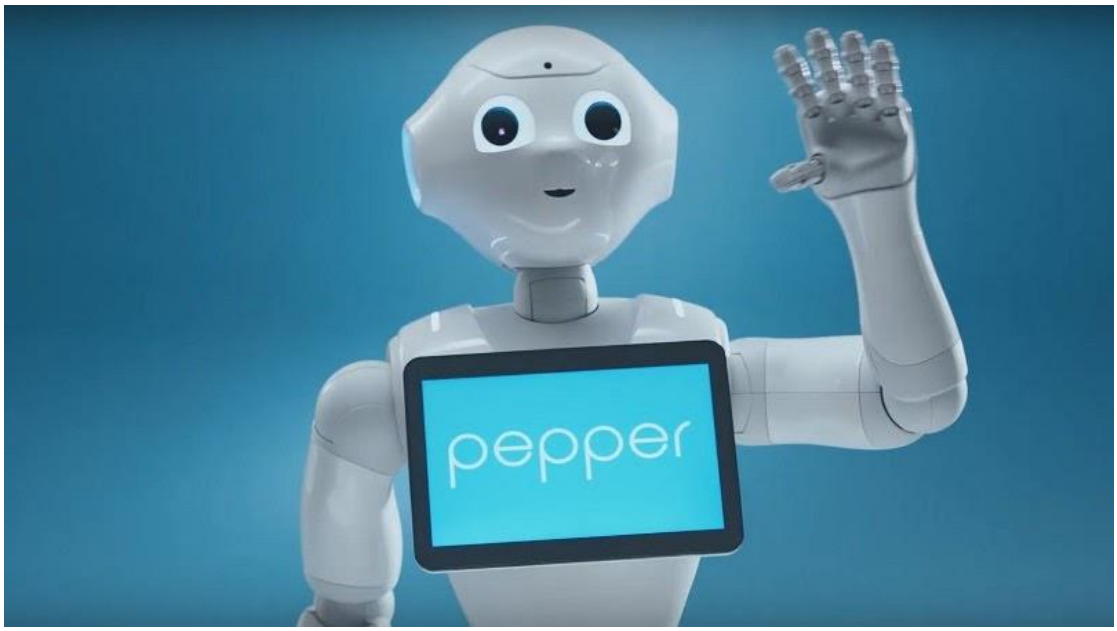
Projet de 2e année

« Mise en œuvre de protocole de renforcement cognitif avec un robot humanoïde Pepper »



Auteur : Linxue LAI

Email : linxue.lai@depinfonancy.net

Encadrant : Patrick HENAFF

2019 - 2020

Table des matières

1 Présentation du projet

L'objectif de ce projet est d'évaluer la pertinence de l'usage d'un robot humanoïde pour la stimulation cognitive de personne âgée en EPAHD afin de laisser du temps au personnelle de soin de s'occuper des soins.

Il faut étudier la mise en œuvre de protocole de renforcement physique pour personnes âgées avec le même robot humanoïde : définition du protocole par la partie santé (promenade, coordination motrice des membres supérieurs, etc.)

Selon l'avancement du projet, j'ai mené des études sur le robot Pepper à trois niveaux de techniques :

1) L'utilisation du logiciel *Choregraphe*, qui m'a aidé à apprendre de manière plus complète comment contrôler le robot humanoïde et personnaliser le mouvement du robot.

2) L'utilisation de librairie *qibullet* et la programmation en Python pour visualiser et simuler la promenade du robot sur l'ordinateur. Par exemple, laissons-le se déplacer librement dans un environnement virtuel qu'on a construit afin de lui permettre d'éviter certains obstacles dans le chemin.

3) L'utilisation du SDK Python Naoqi pour contrôler par programme le vrai robot afin de lui faire suivre l'itinéraire que nous avons défini et implémenter certaines actions personnalisées aux points spécifiés.

2 Découverte du robot Pepper

2.1 Pepper, le robot humanoïde

Pepper est un robot humanoïde, mesurant 120 cm et pesant 28 kilogrammes. Il a été développé par la société *SoftBank Robotic* pour communiquer avec les humains, grâce à sa voix et sa gestuelle, naturelles et intuitives. Il est capable d'entendre, de parler, d'identifier les principales émotions, de voir et de se déplacer. Toutes ces actions sont possibles grâce à ses caméras, ses microphones, ses haut-parleurs, ses capteurs sensoriels, sa tablette et ses trois roues multidirectionnelles [1]. Voir l'apparition du robot ci-dessous :

Figure 1. L'apparition du robot

2.2 Connection avec Pepper

Le système d'exploitation de l'ordinateur portable utilisé lors du projet est de Windows 64 bits. Pour contrôler le robot, une connexion doit être effectuée entre le robot et un ordinateur. Il existe 2 façons pour le réaliser : une connexion filaire ou une connexion Wifi.

Figure 2. Connexion filaire du robot et la câble Ethernet

Pour la connexion filaire, nous pouvons utiliser une câble Ethernet qui relie un ordinateur avec le robot comme ce qui est présenté dans la figure 2.

Après avoir configuré un routeur, la connexion Wifi avec robot Pepper sera accessible.

Dans la phase initiale de ce projet, une connexion filaire a été utilisée pour tester le fonctionnement du robot, puis une connexion sans fil a été utilisée pour s'assurer que le robot pouvait se déplacer librement dans une certaine zone.

2.3 Vie autonome du Pepper

Par défaut, Pepper a la vie autonome activée, ce qui signifie que le robot est réactif et cherche à interagir avec une stimulation extérieure. Si nous orientons sa tête vers les humains, il réagit aux sons, aux mouvements et aux contacts tactiles. Il est même capable de dialoguer avec nous :

Approchons-nous du robot aux alentours d'un mètre. Quand il est prêt à nous écouter, ses yeux doivent devenir bleus. Disons une phrase courte, qui est pré-enregistrée dans le robot. S'il nous a entendu, ses yeux deviennent verts. Il traite l'information. Ses yeux sont blancs quand le robot nous répond. Nous pouvons parler avec Pepper en français.

Il est possible de désactiver la vie autonome du robot. Lors de la désactivation de la vie autonome, Pepper ne répond plus mais exécute les commandes de programmation avec moins de respect pour la sécurité. Cet aspect, bien que moins sûr, on facilite la manœuvre de Pepper. Les deux modes ne sont pas tout à fait compatibles car si l'on veut déplacer le robot, il faut renoncer à l'aspect interaction et vice versa.

