 1 1</ambient>
    </material>
  </visual>
  <pose frame=''>0.93525 0.854322 0 0 0 -1.5708</pose>
</link>
<link name='Wall_5'>
  <collision name='Wall_5_Collision'>
    <geometry>
      <box>
        <size>0.827905 0.15 0.4</size>
      </box>
    </geometry>
    <pose frame=''>0 0 0.2 0 -0 0</pose>
  </collision>
  <visual name='Wall_5_Visual'>
```

```
      <pose frame="">0 0 0.2 0 -0 0</pose>
      <geometry>
        <box>
          <size>0.827905 0.15 0.4</size>
        </box>
      </geometry>
      <material>
        <script>
          <uri>file://media/materials/scripts/gazebo.material</uri>
          <name>Gazebo/Grey</name>
        </script>
        <ambient>1 1 1 1</ambient>
      </material>
    </visual>
    <pose frame="">1.27425 0.676322 0 0 0 -0.008849</pose>
  </link>
  <link name='Wall_7'>
    <collision name='Wall_7_Collision'>
      <geometry>
        <box>
          <size>2.53961 0.15 0.4</size>
        </box>
      </geometry>
      <pose frame="">0 0 0.2 0 -0 0</pose>
    </collision>
    <visual name='Wall_7_Visual'>
      <pose frame="">0 0 0.2 0 -0 0</pose>
      <geometry>
        <box>
          <size>2.53961 0.15 0.4</size>
        </box>
      </geometry>
      <material>
        <script>
          <uri>file://media/materials/scripts/gazebo.material</uri>
          <name>Gazebo/CeilingTiled</name>
        </script>
        <ambient>1 1 1 1</ambient>
      </material>
    </visual>
    <pose frame="">-0.25925 1.06432 0 0 0 -0.029293</pose>
  </link>
  <link name='Wall_9'>
    <collision name='Wall_9_Collision'>
      <geometry>
        <box>
          <size>0.908032 0.15 0.4</size>
        </box>
      </geometry>
      <pose frame="">0 0 0.2 0 -0 0</pose>
    </collision>
    <visual name='Wall_9_Visual'>
      <pose frame="">0 0 0.2 0 -0 0</pose>
      <geometry>
        <box>
          <size>0.908032 0.15 0.4</size>
        </box>
      </geometry>
      <material>
        <script>
          <uri>file://media/materials/scripts/gazebo.material</uri>
          <name>Gazebo/Grey</name>
        </script>
        <ambient>1 1 1 1</ambient>
      </material>
    </visual>
    <pose frame="">-1.45025 0.720322 0 0 0 -1.56156</pose>
  </link>
  <static>1</static>
 </model>
</sdf>
```

Annexe 2 (2) : Modèle du monde virtuel (.config)

```
<?xml version="1.0" ?>
<model>
   <name>worldTB3TW53</name>
   <version>1.0</version>
   <sdf version="1.6">model.sdf</sdf>
   <author>
      <name></name>
      <email></email>
   </author>
   <description></description>
</model>
```

Annexe 3 : World du monde virtuel

```
<sdf version='1.6'>
  <world name='default'>
    <light name='sun' type='directional'>
      <cast_shadows>1</cast_shadows>
      <pose frame=''>0 0 10 0 -0 0</pose>
      <diffuse>0.8 0.8 0.8 1</diffuse>
      <specular>0.1 0.1 0.1 1</specular>
      <attenuation>
        <range>1000</range>
        <constant>0.9</constant>
        <linear>0.01</linear>
        <quadratic>0.001</quadratic>
      </attenuation>
      <direction>-0.5 0.5 -1</direction>
    </light>
    <model name='ground_plane'>
      <static>1</static>
      <link name='link'>
        <collision name='collision'>
          <geometry>
            <plane>
              <normal>0 0 1</normal>
              <size>100 100</size>
            </plane>
          </geometry>
          <surface>
            <friction>
              <ode>
                <mu>100</mu>
                <mu2>50</mu2>
              </ode>
              <torsional>
                <ode/>
              </torsional>
            </friction>
            <contact>
              <ode/>
            </contact>
            <bounce/>
          </surface>
          <max_contacts>10</max_contacts>
        </collision>
        <visual name='visual'>
          <cast_shadows>0</cast_shadows>
          <geometry>
            <plane>
              <normal>0 0 1</normal>
              <size>100 100</size>
            </plane>
          </geometry>
          <material>
            <script>
              <uri>file://media/materials/scripts/gazebo.material</uri>
              <name>Gazebo/Grey</name>
            </script>
          </material>
        </visual>
        <self_collide>0</self_collide>
        <kinematic>0</kinematic>
        <gravity>1</gravity>
      </link>
    </model>
    <gravity>0 0 -9.8</gravity>
    <magnetic_field>6e-06 2.3e-05 -4.2e-05</magnetic_field>
    <atmosphere type='adiabatic'/>
    <physics name='default_physics' default='0' type='ode'>
      <max_step_size>0.001</max_step_size>
      <real_time_factor>1</real_time_factor>
      <real_time_update_rate>1000</real_time_update_rate>
    </physics>
    <scene>
```

