28/07/2021
BETAILLE
Antonin

Compilation des comptes-rendus hebdomadaire

Projet : Bras manipulateur

SECTION : Turtlebot

Table des matières

Déploiement du TurtleBot

Mise à niveau du projet

Premier objectif : Comprendre et remettre en place le dispositif développé par Killian Rolland

Travail préliminaire :

- Lecture des travaux réalisés par Killian Rolland

- Compréhension générale du principe de fonctionnement

- Suivi du « Document de Déploiement »

Tentative de fonctionnement hybride PC Noetic/ Turtlebot Kinetic

Mise en place des tests :

- Connexion de la Raspberry du TurtleBot sur la livebox-8d62_5GHz à l'aide d'un module d'écran de Raspberry

- Mise en place du setup permettant d'utiliser ROS sur la raspberry via SSH

- Lancement du node de SLAM et cartographie du rez de chaussée à l'aide du node de Téléopération

- Tentative d'utilisation du node de Navigation

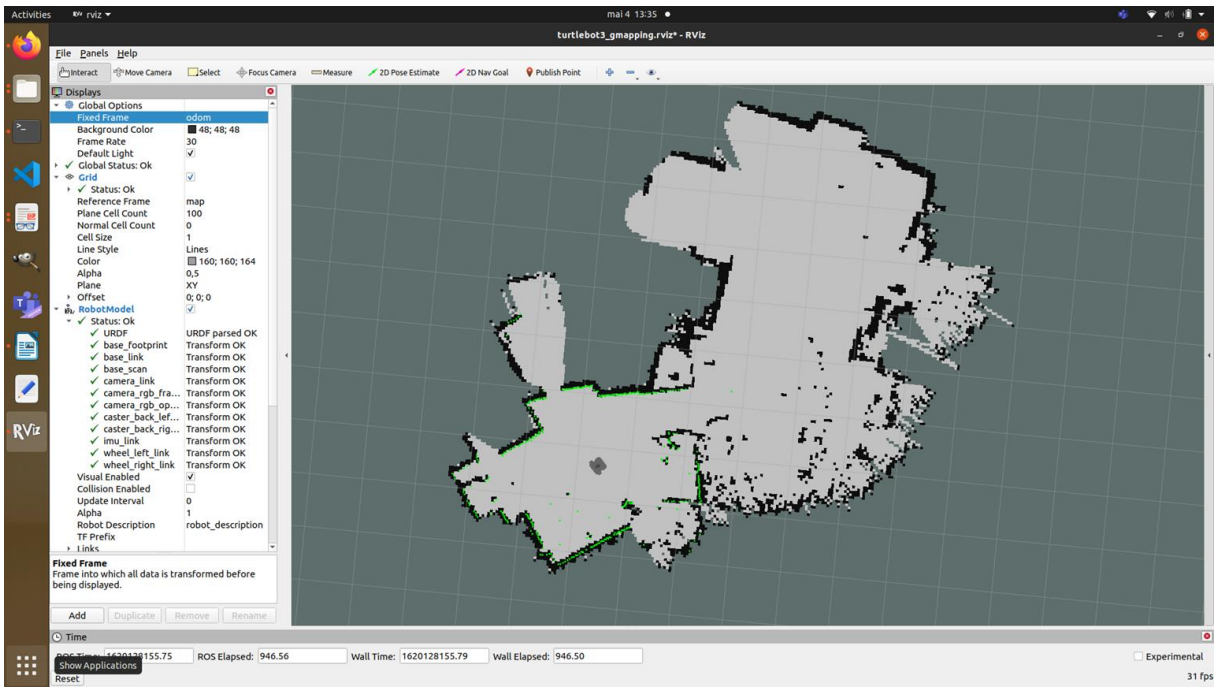

Figure 1: Affichage de l'appartement sous Rviz (Noetic) après SLAM

Node de Navigation

Le node de Navigation ayant été modifié pour permettre de ne prendre en compte que des positions établies au préalable, la navigation via RVIZ n'était plus fonctionnelle.