Afin de mieux tester des mouvements définit par nous-même, nous devons désactiver la vie autonome du robot. Cela peut être facilement réalisé par le logiciel d'interaction de Pepper (*Choregraphe*). Et pour le "réveiller" nous pouvons également utiliser bouton du *Choregraphe* ou utiliser des fonctions en python Naoqi.

3 L'utilisation du *Choregraphe*

Choregraphe est un logiciel qui permet de créer des applications complexes sans écrire des lignes de code et de les tester sur un robot virtuel ou sur un robot réel. Il nous permet de contrôler le robot Pepper et également de modifier des boîtes déjà intégrés au logiciel en y intégrant son

propre code Python.

Afin d'apprendre rapidement la fonction et la configuration du robot Pepper, j'ai lu le document Guide-Pepper au début des séances du projet et utilisé logiciel *Choregraphe* pour faire des expériences de simulation.

3.1 Installation

Les étapes de l'installation est relativement simple :

1) Téléchargeons le logiciel *Choregraphe* 2.5.10.7 win32 et l'activer.

2) Démarrons l'installation, acceptons le contrat de licence et suivez les étapes.

3) Ouvrons *Choregraphe* et activons la licence en entrant la clé valide. Si nous n'en avons pas, nous pouvons utiliser la version d'essai pendant 80 jours.

3.2 Expériences de simulation

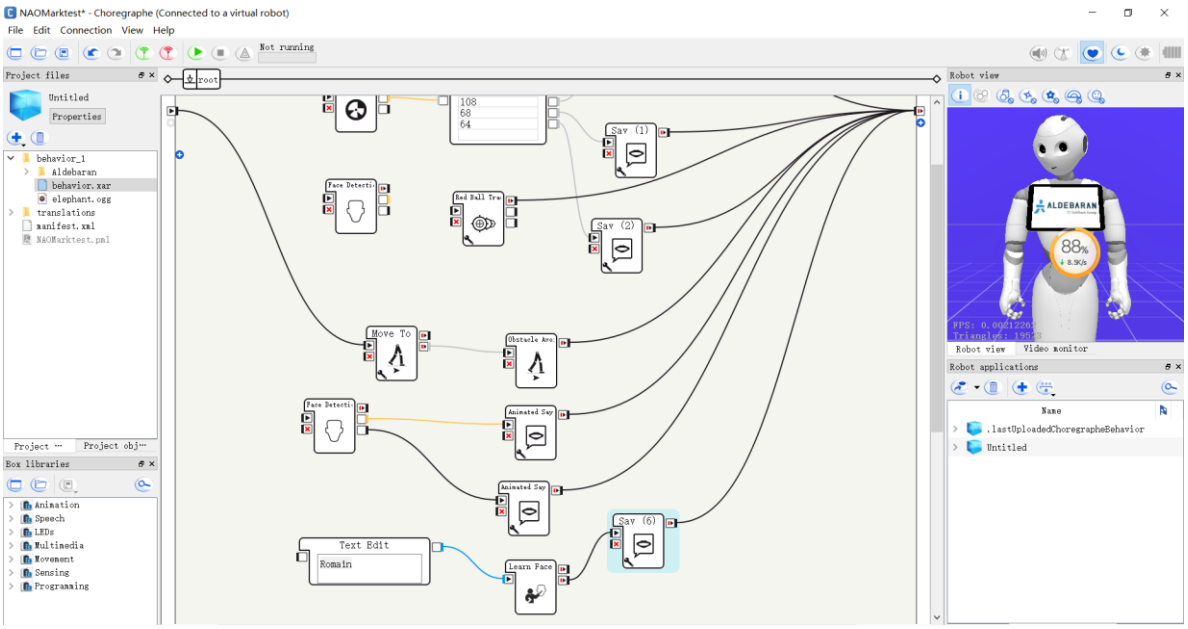


Figure 3. L'interface du Choregraphe

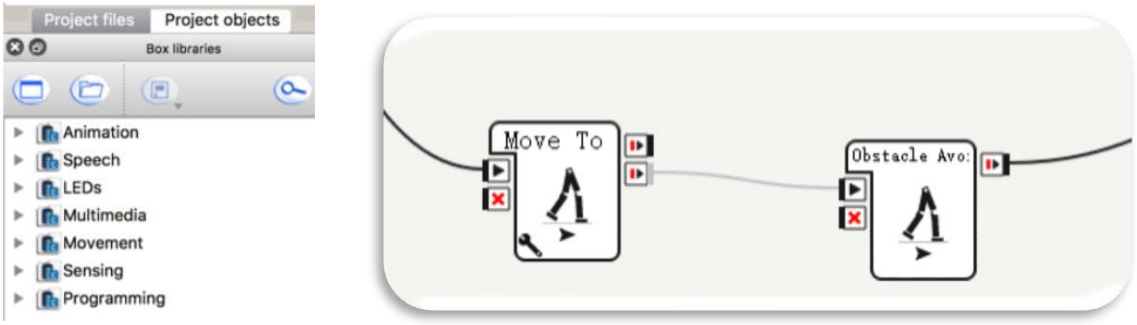


*Figure 4. La fenêtre **Box librarie** et des boites*

Le figure 3 et 4 présente l'interface du *Choregraphe* et la fenêtre ***Box libraries***. Nous y trouvons toutes les actions (boîtes) que nous pouvons ajouter dans l'espace de programmation. Elle est constituée de plusieurs listes qui classent les boîtes de comportement dans des dossiers.

- Pour l'objectif de contrôler le Pepper à marcher dans le couloir et tourner s'il rencontre des obstacles. J'ai fait des tests par *Choregraphe* en utilisant les boites « Move To » et «

Obstacle Avoidance ».

- Pour mémoriser les visages de humains et les détecter, il faut utiliser les boites « Learn Face » et « Face Reco. ».
- Pour faire des interactions avec un humain, il existe plusieurs boîtes intéressantes. Par exemple, les boîtes « Choice » et « Switch Case » permet au robot de choisir de réagir différemment s selon les différentes informations d'entrée (Par exemple, il répond à différentes phrases selon ce qu'il a entendu ou vu.)

En utilisant *Choregraphe* pour simuler des expériences sur le robot virtuel et le vrai robot, nous pouvons constater que certaines fonctions et certains codes peuvent être portés sur python Naoqi. Par exemple, grâce à la fonction **move (x, y, theta)** ou **moveTo (x, y, theta)**, nous pouvons contrôler le robot de faire promenade par python 2.7. Ce sera donc possible de contrôler le robot pour suivre l'itinéraire que nous voulons. Et un autre exemple : en utilisant la fonction **say (str (phrase))**, le robot va dire la phrase spécifiée par nous.