```
  <ambient>0.4 0.4 0.4 1</ambient>
  <background>0.7 0.7 0.7 1</background>
  <shadows>1</shadows>
</scene>
<spherical_coordinates>
  <surface_model>EARTH_WGS84</surface_model>
  <latitude_deg>0</latitude_deg>
  <longitude_deg>0</longitude_deg>
  <elevation>0</elevation>
  <heading_deg>0</heading_deg>
</spherical_coordinates>
<model name='worldTB3TW53'>
  <pose frame=''>-0.09725 0.063678 0 0 -0 0</pose>
  <link name='Wall_10'>
    <collision name='Wall_10_Collision'>
      <geometry>
        <box>
          <size>0.5 0.15 0.4</size>
        </box>
      </geometry>
      <pose frame=''>0 0 0.2 0 -0 0</pose>
      <max_contacts>10</max_contacts>
      <surface>
        <contact>
          <ode/>
        </contact>
        <bounce/>
        <friction>
          <torsional>
            <ode/>
          </torsional>
          <ode/>
        </friction>
      </surface>
    </collision>
    <visual name='Wall_10_Visual'>
      <pose frame=''>0 0 0.2 0 -0 0</pose>
      <geometry>
        <box>
          <size>0.5 0.15 0.4</size>
        </box>
      </geometry>
      <material>
        <script>
          <uri>file://media/materials/scripts/gazebo.material</uri>
          <name>Gazebo/Grey</name>
        </script>
        <ambient>1 1 1 1</ambient>
      </material>
    </visual>
    <pose frame=''>-1.62175 0.344822 0 0 -0 3.1216</pose>
    <self_collide>0</self_collide>
    <kinematic>0</kinematic>
    <gravity>1</gravity>
  </link>
  <link name='Wall_11'>
    <collision name='Wall_11_Collision'>
      <geometry>
        <box>
          <size>1.54539 0.15 0.4</size>
        </box>
      </geometry>
      <pose frame=''>0 0 0.2 0 -0 0</pose>
      <max_contacts>10</max_contacts>
      <surface>
        <contact>
          <ode/>
        </contact>
        <bounce/>
        <friction>
          <torsional>
            <ode/>
          </torsional>
          <ode/>
        </friction>
      </surface>
```

```
    </collision>
    <visual name='Wall_11_Visual'>
      <pose frame=''>0 0 0.2 0 -0 0</pose>
      <geometry>
        <box>
          <size>1.54539 0.15 0.4</size>
        </box>
      </geometry>
      <material>
        <script>
          <uri>file://media/materials/scripts/gazebo.material</uri>
          <name>Gazebo/Grey</name>
        </script>
        <ambient>1 1 1 1</ambient>
      </material>
    </visual>
    <pose frame=''>-1.78875 -0.349178 0 0 0 -1.55933</pose>
    <self_collide>0</self_collide>
    <kinematic>0</kinematic>
    <gravity>1</gravity>
  </link>
  <link name='Wall_12'>
    <collision name='Wall_12_Collision'>
      <geometry>
        <box>
          <size>1.75045 0.15 0.4</size>
        </box>
      </geometry>
      <pose frame=''>0 0 0.2 0 -0 0</pose>
      <max_contacts>10</max_contacts>
      <surface>
        <contact>
          <ode/>
        </contact>
        <bounce/>
        <friction>
          <torsional>
            <ode/>
          </torsional>
          <ode/>
        </friction>
      </surface>
    </collision>
    <visual name='Wall_12_Visual'>
      <pose frame=''>0 0 0.2 0 -0 0</pose>
      <geometry>
        <box>
          <size>1.75045 0.15 0.4</size>
        </box>
      </geometry>
      <material>
        <script>
          <uri>file://media/materials/scripts/gazebo.material</uri>
          <name>Gazebo/Grey</name>
        </script>
        <ambient>1 1 1 1</ambient>
      </material>
    </visual>
    <pose frame=''>-0.98075 -1.04318 0 0 -0 0.004375</pose>
    <self_collide>0</self_collide>
    <kinematic>0</kinematic>
    <gravity>1</gravity>
  </link>
  <link name='Wall_13'>
    <collision name='Wall_13_Collision'>
      <geometry>
        <box>
          <size>0.840203 0.15 0.4</size>
        </box>
      </geometry>
      <pose frame=''>0 0 0.2 0 -0 0</pose>
      <max_contacts>10</max_contacts>
      <surface>
        <contact>
          <ode/>
        </contact>
```