Par conséquent j'ai donné les positions correspondant aux points d'intervention « cuisine » et « chambre » et j'ai modifié le fichier turtle.sh pour qu'il corresponde à ma configuration.

La dernière étape consistant à lancer la page web permettant de déplacer le robot n'a pas fonctionné. Les causes éventuelles de cet échec sont l'absence du module de caméra et une erreur MD5sum liée à la distribution de ROS n'étant pas la même sur le master (noetic) et la raspberry (kinetic). L'option envisagée est de réinstaller ROS sur la Raspberry.

Test interface web :

- Lancement de la page web fonctionnel

- La vidéo, le déplacement par point cible et l'affichage de la vraie valeur de la batterie ne sont pas fonctionnels

- Le déplacement par joystick fonctionne

- Le package ros-noetic-web-video-server n'existe pas, donc la vidéo ne peut pas être fonctionnelle si le rosmaster est sur ubuntu 20.04

Changement de distribution PC melodic / Turtlebot melodic

Le package nécessaire à l'affichage de la vidéo n'existant que sous la version kinetic ou melodic de ROS, il a fallu changer d'ordinateur.

J'ai récupéré un ordinateur de la Chaire M@D équipé du logiciel Oracle MV VirtualBox afin de mettre en place une machine virtuelle. Mon objectif était de revenir à Ubuntu 18.04 pour avoir une version relativement récente de la distribution linux. Or la version 18.04 doit fonctionner avec ROS melodic. Ce qui n'était pas compatible avec la version kinetic de la raspberry pi. J'ai donc fait le choix de réinstaller ROS melodic sur la raspberry pi.

Problème rencontré :

Malgré le fait que la raspberrry soit connecté à la livebox avec une adresse ip et que le ssh était possible, je ne pouvais pas mettre à jour la raspberry. En effet, le réseau était inaccessible. Le problème se réglait en modifiant le fichier etc/network/interfaces afin de définir la passerelle 192.168.1.1

Mise en place de la machine virtuelle :

- Installation de la machine virtuelle et paramétrage

- Installation de ROS melodic

- Définition de l'adresse IP et de la passerelle pour permettre la communication entre la raspberry et la machine virtuelle

→ voir options réseaux et opt/network/interfaces

Installation de ROS :

- installation de ROS melodic sur la raspberry et installation des packages du turtlebot

→ Finalement il faut reflasher la carte SD pour installer la version Ubuntu 18.04 LTS (Bionic Beaver) car Raspian 9 est trop ancien pour supporter l'installation de ROS Melodic et des packages du turtlebot3. La machine virtuelle ne prenant pas en charge la lecture de la carte SD.

Fonctionnement de la vidéo embarquée :

- Mise en place de raspicam_node

→ Package installé depuis la source car destiné à kinetic. Mais problème de librairie insolvable avec Ubuntu 18.04 : mmal libraries

https://githubmemory.com/repo/UbiquityRobotics/raspicam_node/issues/105

Solution envisagée : utiliser usb_cam node à la place

Tentative d'adaptation du node raspicam :

- Tentative d'installation de mmal depuis une solution proposée sur ROS Answers:

https://answers.ros.org/question/356280/raspberry-pi-cam-node-for-melodic-ubuntu-1804-server/

L'installation a fonctionné, la compilation aussi.

Autres actions réalisées:

- redirection vers les nouvelles bibliothèques avec :

https://www.raspberrypi.org/forums/viewtopic.php?t=51925

- fixation des erreurs de permission de dev/vchiq/ (sudo chmod 777 dev/vchiq)

→ Cependant au final la caméra ne se lance pas :

mmal : Failed to create camera_info component

- Tentative de réparation du fichier turtle.sh qui lance la totalité du projet.

Problématique : comment effectuer un roslaunch dans un .sh depuis une connexion SSH ?

→ Utilisation de env-loader recommandée mais besoin de recherches

Conclusion sur la viabilité de la version ROS melodic :

Impossibilité d'utiliser le module de caméra de la raspberry. Il faudrait obligatoirement utiliser une caméra USB en utilisant le node associé. Le temps d'investissement est trop long pour cela.