4 L'utilisation du *qibullet*

Pendant le confinement, il n'a pas été possible au début d'entrer à l'école pour faire des expériences avec le robot réel. Donc j'ai construit un environnement virtuel sur mon ordinateur pour faire des expériences virtuelles. Il y a deux façons différentes d'interagir avec le modèle virtuel du robot en simulation : l'API *qiBullet* basée sur Python ou le Framework ROS. J'ai choisi l'API *qibullet*.

4.1 L'installation et la description du *qibullet*

La module *qiBullet* est une simulation Python basée sur Bullet pour les robots de *SoftBank Robotics*. Actuellement, seul le robot Pepper peut être simulé.

Selon le document de *qiBullet*, nous pouvons l'installer via *pip* en python 2.7 ou python 3 :

```
pip install --user qibullet
```

Mais pour l'environnement de mon ordinateur, l'installation sur python 2.7 posait des problèmes, donc je l'ai installé sur python 3 par ses étapes :

1) Télécharger et installer à nouveau python3 et pip3.
2) Installer NET Framework 4.8 Dev pack et Build Tools for Visual Studio 2017
3) Suivre le lien : https://pypi.org/project/qibullet/
4) Installer les modules nécessaires :
   - numpy
   - pybullet
5) Installer le module *qiBullet* par python 3 :

```
pip3 install --user qibullet
```

6) Installer *OpenCV*-Python : (version 4.2.0.32)

```
pip3 install opencv-python
```

7) Lancer pepper_basic.py (Annexe 1), on verra La fenêtre de la simulation :

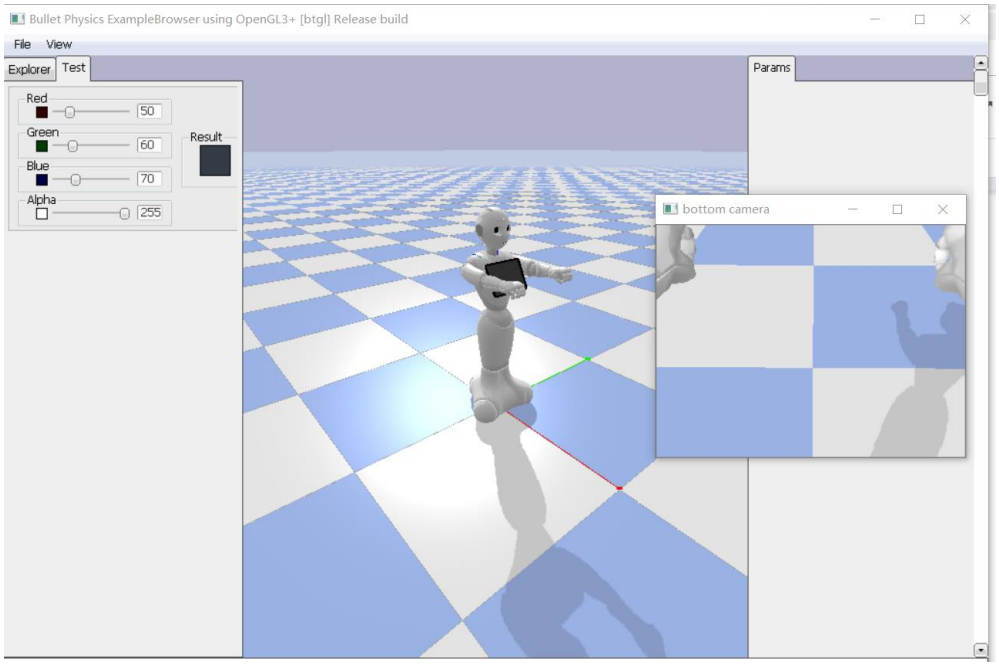


Figure 5. La fenêtre de la simulation

4.2 Promenade et évite des obstacles par l'algorithme *Braitenberg*

L'idée pour contrôler le déplacement du robot dans l'espace et éviter des obstacles en même temps c'est :

1. Permettre le robot de marcher dans la scène :
   pepper.moveTo(x,y,theta,frame=PepperVirtual.FRAME_ROBOT,_async=False)
   ## Cette méthode peut être appelée synchrone ou asynchrone. En mode asynchrone, la fonction peut être appelée lorsqu'elle est déjà lancée, cela mettra à jour l'objectif du mouvement
   pepper. move(x,y,theta) ## C'est appelée de manière asynchrone
   ## Les paramètres sont exprimés en m/s pour les vitesses de translation et en rad/s pour la vitesse de rotation.
2. Construire un environnement virtuel et mettre des obstacles dans la scène :

```
pybullet.setAdditionalSearchPath(pybullet_data.getDataPath())
  pybullet.loadURDF(
  "samurai.urdf",
  basePosition=[0, 0, -0.1],
  globalScaling=0.8,
  physicsClientId=client)
  pybullet.loadURDF(
  "sphere_small.urdf",
  basePosition=[2, 0, 0.0],
  globalScaling=10.0,
  physicsClientId=client)
  ……
```

On observe que quand le robot rencontre l'obstacle, il ne s'arrête pas. Donc lorsqu'il se déplace, il faut détecter en même temps les obstacles par les lasers :

```
pepper.subscribeLaser()
right_scan = pepper.getRightLaserValue()
front_scan = pepper.getFrontLaserValue()
left_scan = pepper.getLeftLaserValue()
```

Les variables right_scan, front_scan et left_scan retournent trois groupes de datas (Chaque groupe a 15 lasers au maximum). La configuration est présentée dans figure 6.