```
      <bounce/>
      <friction>
        <torsional>
          <ode/>
        </torsional>
        <ode/>
      </friction>
    </surface>
  </collision>
  <visual name='Wall_13_Visual'>
    <pose frame=''>0 0 0.2 0 -0 0</pose>
    <geometry>
      <box>
        <size>0.840203 0.15 0.4</size>
      </box>
    </geometry>
    <material>
      <script>
        <uri>file://media/materials/scripts/gazebo.material</uri>
        <name>Gazebo/Grey</name>
      </script>
      <ambient>1 1 1 1</ambient>
    </material>
  </visual>
  <pose frame=''>-0.18475 -0.694178 0 0 -0 1.58237</pose>
  <self_collide>0</self_collide>
  <kinematic>0</kinematic>
  <gravity>1</gravity>
</link>
<link name='Wall_14'>
  <collision name='Wall_14_Collision'>
    <geometry>
      <box>
        <size>0.743054 0.15 0.4</size>
      </box>
    </geometry>
    <pose frame=''>0 0 0.2 0 -0 0</pose>
    <max_contacts>10</max_contacts>
    <surface>
      <contact>
        <ode/>
      </contact>
      <bounce/>
      <friction>
        <torsional>
          <ode/>
        </torsional>
        <ode/>
      </friction>
    </surface>
  </collision>
  <visual name='Wall_14_Visual'>
    <pose frame=''>0 0 0.2 0 -0 0</pose>
    <geometry>
      <box>
        <size>0.743054 0.15 0.4</size>
      </box>
    </geometry>
    <material>
      <script>
        <uri>file://media/materials/scripts/gazebo.material</uri>
        <name>Gazebo/Grey</name>
      </script>
      <ambient>1 1 1 1</ambient>
    </material>
  </visual>
  <pose frame=''>0.10775 -0.352678 0 0 0 -0.01349</pose>
  <self_collide>0</self_collide>
  <kinematic>0</kinematic>
  <gravity>1</gravity>
</link>
<link name='Wall_15'>
  <collision name='Wall_15_Collision'>
    <geometry>
      <box>
        <size>0.894043 0.15 0.4</size>
```

```
      </box>
    </geometry>
    <pose frame=''>0 0 0.2 0 -0 0</pose>
    <max_contacts>10</max_contacts>
    <surface>
      <contact>
        <ode/>
      </contact>
      <bounce/>
      <friction>
        <torsional>
          <ode/>
        </torsional>
        <ode/>
      </friction>
    </surface>
  </collision>
  <visual name='Wall_15_Visual'>
    <pose frame=''>0 0 0.2 0 -0 0</pose>
    <geometry>
      <box>
        <size>0.894043 0.15 0.4</size>
      </box>
    </geometry>
    <material>
      <script>
        <uri>file://media/materials/scripts/gazebo.material</uri>
        <name>Gazebo/Grey</name>
      </script>
      <ambient>1 1 1 1</ambient>
    </material>
  </visual>
  <pose frame=''>0.40025 -0.728678 0 0 0 -1.58155</pose>
  <self_collide>0</self_collide>
  <kinematic>0</kinematic>
  <gravity>1</gravity>
</link>
<link name='Wall_16'>
  <collision name='Wall_16_Collision'>
    <geometry>
      <box>
        <size>1.52311 0.15 0.4</size>
      </box>
    </geometry>
    <pose frame=''>0 0 0.2 0 -0 0</pose>
    <max_contacts>10</max_contacts>
    <surface>
      <contact>
        <ode/>
      </contact>
      <bounce/>
      <friction>
        <torsional>
          <ode/>
        </torsional>
        <ode/>
      </friction>
    </surface>
  </collision>
  <visual name='Wall_16_Visual'>
    <pose frame=''>0 0 0.2 0 -0 0</pose>
    <geometry>
      <box>
        <size>1.52311 0.15 0.4</size>
      </box>
    </geometry>
    <material>
      <script>
        <uri>file://media/materials/scripts/gazebo.material</uri>
        <name>Gazebo/Grey</name>
      </script>
      <ambient>1 1 1 1</ambient>
    </material>
  </visual>
  <pose frame=''>1.08275 -1.10068 0 0 -0 0</pose>
  <self_collide>0</self_collide>
```