J'ai créé une nouvelle machine virtuelle utilisant Ubuntu 16.04 pour installer ROS kinetic et ne plus avoir de problèmes avec les bibliothèques. J'ai également réinitialisé la carte SD de la raspberry afin de revenir à la configuration de départ.

Je garde cependant la machine virtuelle utilisant Ubuntu 18.04 afin de travailler avec le bras Niryo si besoin

Changement de distribution PC kinetic / Turtlebot kinetic

Nouvelle machine virtuelle :

Installation de la nouvelle machine virtuelle sous Ubuntu 16.04.

Installation de ROS kinetic via le quick start de Robotis

Mise en place de ROS kinetic :

- installation fonctionnelle de ROS kinetic sur la raspberry et installation des packages du turtlebot3 (E-manual Robotis section kinetic)

- nodes de SLAM, teleopération fonctionnels. Cependant la présence des baies vitrées détériore le processus de création de la carte pendant le SLAM.

Tests de la nouvelle configuration :

- Création de la carte grâce au node de SLAM pendant la matinée, quand la luminosité est encore assez faible

- Test du node de navigation difficile

→ PROBLÈME : Inversion de la direction du nuage de points du LIDAR par rapport à la carte sur rviz (problème réglé, le LIDAR était monté à l'envers)

- Node de navigation testé et fonctionnel

- Toujours des problèmes avec turtle.sh et la commande roslaunch.

Test de l'interface web satisfaisant

En lançant les nodes manuellement, sans passer par turtle.sh, j'ai finalement pu tester l'interface web. La caméra est affichée, ainsi que le niveau de batterie, le joystick fonctionne. Les positions enregistrées sont disponibles et le robot peut les rejoindre

Résolution d'un problème lié à Nodejs pour le lancement de live-server:

```
$ sudo npm cache clean -f

$ sudo npm install -g n

$ sudo n stable
```

Avec le script turtle.sh mis à jour, le fonctionnement de l'interface web est toujours aussi satisfaisant

Test de la précision du node de navigation

Précision assez peu satisfaisante pour l'application que nous voulons.

En indiquant une position et une orientation sur Rviz, le robot ne s'oriente pas parfaitement seulon l'orientation voulue.

De plus le calcul de trajectoire se réactualisant sans arrêt, le robot se repositionne par à-coups successifs. Cela posera des problèmes dans le cas du transport d'un récipient rempli de liquide.

La modification des paramètres du package de navigation, qui gèrent l'erreur acceptable, la vitesse et l'accélération, permettra peut-être d'améliorer les problèmes précédents.

Amélioration de l'activation de l'interface web

Le fichier turtle.sh utilisé dans le processus de déploiement du turtlebot n'était pas fonctionnel.

En effet, la commande :

gnome-terminal -x sshpass -p 'turtlebot' ssh pi@192.168.1.128 "roslaunch turtlebot3_bringup turtlebot3_robot.launch; bash;"

permettant de se connecter en ssh au turtlebot pour l'activer, retournait une erreur du type :

roslaunch : command not found

car il semblerait qu'en procédant de la sorte (sshpass + commande liée à ROS), le fichier .bashrc n'est pas pris en compte. Il faut donc écrire ces lignes de commande entre guillemets :

cd catkin_ws;

source /opt/ros/kinetic/setup.bash;

export ROS_MASTER_URI=http://192.168.1.122:11311;

export ROS_HOSTNAME=192.168.1.128;

source devel/setup.bash;

roslaunch turtlebot3_bringup turtlebot3_robot.launch;

Il faut ensuite utiliser le même procédé pour activer le node de caméra :

gnome-terminal -x sshpass -p 'turtlebot' ssh -t pi@192.168.1.128 "cd catkin_ws;

source /opt/ros/kinetic/setup.bash;

export ROS_MASTER_URI=http://192.168.1.122:11311;

export ROS_HOSTNAME=192.168.1.128;

source devel/setup.bash;

roslaunch turtlebot3_bringup turtlebot3_rpicamera.launch;"

Ainsi, il est possible d'activer le turtlebot de façon à lancer les différents nodes nécessaire au lancement de l'interface web

Raccourcis et informations disponibles dans les terminaux

Jusque-là je n'y avais pas prêté attention mais le .bashrc de la machine virtuelle et celui de la raspberry du turtlebot sont prévus pour accepter la création d' « alias ».