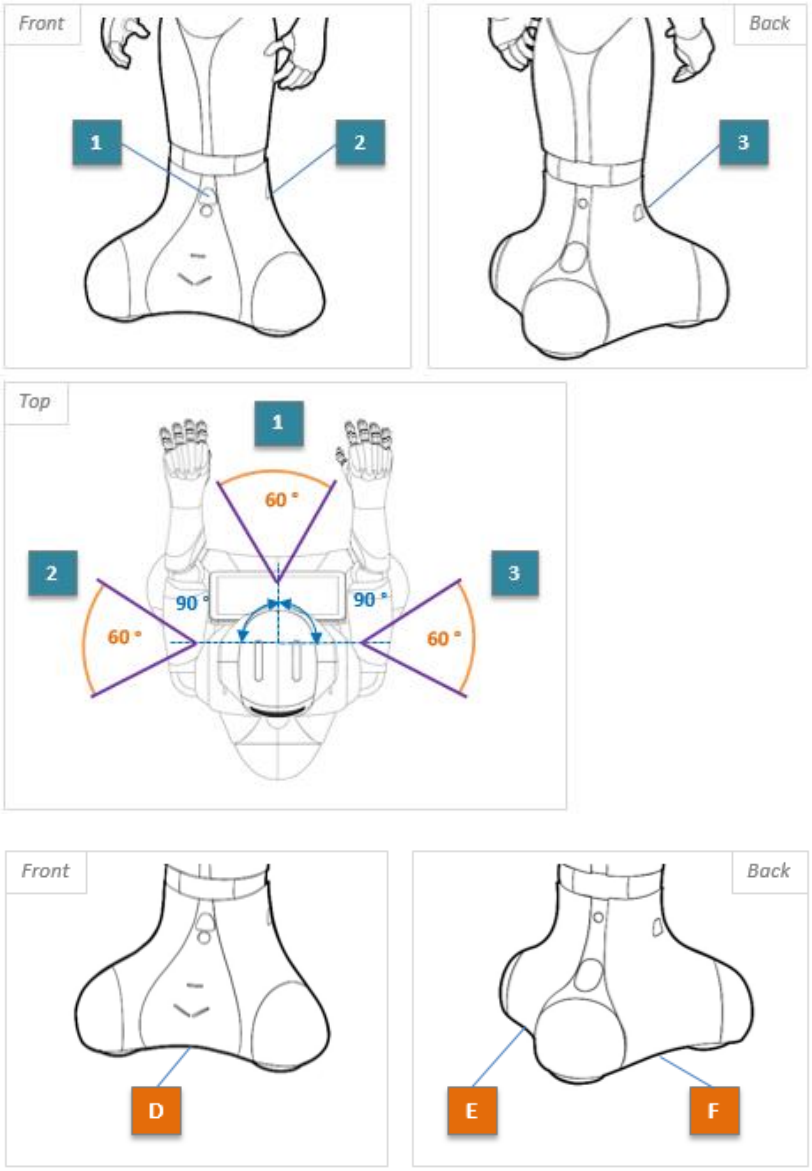
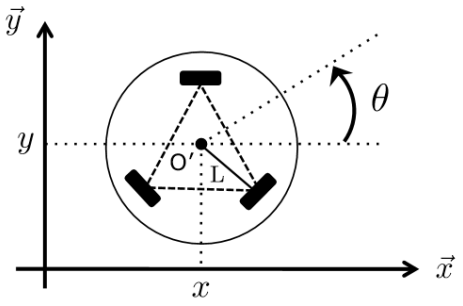


Figure 6. Les locations des capteurs lasers

1) Construire le modèle du *Braitenberg* pour calculer la vitesse du robot en se basant sur les données de distance de lasers.
   Pepper est un robot omnidirectionnel. On peut agir indépendamment sur les vitesses (de translation selon les axes x, y, et de rotation autour de z)



Modèle cinématique :

$$\begin{cases} \dot{x} = u_1 \\ \dot{y} = u_2 \\ \dot{\theta} = u_3 \end{cases}$$

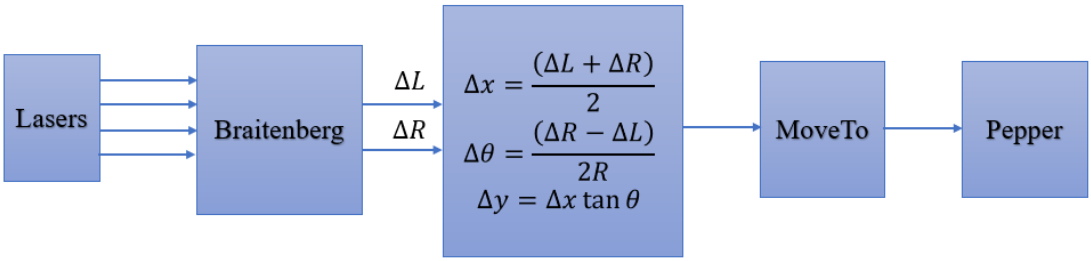
$u = (u_1\ u_2\ u_3)^T$:
Vecteur de commande du robot.

La sorte du *Braitenberg* c'est : $\begin{bmatrix} \Delta L \\ \Delta R \end{bmatrix}$.

On s'inspire du modèle d'un robot à 2 roues motrices indépendantes :

$$\begin{cases} \dot{x}_M = \frac{r}{2}(\dot{q}_L + \dot{q}_R)\cos\theta \\ \dot{y}_M = \frac{r}{2}(\dot{q}_L + \dot{q}_R)\sin\theta \\ \dot{\theta} = \frac{r}{2}(\dot{q}_L - \dot{q}_R) \end{cases}$$ (voir l'exemple du robot Pioneer).

Donc pour le Pepper, $\begin{cases} \Delta x = \frac{(\Delta L + \Delta R)}{2} \\ \Delta\theta = \frac{(\Delta R - \Delta L)}{2R} \\ \Delta y = \Delta x \tan\theta \end{cases}$ (R = L sur la figure ci-dessus). Et la fonction pepper.moveTo() utilise $\Delta x, \Delta y, \Delta\theta$. Nous pouvons obtenir le modèle ci-dessous :



2) Les codes [4] et la démonstration dans la figure 7 :