```
  <kinematic>0</kinematic>
  <gravity>1</gravity>
 </link>
 <link name='Wall_19'>
  <collision name='Wall_19_Collision'>
   <geometry>
    <box>
     <size>1.20032 0.15 0.4</size>
    </box>
   </geometry>
   <pose frame=''>0 0 0.2 0 -0 0</pose>
   <max_contacts>10</max_contacts>
   <surface>
    <contact>
     <ode/>
    </contact>
    <bounce/>
    <friction>
     <torsional>
      <ode/>
     </torsional>
     <ode/>
    </friction>
   </surface>
  </collision>
  <visual name='Wall_19_Visual'>
   <pose frame=''>0 0 0.2 0 -0 0</pose>
   <geometry>
    <box>
     <size>1.20032 0.15 0.4</size>
    </box>
   </geometry>
   <material>
    <script>
     <uri>file://media/materials/scripts/gazebo.material</uri>
     <name>Gazebo/Grey</name>
    </script>
    <ambient>1 1 1 1</ambient>
   </material>
  </visual>
  <pose frame=''>1.64875 0.149322 0 0 -0 1.63844</pose>
  <self_collide>0</self_collide>
  <kinematic>0</kinematic>
  <gravity>1</gravity>
 </link>
 <link name='Wall_21'>
  <collision name='Wall_21_Collision'>
   <geometry>
    <box>
     <size>0.75 0.15 0.4</size>
    </box>
   </geometry>
   <pose frame=''>0 0 0.2 0 -0 0</pose>
   <max_contacts>10</max_contacts>
   <surface>
    <contact>
     <ode/>
    </contact>
    <bounce/>
    <friction>
     <torsional>
      <ode/>
     </torsional>
     <ode/>
    </friction>
   </surface>
  </collision>
  <visual name='Wall_21_Visual'>
   <pose frame=''>0 0 0.2 0 -0 0</pose>
   <geometry>
    <box>
     <size>0.75 0.15 0.4</size>
    </box>
   </geometry>
   <material>
    <script>
```

```
        <uri>file://media/materials/scripts/gazebo.material</uri>
        <name>Gazebo/Grey</name>
      </script>
      <ambient>1 1 1 1</ambient>
    </material>
  </visual>
  <pose frame=''>1.76925 -0.800678 0 0 -0 1.5708</pose>
  <self_collide>0</self_collide>
  <kinematic>0</kinematic>
  <gravity>1</gravity>
</link>
<link name='Wall_22'>
  <collision name='Wall_22_Collision'>
    <geometry>
      <box>
        <size>0.301384 0.15 0.4</size>
      </box>
    </geometry>
    <pose frame=''>0 0 0.2 0 -0 0</pose>
    <max_contacts>10</max_contacts>
    <surface>
      <contact>
        <ode/>
      </contact>
      <bounce/>
      <friction>
        <torsional>
          <ode/>
        </torsional>
        <ode/>
      </friction>
    </surface>
  </collision>
  <visual name='Wall_22_Visual'>
    <pose frame=''>0 0 0.2 0 -0 0</pose>
    <geometry>
      <box>
        <size>0.301384 0.15 0.4</size>
      </box>
    </geometry>
    <material>
      <script>
        <uri>file://media/materials/scripts/gazebo.material</uri>
        <name>Gazebo/Grey</name>
      </script>
      <ambient>1 1 1 1</ambient>
    </material>
  </visual>
  <pose frame=''>1.72675 -0.437678 0 0 -0 2.16427</pose>
  <self_collide>0</self_collide>
  <kinematic>0</kinematic>
  <gravity>1</gravity>
</link>
<link name='Wall_4'>
  <collision name='Wall_4_Collision'>
    <geometry>
      <box>
        <size>0.5 0.15 0.4</size>
      </box>
    </geometry>
    <pose frame=''>0 0 0.2 0 -0 0</pose>
    <max_contacts>10</max_contacts>
    <surface>
      <contact>
        <ode/>
      </contact>
      <bounce/>
      <friction>
        <torsional>
          <ode/>
        </torsional>
        <ode/>
      </friction>
    </surface>
  </collision>
  <visual name='Wall_4_Visual'>
```