Ces raccourcis permettent d'exécuter une ou plusieurs commandes en utilisant un mot clé.

La syntaxe pour créer un alias est la suivante :

alias nom_alias='commande1 & commande2 & … & commande_finale'

Par exemple : eb et sb, déjà présent par défaut, permettent respectivement de modifier le .bashrc (nano ~/.bashrc) puis de le réactualiser pour prendre en compte les modifications (source ~/.bashrc).

Pour faciliter les modifications ou simplement lancer certaines fonctionnalités plus rapidement j'ai ajouté les alias suivants :

Sur la machine virtuelle :

```
alias tbot='sshpass -p 'turtlebot' ssh pi@192.168.1.128'
```
→ connexion directe au turtlebot via ssh

```
alias tsh='cd ~/catkin_ws/src/turtlebot3 && ./turtle.sh'
```
→ Lancement de l'interface web via l'exécution de turtle.sh

```
alias tkeys='roslaunch turtlebot3_teleop turtlebot3_teleop_key.launch'
```
→ Lancement du node de commande du turtlebot au clavier

```
alias tstart='roscore & sleep 5 & gnome-terminal -x sshpass -p 'turtlebot' ssh -t
pi@192.168.1.128 "cd catkin_ws;
source /opt/ros/kinetic/setup.bash;
export ROS_MASTER_URI=http://192.168.1.122:11311;
export ROS_HOSTNAME=192.168.1.128;
source devel/setup.bash;
roslaunch turtlebot3_bringup turtlebot3_robot.launch;
" & gnome-terminal'
```
→ Activation du turtlebot pour les différents nodes de ROS puis ouverture d'un autre terminal

```
alias tpose='rostopic echo /amcl_pose'
```
→ Donne la position du turtlebot dans rviz