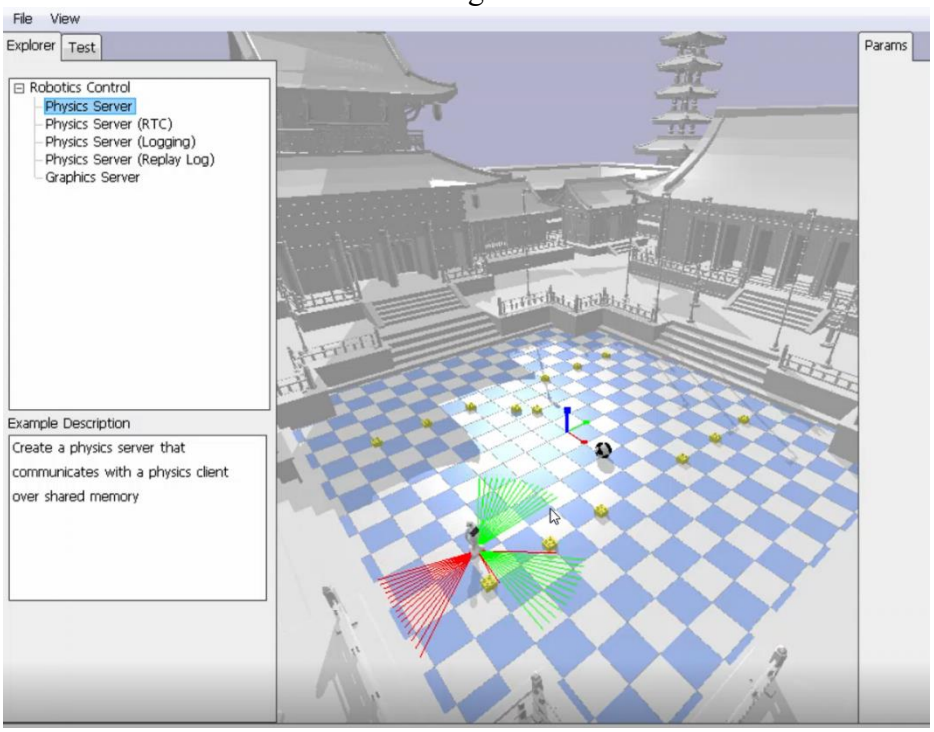


Figure 7. Une snapshot de la scène de simulation

3) L'analyse du résultat et la conclusion :

J'ai essayé d'augmenter le nombre de lasers (9 lasers, 27 lasers, 45 lasers) mesurés pour tester la capacité du robot à éviter les obstacles [4], le résultat de 27 lasers est mieux que celui de 9 lasers mais similaires à celui de 45 lasers.

Le robot Pepper marche bien dans cet environnement virtuel et il peut éviter des obstacles qu'on a posé dans le chemin. Mais il y a des limitations :

- Les lasers ne peuvent pas bien détecter les obstacles trop petits ou plus haut. Par exemple si je mets une table dans cet environnement, le laser du robot ne peut pas détecter les quatre pieds de la table (trop mince). Et la face de la table est haute, le laser ne peut pas le détecter, donc le robot sera trébuché par la table.
- Et le robot marche tout le temps tout droit, on ne peut pas encore lui orienter d'un point fixé jusqu'au point cible.

Bien que la fonction de simulation actuellement fournie par *qiBullet* ne soit pas aussi complète que le vrai robot, les avantages de l'utiliser pour des expériences de simulation sont évidents : la répétition itérative de tâches spécifiques peut présenter des risques pour les robots et certains environnements peuvent être difficiles à configurer. Des outils logiciels comme *qiBullet* permettent de simuler des environnements complexes et la dynamique des robots peuvent ainsi pallier ce problème, permettant d'effectuer les processus précités sur des modèles virtuels.

5 L'utilisation du *Naoqi*

5.1 L'introduction et l'installation du *Naoqi*

Naoqi est le nom du logiciel principal qui s'exécute sur le robot et le contrôle.

Le *Naoqi* est le Framework de programmation utilisée pour programmer Pepper ou NAO. Il répond aux besoins courants de la robotique : parallélisme, ressources, synchronisation, événements.

Ce Framework permet une communication homogène entre les différents modules (mouvement, audio, vidéo), une programmation homogène et un partage d'informations homogène.

Les avantages du *Naoqi* :

- Il est multi-plateformes, ce qui signifie qu'il est possible de développer avec lui sur Windows, Linux ou Mac.
- Il est multi-langage, avec une API identique pour C ++ et Python.
- Il fournit également une introspection, ce qui signifie que le Framework sait quelles fonctions sont disponibles dans les différents modules et où.

Les étapes de l'installation :

- **Python :**
- Assurons-nous de télécharger Python 2.7 - 32 bits.
- Nous pouvons trouver l'installateur ici : http://python.org/download/
- **Naoqi pour Python :**
- Récupérons et exécutons le programme d'installation :

pynaoqi-python-2.7-naoqi-x.x-win32.exe

- Nous pouvons télécharger la dernière version du site Web de la communauté *Aldebaran*.

5.2 Expériences du robot réel par python *Naoqi*

5.2.1 L'objectif et le planning

L'objectif : Si on veut que le robot quitte un point A pour aller à un point B, puis revenir au point A, il faut construire l'algorithme d'odométrie ou bien utiliser les modules de navigation de Pepper [5].

Utiliser AL navigation semble être plus efficace car le robot peut reconstruire une carte de son environnement et peut se localiser dans la carte [6]. Et il existe également des outils d'analyse du comportement des personnes [7] qui sont utiles pour ce projet.

Les étapes que je dois réaliser sont les suivantes :

1) Le robot doit suivre une trajectoire en forme de carré ou de rectangle, (par exemple 2m x 2m) qui est fixé dans le référentiel absolu.
2) Le robot doit éviter des obstacles fixes sur cette trajectoire sans perdre la trajectoire.
3) Puis il doit éviter des personnes qui traversent sa trajectoire.
4) Le robot doit reconnaître des humains.
5) Le robot peut faire des interactions avec humain aux points fixés.

5.2.2 L'implémentation des modules

La mise en œuvre de l'application proposée intègre différents modules spécifiques de Pepper tels que :

ALMemory - Fournit des données en direct de différents paramètres tels que la position, la température, la rigidité. Il permet également le stockage et la récupération des informations enregistrées ;

ALMotion - Fournit des méthodologies liées au mouvement du robot ;

ALNavigation - La fonctionnalité de navigation permet à Pepper d'apprendre des emplacements précédemment inconnus, de définir une carte, d'éviter les obstacles et de récupérer des points associés à la localisation sur la carte ;

ALTracker - Ce module permet au robot de suivre différentes cibles, telles que des points de repère et des visages, ou même une boule rouge. Ils peuvent être utilisés dans le but d'établir un pont entre la détection de cible et le mouvement du robot pour que le robot garde en vue la cible au milieu de la caméra ;

ALFaceDetection - Il s'agit d'un module de vision dans lequel le robot essaie de détecter, et éventuellement de reconnaître, les visages devant lui.

ALTextToSpeech - Ce module permet au robot de parler. Il envoie des commandes à un moteur de synthèse vocale et autorise également la personnalisation vocale. Le résultat de la synthèse est envoyé aux haut-parleurs du robot.

5.2.3 La réalisation du planning

Après d'avoir fait plusieurs expériences, j'ai réussi à contrôler le robot à suivre une trajectoire en forme de carré de taille 2 m x 2 m, qui est fixe dans le référentiel absolu. Et le robot peut éviter des obstacles fixes sur cette trajectoire sans perdre la trajectoire. S'il y a des humains qui marchent devant lui ou traversent sa trajectoire, le robot va arrêter à une une distance définie. Ensuite il continue rouler quand il a détecté que les humains sont partis. Le robot peut également faire des démonstrations aux points fixés : par exemple il commence la démo en disant une phrase « Bonjour, je m'appelle Pepper, je ferai des démonstrations ».