```
    <pose frame=''>0 0 0.2 0 -0 0</pose>
    <geometry>
      <box>
        <size>0.5 0.15 0.4</size>
      </box>
    </geometry>
    <material>
      <script>
        <uri>file://media/materials/scripts/gazebo.material</uri>
        <name>Gazebo/Grey</name>
      </script>
      <ambient>1 1 1 1</ambient>
    </material>
  </visual>
  <pose frame=''>0.93525 0.854322 0 0 0 -1.5708</pose>
  <self_collide>0</self_collide>
  <kinematic>0</kinematic>
  <gravity>1</gravity>
</link>
<link name='Wall_5'>
  <collision name='Wall_5_Collision'>
    <geometry>
      <box>
        <size>0.827905 0.15 0.4</size>
      </box>
    </geometry>
    <pose frame=''>0 0 0.2 0 -0 0</pose>
    <max_contacts>10</max_contacts>
    <surface>
      <contact>
        <ode/>
      </contact>
      <bounce/>
      <friction>
        <torsional>
          <ode/>
        </torsional>
        <ode/>
      </friction>
    </surface>
  </collision>
  <visual name='Wall_5_Visual'>
    <pose frame=''>0 0 0.2 0 -0 0</pose>
    <geometry>
      <box>
        <size>0.827905 0.15 0.4</size>
      </box>
    </geometry>
    <material>
      <script>
        <uri>file://media/materials/scripts/gazebo.material</uri>
        <name>Gazebo/Grey</name>
      </script>
      <ambient>1 1 1 1</ambient>
    </material>
  </visual>
  <pose frame=''>1.27425 0.676322 0 0 0 -0.008849</pose>
  <self_collide>0</self_collide>
  <kinematic>0</kinematic>
  <gravity>1</gravity>
</link>
<link name='Wall_7'>
  <collision name='Wall_7_Collision'>
    <geometry>
      <box>
        <size>2.53961 0.15 0.4</size>
      </box>
    </geometry>
    <pose frame=''>0 0 0.2 0 -0 0</pose>
    <max_contacts>10</max_contacts>
    <surface>
      <contact>
        <ode/>
      </contact>
      <bounce/>
      <friction>
```

```
        <torsional>
          <ode/>
        </torsional>
        <ode/>
      </friction>
    </surface>
  </collision>
  <visual name='Wall_7_Visual'>
    <pose frame=''>0 0 0.2 0 -0 0</pose>
    <geometry>
      <box>
        <size>2.53961 0.15 0.4</size>
      </box>
    </geometry>
    <material>
      <script>
        <uri>file://media/materials/scripts/gazebo.material</uri>
        <name>Gazebo/CeilingTiled</name>
      </script>
      <ambient>1 1 1 1</ambient>
    </material>
  </visual>
  <pose frame=''>-0.25925 1.06432 0 0 0 -0.029293</pose>
  <self_collide>0</self_collide>
  <kinematic>0</kinematic>
  <gravity>1</gravity>
 </link>
 <link name='Wall_9'>
  <collision name='Wall_9_Collision'>
    <geometry>
      <box>
        <size>0.908032 0.15 0.4</size>
      </box>
    </geometry>
    <pose frame=''>0 0 0.2 0 -0 0</pose>
    <max_contacts>10</max_contacts>
    <surface>
      <contact>
        <ode/>
      </contact>
      <bounce/>
      <friction>
        <torsional>
          <ode/>
        </torsional>
        <ode/>
      </friction>
    </surface>
  </collision>
  <visual name='Wall_9_Visual'>
    <pose frame=''>0 0 0.2 0 -0 0</pose>
    <geometry>
      <box>
        <size>0.908032 0.15 0.4</size>
      </box>
    </geometry>
    <material>
      <script>
        <uri>file://media/materials/scripts/gazebo.material</uri>
        <name>Gazebo/Grey</name>
      </script>
      <ambient>1 1 1 1</ambient>
    </material>
  </visual>
  <pose frame=''>-1.45025 0.720322 0 0 0 -1.56156</pose>
  <self_collide>0</self_collide>
  <kinematic>0</kinematic>
  <gravity>1</gravity>
 </link>
 <static>1</static>
</model>
<state world_name='default'>
 <sim_time>63 116000000</sim_time>
 <real_time>63 496772576</real_time>
 <wall_time>1544170680 513200632</wall_time>
 <iterations>63116</iterations>
```