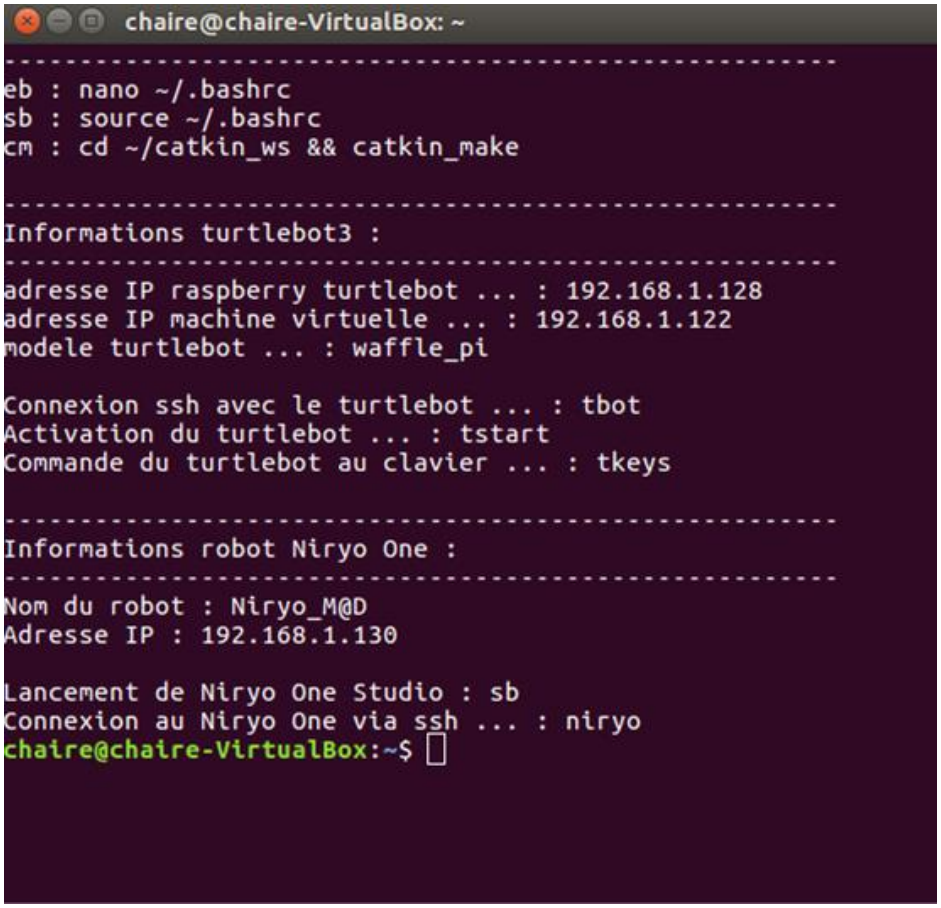

Illustration 2: Message informatif au lancement du terminal de la machine virtuelle (a sûrement évolué depuis le 27/05/21)

alias a_u='rosrun emergency_stop emergency_stop'

→ Lancement de l'arrêt d'urgence

~~alias tactiv='rosrun emergency_stop emergency_activation'~~

→ Lancement de l'activation après l'arrêt d'urgence (utilisation avec le bouton d'activation à mettre en place)

→ alias supprimé

alias tros_h='sshpass -p 'turtlebot' ssh -t pi@192.168.43.242 "cd catkin_ws;
source /opt/ros/kinetic/setup.bash;
export ROS_MASTER_URI=http://192.168.43.242:11311;
export ROS_HOSTNAME=192.168.43.242;
source devel/setup.bash;

roscore;"'
→ Lancement de roscore sur la raspberry du turtlebot lorsqu'elle est ROS_MASTER (ajout de l'adresse ip dans /etc/hosts comme pour le niryo)

alias tsh='cd ~/catkin_ws/src/turtlebot3 && ./turtle.sh'
→ Lancement de l'interface web via l'exécution de turtle.sh

alias a_u='rosrun emergency_stop_for_tests emergency_stop_for_tests'
→ Lancement de l'arrêt d'urgence pour les tests (cet alias ne fonctionne plus comme le précédent du même nom)

alias tdock='cd ~/ && ./docking.sh'
→ Lance le docking plus facilement

Sur la raspberry pi du turtlebot :

alias bringup='roslaunch turtlebot3_bringup turtlebot3_robot.launch'
→ Activation du turtlebot

alias cam='roslaunch turtlebot3_bringup turtlebot3_rpicamera.launch'
→ Activation de la caméra de la raspberry

Mise en place du docking

Suivi du processus d'installation expliqué dans le manuel de déploiement du turtle bot en lien avec le site de Robotis :
https://emanual.robotis.com/docs/en/platform/turtlebot3/basic_examples/#automatic-parking-vision

Ainsi nous pouvons utiliser la fonctionnalité : « Automatic parking vision »

Note importante :
Print out AR marker 17 which is the most right bottom one from the link. The marker should be as big as the one in this instruction video (8cm x 8cm)
→ **Il y a donc un sens à respecter pour disposer le marqueur**

Raccourci : Écriture d'un script **docking.sh** permettant de regrouper toutes les instructions utilisées pour démarrer le docking.

Remarque : La précision de positionnement du docking n'est pas parfaite. Le turtlebot n'est pas exactement dans l'axe du marqueur. Cependant le marqueur imprimé est un peu plus petit que prévu (il faut tester avec les bonnes dimensions)
D'autre part, l'erreur d'orientation est minime

Idée de correctif : Si la fonction de docking le permet, il pourrait être intéressant de mettre un objet contraignant le déplacement du turtlebot vers l'avant. Ainsi, lorsqu'il se rapproche du marqueur pendant la phase finale de la fonction, il pourrait être guidé de façon à arriver exactement avec la bonne orientation (un peu à la façon d'un entonnoir)
→ Ce n'est pas la piste qui sera explorée pour la suite du projet

Écriture d'un script d'urgence pour stopper le turtlebot

Création du fichier **stop.sh** permettant de se reconnecter au robot pour lancer le mode de contrôle du robot au clavier. Ainsi nous pouvons reprendre le dessus sur les commandes de mouvement.
→ Il est possible que ce fichier n'existe plus

Tests de précision du docking

Les démonstrations faites pour les élèves de CM2 ont mis en évidence le fait que le robot ne s'arrête pas en fin de docking si le marqueur est situé un peu trop haut par rapport à la caméra. Il se rapproche proche du marqueur mais continue d'avancer alors qu'il a déjà percuté le mur.