Ensuite, chaque fois il a atteint au point fixé, il va arrêter et effectuer les actions suivantes dans l'ordre : « danser », « serrer la main avec un humain », « changer le couleur des yeux », « chercher un humain et détecter son émotion ». Voir la démonstration dans Google drive [10].

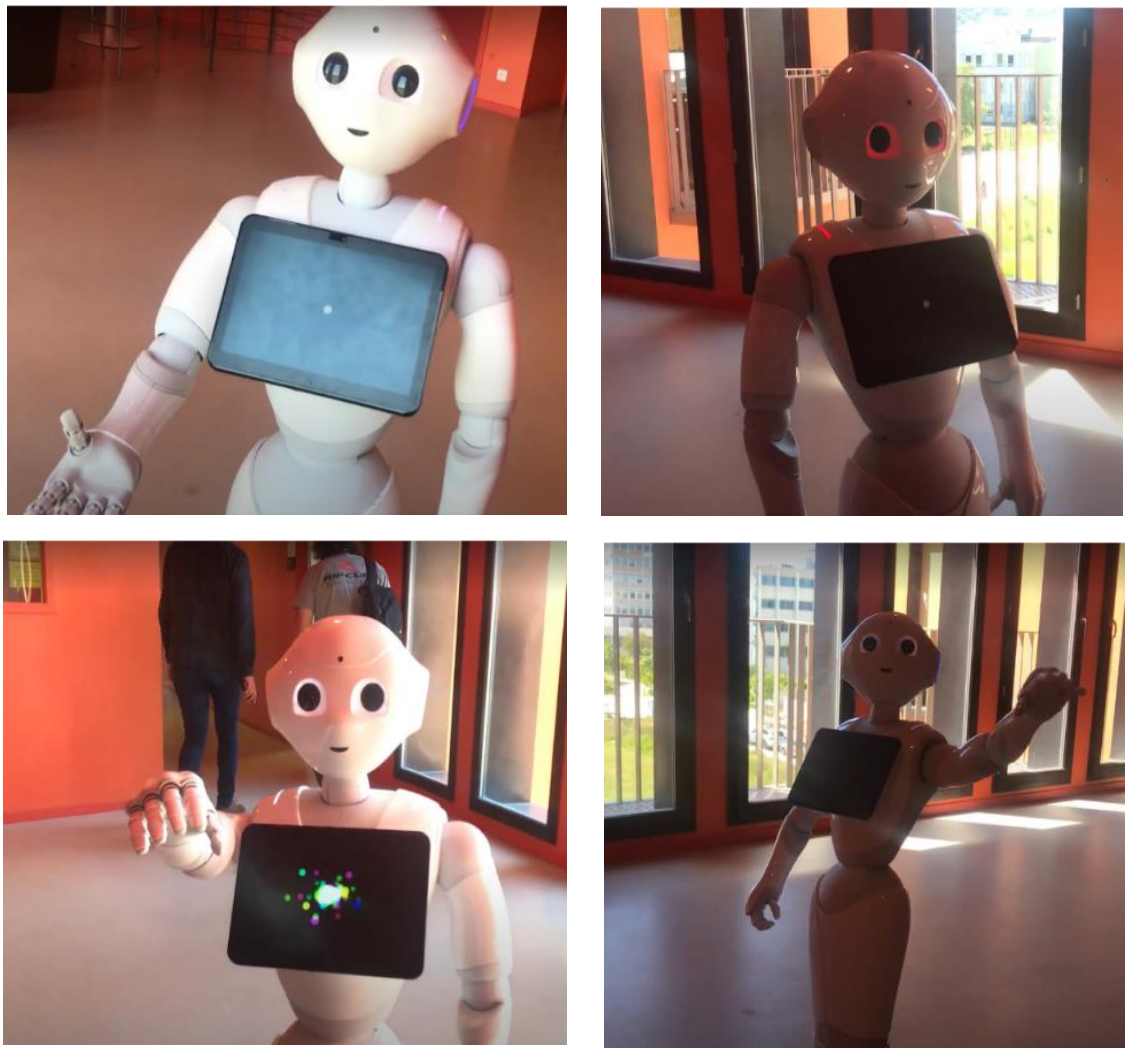


Figure 8. Des actions définies : « serrer la main avec un humain », « changer le couleur des yeux », « chercher un humain et détecter son émotion », « danser »

Les étapes à suivre pour la réalisation :

1) Premièrement faire l'exploration en utilisant *navigation.explore(rayon)* pour créer une carte et pouvoir utiliser des méthodes de navigation. Après l'exploration, le robot s'arrêtera à une position aléatoire.
2) Le rayon détermine la distance la plus éloignée que le robot peut explorer. Le robot peut entrer dans un couloir étroit, mais il ne peut pas aller trop loin si le rayon n'est pas assez grand. Il va donc se retourner et explorer d'autres endroits.
3) Ensuite, utiliser la fonction *navigation.navigateToInMap ([X, Y, Theta])* pour guider le robot vers un point fixe dans la carte créée. Donc quatre points fixés au minimum permettent au robot de suivre une trajectoire carrée.

Remarque : pour l'instant, Pepper n'utilise pas la cible *Theta* finale, mais uniquement la position *X* et *Y*. Pour l'orientation finale, on peut utiliser un *ALMotion.moveTo(X, Y, Theta)* pour l'angle souhaité.

Effectuer le mode exploration et une carte départementale qui est sortie après l'explorations

(voir plus de cartes sorties dans l'Annexe 2) :

Figure 9. L'environnement réel (à gauche) et une carte de la zone explorée (à droite).