```
<model name='ground_plane'>
  <pose frame=''>0 0 0 0 -0 0</pose>
  <scale>1 1 1</scale>
  <link name='link'>
    <pose frame=''>0 0 0 0 -0 0</pose>
    <velocity>0 0 0 0 -0 0</velocity>
    <acceleration>0 0 0 0 -0 0</acceleration>
    <wrench>0 0 0 0 -0 0</wrench>
  </link>
</model>
<model name='worldTB3TW53'>
  <pose frame=''>-0.09725 0.063678 0 0 -0 0</pose>
  <scale>1 1 1</scale>
  <link name='Wall_10'>
    <pose frame=''>-1.719 0.4085 0 0 -0 3.1216</pose>
    <velocity>0 0 0 0 -0 0</velocity>
    <acceleration>0 0 0 0 -0 0</acceleration>
    <wrench>0 0 0 0 -0 0</wrench>
  </link>
  <link name='Wall_11'>
    <pose frame=''>-1.886 -0.2855 0 0 0 -1.55933</pose>
    <velocity>0 0 0 0 -0 0</velocity>
    <acceleration>0 0 0 0 -0 0</acceleration>
    <wrench>0 0 0 0 -0 0</wrench>
  </link>
  <link name='Wall_12'>
    <pose frame=''>-1.078 -0.979502 0 0 -0 0.004375</pose>
    <velocity>0 0 0 0 -0 0</velocity>
    <acceleration>0 0 0 0 -0 0</acceleration>
    <wrench>0 0 0 0 -0 0</wrench>
  </link>
  <link name='Wall_13'>
    <pose frame=''>-0.282 -0.6305 0 0 -0 1.58237</pose>
    <velocity>0 0 0 0 -0 0</velocity>
    <acceleration>0 0 0 0 -0 0</acceleration>
    <wrench>0 0 0 0 -0 0</wrench>
  </link>
  <link name='Wall_14'>
    <pose frame=''>0.0105 -0.289 0 0 0 -0.01349</pose>
    <velocity>0 0 0 0 -0 0</velocity>
    <acceleration>0 0 0 0 -0 0</acceleration>
    <wrench>0 0 0 0 -0 0</wrench>
  </link>
  <link name='Wall_15'>
    <pose frame=''>0.303 -0.665 0 0 0 -1.58155</pose>
    <velocity>0 0 0 0 -0 0</velocity>
    <acceleration>0 0 0 0 -0 0</acceleration>
    <wrench>0 0 0 0 -0 0</wrench>
  </link>
  <link name='Wall_16'>
    <pose frame=''>0.9855 -1.037 0 0 -0 0</pose>
    <velocity>0 0 0 0 -0 0</velocity>
    <acceleration>0 0 0 0 -0 0</acceleration>
    <wrench>0 0 0 0 -0 0</wrench>
  </link>
  <link name='Wall_19'>
    <pose frame=''>1.5515 0.213 0 0 -0 1.63844</pose>
    <velocity>0 0 0 0 -0 0</velocity>
    <acceleration>0 0 0 0 -0 0</acceleration>
    <wrench>0 0 0 0 -0 0</wrench>
  </link>
  <link name='Wall_21'>
    <pose frame=''>1.672 -0.737 0 0 -0 1.5708</pose>
    <velocity>0 0 0 0 -0 0</velocity>
    <acceleration>0 0 0 0 -0 0</acceleration>
    <wrench>0 0 0 0 -0 0</wrench>
  </link>
  <link name='Wall_22'>
    <pose frame=''>1.6295 -0.374 0 0 -0 2.16427</pose>
    <velocity>0 0 0 0 -0 0</velocity>
    <acceleration>0 0 0 0 -0 0</acceleration>
    <wrench>0 0 0 0 -0 0</wrench>
  </link>
  <link name='Wall_4'>
    <pose frame=''>0.838 0.918 0 0 0 -1.5708</pose>
    <velocity>0 0 0 0 -0 0</velocity>
```

```
        <acceleration>0 0 0 0 -0 0</acceleration>
        <wrench>0 0 0 0 -0 0</wrench>
      </link>
      <link name='Wall_5'>
        <pose frame=''>1.177 0.74 0 0 0 -0.008849</pose>
        <velocity>0 0 0 0 -0 0</velocity>
        <acceleration>0 0 0 0 -0 0</acceleration>
        <wrench>0 0 0 0 -0 0</wrench>
      </link>
      <link name='Wall_7'>
        <pose frame=''>-0.3565 1.128 0 0 0 -0.029293</pose>
        <velocity>0 0 0 0 -0 0</velocity>
        <acceleration>0 0 0 0 -0 0</acceleration>
        <wrench>0 0 0 0 -0 0</wrench>
      </link>
      <link name='Wall_9'>
        <pose frame=''>-1.5475 0.784 0 0 0 -1.56156</pose>
        <velocity>0 0 0 0 -0 0</velocity>
        <acceleration>0 0 0 0 -0 0</acceleration>
        <wrench>0 0 0 0 -0 0</wrench>
      </link>
    </model>
    <light name='sun'>
      <pose frame=''>0 0 10 0 -0 0</pose>
    </light>
  </state>
  <gui fullscreen='0'>
    <camera name='user_camera'>
      <pose frame=''>-1.78409 1.99167 6.00544 0 1.17964 -1.143</pose>
      <view_controller>orbit</view_controller>
      <projection_type>perspective</projection_type>
    </camera>
  </gui>
 </world>
</sdf>
```

Annexe 4 : Launch du virtual worldTB3TW53

```
<launch>
   <!--arg name="model" default="$(env TURTLEBOT3_MODEL)" doc="model type
[burger, waffle, waffle_pi]"/-->
<!--   <arg name="tb3" default="tb3"/> -->
<!--   <arg name="tb3x_pos" default="0.0"/> -->
<!--   <arg name="tb3y_pos" default="0.0"/> -->
<!--   <arg name="tb3z_pos" default="0.0"/> -->

  <include file="$(find gazebo_ros)/launch/empty_world.launch">
    <arg name="world_name" value="$(find
turtlebot3_gazebo)/worlds/mapTB3TW53.world"/>
    <arg name="paused" value="false"/>
    <arg name="use_sim_time" value="true"/>
    <arg name="gui" value="true"/>
    <arg name="headless" value="false"/>
    <arg name="debug" value="false"/>
  </include>