Il faudrait donc estimer quelle est la position idéale pour positionner ce marqueur.

→ En positionnant exactement le centre du marqueur au niveau de la caméra le docking est efficace

Travail sur le docking

L'avantage du docking dans la navigation :

Puisque la navigation par SLAM n'assure pas un positionnement totalement fiable il m'a semblé que la solution de sécurité devait s'orienter vers une généralisation du docking pour chaque commande. En effet bien que le docking ne permette pas au turtlebot de s'orienter parfaitement face au marqueur, il permettait au moins de se rapprocher de l'objectif avec toujours la caméra orientée en direction du centre du marqueur. Au final, j'ai utilisé cette dernière observation pour palier à la première.

Finition du docking :

Dans le fonctionnement du docking, tout est découpé en séquences. Les dernières consistent à se rapprocher du marqueur et à vérifier que le robot est bien orienté en direction du centre du marqueur avant d'arrêter les moteurs. J'ai donc intercalé une action supplémentaire avant la séquence d'arrêt. J'ai demandé au turtlebot d'avancer encore quelques instants pour venir au contact du mur. De cette façon il est capable de se positionner tout seul parallèlement au marqueur, toujours à la même position. Pour faciliter cette manœuvre j'ai récupéré l'angle du robot par rapport au marqueur pour estimer si le robot vient de la droite ou de la gauche et ainsi faire accélérer soit la roue droite soit la gauche pour faire le virage

Ajout d'un pare-choc à l'avant du robot :

En découvrant que le robot pouvait se repositionner seul au contact du mur j'ai remarqué que c'est la caméra qui venait taper le mur en premier. Or, en plus d'être dangereux pour la caméra, cela ne permettait pas de positionner le turtlebot de façon optimale. J'ai donc prévu de rajouter un pare-choc devant le robot pour que ce soit lui qui touche le mur en premier. Ce pare-choc devait être pensé de façon à accompagner le virage. Pour plus de sécurité, il serait préférable de lui ajouter un revêtement en mousse pour amortir le choc

Mise en pratique :

Actuellement, le pare-choc est une barre en mousse solide utilisée initialement pour protéger un colis. Je l'ai scotchée à l'avant du turtlebot, sous la caméra.
Les tests réalisés avec ce prototype sont déjà très satisfaisants. Il fallait ensuite que j'estime à quel point le pare-choc était fiable.

Réalisation du pare-chocs

CAO :

Le pare-chocs a été pensé pour être réalisé par impression 3D. La conception a été effectuée sur Inventor Pro 2020. 4 trous ont été prévus pour qu'il se fixe sur la première plateforme du

châssis du turtlebot, à l'avant, sous la caméra embarquée. L'avant du pare-chocs dépasse suffisamment devant la caméra pour que celle-ci ne touche pas le mur pendant le docking. Il est prévu pour être léger mais suffisament résistant.

Pour un aspect esthétique plus sympathique, j'ai ajouté le logo de la chaire.

Impression 3D :

Seul un prototype a été imprimé sur l'imprimante Zmorph VX. Avec le logiciel Voxelizer, l'impression a été configurée avec une structure alvéolée avec un remplissage 20 %. L'impression a duré environ 4h et n'a pas été parfaitement réalisée, car le haut de la partie avant du pare-chocs s'est un peu affaissée. Cela peut être dû au soleil qui venait taper directement sur la pièce pendant l'impression. De plus, il est important de noter que la visserie qui était à ma disposition ne permettait pas de fixe le pare-choc au turtlebot. Je n'ai donc pas pu tester son efficacité en conditions réelles.