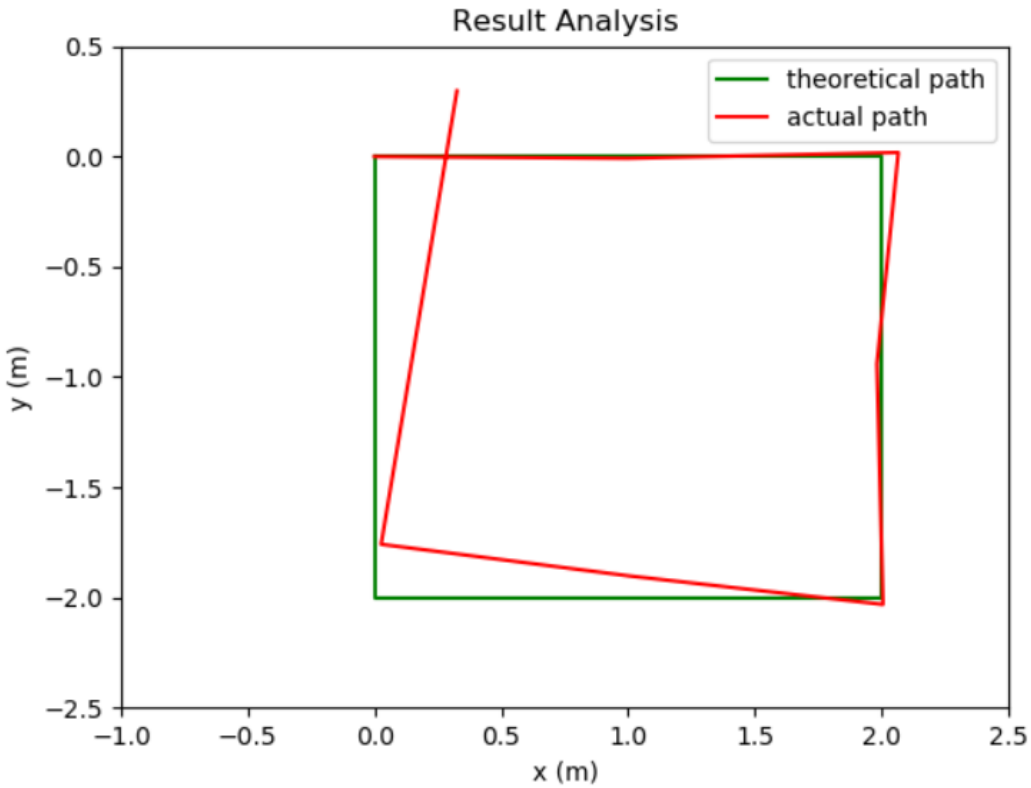


Figure 10. La trajectoire carrée.

Lorsque le robot se déplace du point A au point B, s'il y a un obstacle sur la route, le robot va éviter l'obstacle par recalculant sa trajectoire. Et il continuera atteindre le point cible s'il n'y a pas d'obstruction à l'emplacement cible. Voir la démonstration du mode d'exploration et la démonstration de navigation dans le Google drive [8] et les codes dans le lien Github[9].

6 Conclusions

Ce projet a permis de réaliser la navigation autonome de l'évitement automatique des obstacles et de l'interaction avec les humains pour les robots Pepper. Le robot peut éviter les collisions lors de la navigation, et il peut se déplacer selon un itinéraire spécifié en fonction de points cibles définis à l'avance. Au début du projet, *Choregraphe* a été utilisé pour apprendre et reconnaître les différents modules du robot et comment contrôler le mouvement du robot avec du code Python. À moyen terme en raison du confinement et de l'indisponibilité de vrais robots, j'ai mené les expériences en simulation en utilisant l'API *qiBullet*. Enfin, Python *Naoqi* a été utilisé pour programmer et combiner différents modules fonctionnels du robot Pepper pour contrôler et expérimenter avec le vrai robot. L'endroit pour les tests du robot est le couloir et l'espace libre du département informatique et le TechLab dans notre école.

Ce projet me permet d'apprendre beaucoup de choses et de faire bonnes expériences robotiques virtuelles et réelles, ce qui est très intéressant et franchement ce n'est pas facile.

Les difficultés que j'ai rencontrées c'est qu'il n'y a pas beaucoup de ressource à consulter, notamment le *qiBullet* que j'ai utilisé pendant le confinement, c'est un nouvel outil donc tout est en train de développer et mise à jour. Et le problème de version de python m'a empêché l'avancement du projet et m'a pris pas mal de temps pour trouver des solutions.

À l'avenir, nous pourrons étudier en profondeur d'autres fonctions du robot Pepper, telles que la capacité à reconnaître les images du visage, à distinguer s'il faut porter un masque, etc.

7 Remerciements

Un grand merci à mon encadrant du projet, Monsieur Patrick Hénaff pour ses conseils et son encouragement durant toute l'année.

Merci beaucoup à Monsieur Pascal Vaxivière pour ses aides dans TechLab afin de continuer avancer mon projet pendant la situation difficile.

8 Références

[1] Guide-Pepper : http://www.karsenti.ca/code/wp-content/uploads/2018/10/Guide-Pepper.pdf

[2] qiBullet : https://pypi.org/project/qibullet/

[3] qiBullet : https://www.groundai.com/project/qibullet-a-bullet-based-simulator-for-the-pepper-and-nao-robots/2

[4] Github : 3. qibullet\TestCodes\Avoid3.py https://github.com/LinxueLAI/Projet2A

Démo : https://drive.google.com/open?id=1oVTTbZtTSFXrr0PFqWzQQy2sZCoKePC0

Démo de 27 lasers (code : Avoid4.py) :

https://drive.google.com/open?id=19l-H16ZcXDsa03thT0zCDD72yt0YqJW0

Démo de 45 lasers (code : Avoid5.py) :

https://drive.google.com/open?id=1RBDS0VArpwuT-cz2571rjEjQ-loXjpnd

[5] http://doc.aldebaran.com/2-5/naoqi/motion/alnavigation.html

[6] http://doc.aldebaran.com/2-5/naoqi/motion/alnavigation-api.html

[7] http://doc.aldebaran.com/2-5/naoqi/peopleperception/index.html

[8] L’exploration :https://drive.google.com/open?id=1F4w0oxBIg7jvQzdnK2kGFJlht-DJtjck

Le test de navigation :

https://drive.google.com/open?id=1vt1Qp2Rl1sR_FBNisW-w1I4GfvZN1dw-

[9] https://github.com/LinxueLAI/Projet2A/tree/master/2.naoqi/testNavigation.py

[10] https://drive.google.com/open?id=16mUefF9Ae_MNPfUTuKWPG2MJDsX7YmsD

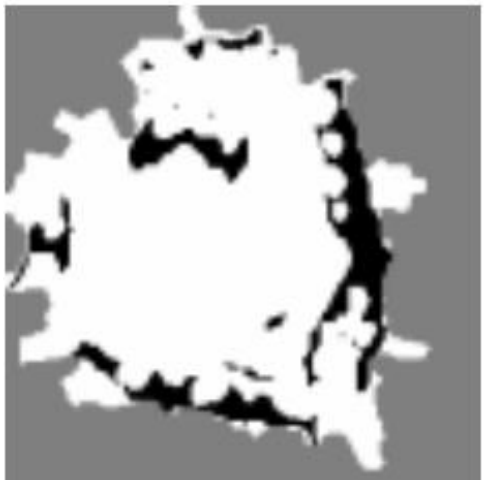
[11] site du TechLab : https://sites.google.com/depinfonancy.net/techlab/projets-2a/bim-viewer