  <!--param name="robot_description" command="$(find xacro)/xacro -a enlever-
inorder $(find turtlebot3_description)/urdf/turtlebot3_waffle_pi.urdf.xacro" /-->


  <!--node pkg="gazebo_ros" type="spawn_model" name="spawn_urdf" args="-urdf -
model turtlebot3_$(arg model) -x $(arg x_pos) -y $(arg y_pos) -z $(arg z_pos) -
param robot_description" /-->

  <!--node pkg="gazebo_ros" type="spawn_model" name="spawn_urdf" args="-urdf -
model $(arg tb3) -x $(arg tb3x_pos) -y $(arg tb3y_pos) -z $(arg tb3z_pos) -param
robot_description" /-->
</launch>
```

Annexe 5 : Programme Python « Aller à un endroit particulier de la carte »

```
#!/usr/bin/env python
# license removed for brevity

import rospy
import actionlib
from move_base_msgs.msg import MoveBaseAction, MoveBaseGoal

def movebase_client():
    print "lancement du client movebase"
    client = actionlib.SimpleActionClient('virtual_tb3_1/move_base',MoveBaseAction)
    client.wait_for_server()

    goal = MoveBaseGoal()
    goal.target_pose.header.frame_id = "map"
    goal.target_pose.header.stamp = rospy.Time.now()
    goal.target_pose.pose.position.x = -1.8
    goal.target_pose.pose.position.y = 0.7
    goal.target_pose.pose.orientation.w = 1

    client.send_goal(goal)
    wait = client.wait_for_result()

    if not wait:
        rospy.logerr("Action server not available!")
        rospy.signal_shutdown("Action server not available!")
    else:
        return client.get_result()

    print "fin de la fonction"

if __name__ == '__main__':
    try:
        rospy.init_node('movebase_client_py')
        result = movebase_client()
        if result:
            rospy.loginfo("Goal execution done!")
    except rospy.ROSInterruptException:
        rospy.loginfo("Navigation test finished.")
```

Annexe 6 : Launch du robot virtuel

```
<launch>
  <include file="$(find gazebo_ros)/launch/empty_world.launch">
    <arg name="world_name"
value="/home/dambelle/Téléchargements/mapTB3TW53.world"/>
    <arg name="paused" value="false"/>
    <arg name="use_sim_time" value="true"/>
    <arg name="gui" value="true"/>
    <arg name="headless" value="false"/>
    <arg name="debug" value="false"/>
  </include>

  <node pkg="map_server" name="map_server" type="map_server"
args="/home/dambelle/Téléchargements/map.yaml" />

  <group ns="virtual_tb3_1">
    <param name="tf_prefix" value="virtual_tb3_1"/>
    <param name="robot_description" command="$(find xacro)/xacro --inorder $(find
turtlebot3_description)/urdf/turtlebot3_waffle_pi.urdf.xacro" />

    <node pkg="robot_state_publisher" type="robot_state_publisher"
name="robot_state_publisher">
      <param name="publish_frequency" type="double" value="50.0"/>
    </node>

    <node pkg="gazebo_ros" type="spawn_model" name="spawn_urdf" args="-urdf -
model turtlebot3_waffle_pi -x 1.0 -y 0.0 -z 0.0 -param robot_description" />

    <node pkg="amcl" type="amcl" name="amcl">
      <remap from="map"                  to="/map"/>
      <remap from="static_map"             to="/static_map"/>

      <param name="use_map_topic"            value="true"/>
      <param name="global_frame_id"          value="/map"/>
      <param name="odom_frame_id"            value="/virtual_tb3_1/odom"/>
      <param name="base_frame_id"            value="/virtual_tb3_1/base_footprint"/>

      <param name="min_particles"            value="500"/>
      <param name="max_particles"            value="3000"/>
      <param name="kld_err"                value="0.02"/>
      <param name="update_min_d"             value="0.20"/>
      <param name="update_min_a"             value="0.20"/>
      <param name="resample_interval"        value="1"/>
      <param name="transform_tolerance"      value="0.5"/>
      <param name="recovery_alpha_slow"      value="0.00"/>
      <param name="recovery_alpha_fast"      value="0.00"/>
      <param name="initial_pose_x"           value="1.0"/>
      <param name="initial_pose_y"           value="0.0"/>
```

```
    <param name="initial_pose_a"           value="0.0"/>
    <param name="gui_publish_rate"         value="50.0"/>