9 Annexes

9.1 Annexe 1 - pepper_basic.py

```
#!/usr/bin/env python
# coding: utf-8

import cv2
import time
from qibullet import SimulationManager
from qibullet import PepperVirtual


def main():
    simulation_manager = SimulationManager()
    client = simulation_manager.launchSimulation(gui=True)
    pepper = simulation_manager.spawnPepper(client, spawn_ground_plane=True)

    pepper.goToPosture("Crouch", 0.6)
    time.sleep(1)
    pepper.goToPosture("Stand", 0.6)
    time.sleep(1)
    pepper.goToPosture("StandZero", 0.6)
    time.sleep(1)
    pepper.subscribeCamera(PepperVirtual.ID_CAMERA_BOTTOM)

    while True:
        img = pepper.getCameraFrame()
        cv2.imshow("bottom camera", img)
        cv2.waitKey(1)


if __name__ == "__main__":
    main()
```

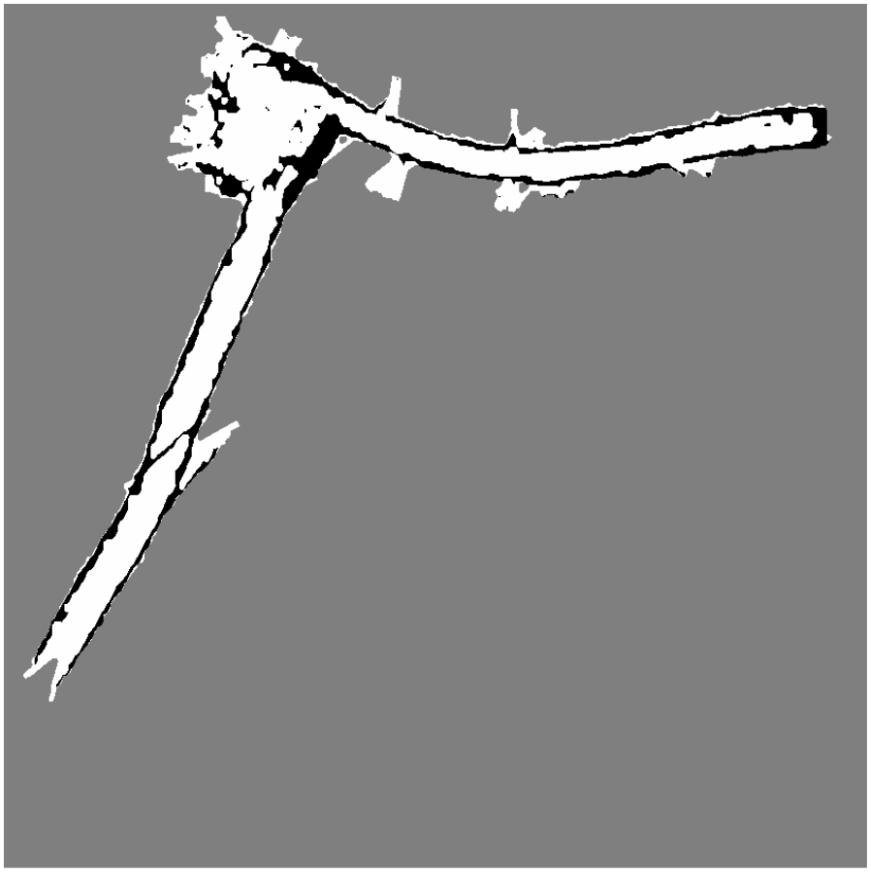
9.2 Annexe 2 – les cartes sortie après l'exploration.



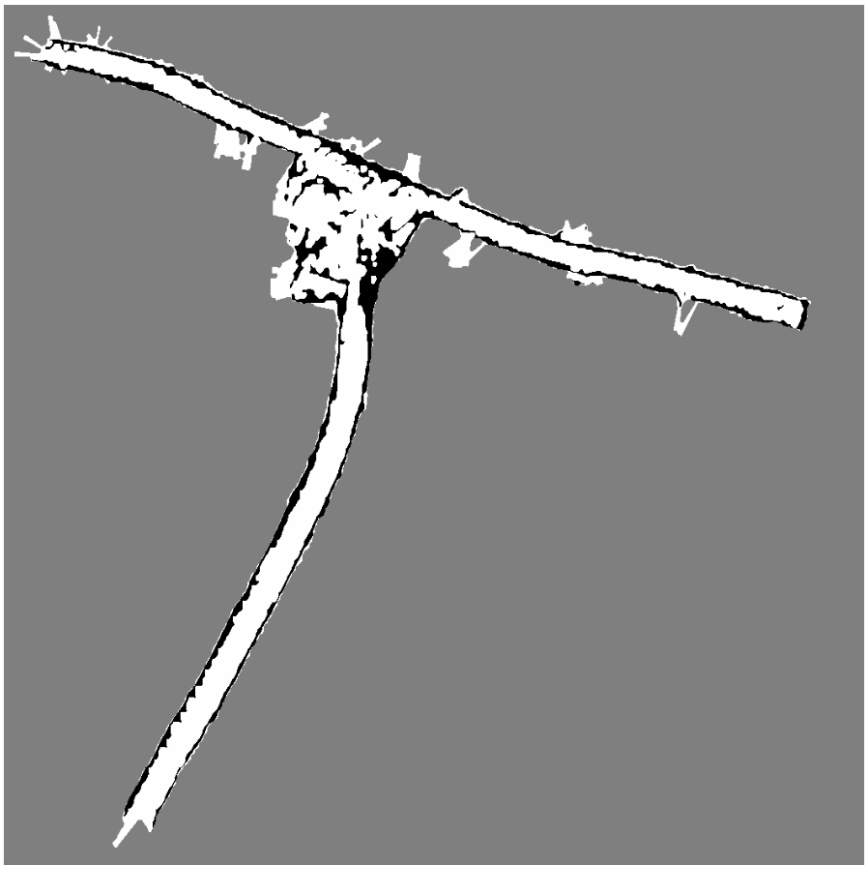
Rayon = 2.5 m

Rayon = 3.0 m

Rayon = 20 m



Rayon = 30 m



Rayon = 40 m