    <param name="laser_max_range"          value="3.5"/>
    <param name="laser_max_beams"          value="180"/>
    <param name="laser_z_hit"              value="0.5"/>
    <param name="laser_z_short"            value="0.05"/>
    <param name="laser_z_max"              value="0.05"/>
    <param name="laser_z_rand"             value="0.5"/>
    <param name="laser_sigma_hit"          value="0.2"/>
    <param name="laser_lambda_short"       value="0.1"/>
    <param name="laser_likelihood_max_dist" value="2.0"/>
    <param name="laser_model_type"         value="likelihood_field"/>

    <param name="odom_model_type"          value="diff"/>
    <param name="odom_alpha1"              value="0.1"/>
    <param name="odom_alpha2"              value="0.1"/>
    <param name="odom_alpha3"              value="0.1"/>
    <param name="odom_alpha4"              value="0.1"/>
  </node>

  <node pkg="move_base" type="move_base" respawn="false" name="move_base"
output="screen">
    <remap from="odom" to="/virtual_tb3_1/odom" />
    <remap from="cmd_vel" to="/virtual_tb3_1/cmd_vel" />
    <remap from="map" to="/map" />

   <param name="base_local_planner"
value="dwa_local_planner/DWAPlannerROS" />
    <rosparam file="$(find
turtlebot3_navigation)/param/costmap_common_params_waffle_pi.yaml"
command="load" ns="global_costmap" />
    <rosparam file="$(find
turtlebot3_navigation)/param/costmap_common_params_waffle_pi.yaml"
command="load" ns="local_costmap" />
    <rosparam file="$(find
turtlebot3_navigation)/param/local_costmap_params.yaml" command="load" />
    <rosparam file="$(find
turtlebot3_navigation)/param/global_costmap_params.yaml" command="load" />
    <rosparam file="$(find turtlebot3_navigation)/param/move_base_params.yaml"
command="load" />
    <rosparam file="$(find
turtlebot3_navigation)/param/dwa_local_planner_params_waffle_pi.yaml"
command="load" />
   </node>
 </group>

 <group ns="virtual_tb3_2">
   <param name="tf_prefix" value="virtual_tb3_2"/>
   <param name="robot_description" command="$(find xacro)/xacro --inorder $(find
```

```
turtlebot3_description)/urdf/turtlebot3_burger.urdf.xacro" />
    <node pkg="gazebo_ros" type="spawn_model" name="spawn_urdf" args="-urdf -
model turtlebot3_burger -x -1.0 -y 0.0 -z 0.0 -param robot_description" />
  </group>

  <node pkg="rviz" type="rviz" name="rviz" args="-d
/home/dambelle/Téléchargements/turtlebot3_navigation.rviz"/>
</launch>
```

Annexe 7 : Programme Python arrêt du robot

```
#!/usr/bin/env python
import rospy
from nav_msgs.msg import Odometry
from geometry_msgs.msg import Twist
from math import *

#-------------------------------------------------------------------------------------------------------------
----------------------------
# 1. Mettre dans des variables les positions en x et y des deux robots
#-------------------------------------------------------------------------------------------------------------
----------------------------

x_r = 0
y_r = 0
x_v = 0
y_v = 0


def callback(msg):
    global x_r
    global y_r
    x_r = msg.pose.pose.position.x
    y_r = msg.pose.pose.position.y


def callback2(msg):
    global x_v
    global y_v
    x_v = msg.pose.pose.position.x
    y_v = msg.pose.pose.position.y


rospy.init_node('robot_stop')
odom_sub = rospy.Subscriber('/tb3_0/odom', Odometry, callback)
odom_sub_2 = rospy.Subscriber('/tb3_1/odom', Odometry, callback2)
rate = rospy.Rate(10) # 10hz



#-------------------------------------------------------------------------------------------------------------
----------------------------
# 2. Provoquer l arret
#-------------------------------------------------------------------------------------------------------------
----------------------------

def arret():
    pub = rospy.Publisher('/tb3_0/cmd_vel', Twist, queue_size=10)
```

```
    vel_msg = Twist()

    while not rospy.is_shutdown() :
        #print (x_r, y_r, x_v, y_v)
        if sqrt ((x_r-x_v)**2 + (y_r-y_v)**2) < 10 : #remettre 0.3 (10 pour les tests)
            vel_msg.linear.x = 0
         #print sqrt ((x_r-x_v)**2 + (y_r-y_v)**2) #test pour voir si on entre dans le if ->
OK
            vel_msg.linear.y = 0
            vel_msg.linear.z = 0
            vel_msg.angular.x = 0
            vel_msg.angular.y = 0
            vel_msg.angular.z = 0
         print vel_msg #test pour voir si les valeur dans le twist sont bien toutes a 0 ->
OK
         pub.publish(vel_msg) # Probleme sur cette ligne. Ne publie pas
         rate.sleep()

if __name__=='__main__':
    try :
        arret()
    except rospy.ROSInterruptException :
        pass
```