Le bouton STOP :

Quand j'ai pris le turtlebot en main avec l'interface web j'ai été très vite confronté à un problème assez contraignant : **l'impossibilité de stopper le robot alors qu'une commande de navigation était en cours**. La manipulation consistait alors à quitter tous les processus en cours et reprendre la main en activant la navigation par clavier pour arrêter les moteurs. L'utilisateur de la base n'étant pas censé pouvoir réaliser ces actions, il est plus logique et plus simple d'avoir à disposition un bouton d'arrêt disponible depuis l'interface web.

Ainsi, il a fallu se pencher sur le package **simple_navigation_goals** implémenté par Killian. J'ai donc relu attentivement le fichier **simple_navigation_goals.cpp** dans le dossier **src** afin de comprendre ce qu'il se passait à l'activation d'un bouton.

Il s'est avéré que conformément au template proposé sur ROS.org :
http://wiki.ros.org/actionlib_tutorials/Tutorials/SimpleActionClient

L'envoi d'une commande est toujours suivi par la commande ac.waitForResult()

https://docs.ros.org/en/diamondback/api/actionlib/html/classactionlib_1_1SimpleActionClient.html#af19db907b90a2f1f810592ea597fb7b1

Cette commande a pour effet de bloquer toute autre action en attendant la complétion de la commande. Il était donc impossible d'intégrer un bouton stop. J'ai donc supprimé cette commande pour permettre de récupérer l'information des clics en continu. J'ai cependant ajouté une fonction de callback à sendGoal() pour avoir une action de flag lorsque la commande de navigation arrive à son terme :

```
ac.sendGoal(goal, &doneCb);
```

Documentation SimpleActionClient associée:

https://docs.ros.org/en/diamondback/api/actionlib/html/classactionlib_1_1SimpleActionClient.html#af19db907b90a2f1f810592ea597fb7b1

→ actionlib::SimpleActionClient< ActionSpec > Class Template Reference

http://docs.ros.org/en/diamondback/api/move_base_msgs/html/namespacemove__base__msgs.html#a19196856ac9dfad7a6f94a92023b58a3

→ move_base_msgs Namespace Reference

Le flag permet d'activer la publication des messages en fonction de la réussite de la commande.

Désormais, toutes les commandes peuvent être interrompues, sauf le docking qui ne dépend pas du même programme.

Remarque : pour une meilleure lisibilité et une réduction de la taille du programme, il faudrait créer plus de fonctions

Bouton d'arrêt d'urgence

Nécessité d'un bouton d'arrêt d'urgence

Bien que la plupart des fonctionnalités de l'interface web aient été mises en place. Un cas de figure concernant le docking m'a forcé à reconsidérer la capacité de l'utilisateur à contrôler le robot en cas d'urgence. En effet, même si j'ai mis un bouton stop en place, ce bouton n'est actif que sur le node /simple_navigation_goals. Il n'a donc pas d'effet sur le node /automatic_parking_vision, ou quelconque autre node agissant sur le topic /cmd_vel. Donc

dans le cas où certains nodes mèneraient à des bugs où le robot reste bloqué sur une commande, il me fallait avoir accès à un bouton d'urgence.

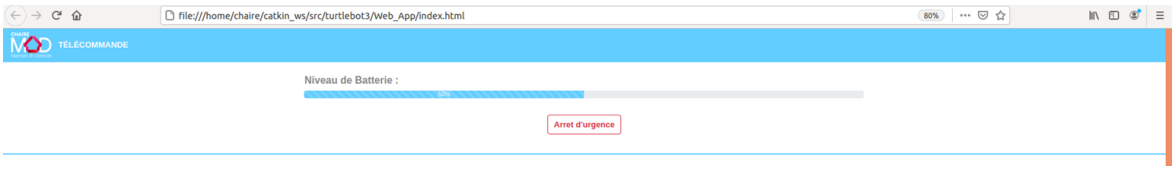

Illustration 1: Bouton d'arrêt d'urgence

Principe d'action

Ce bouton d'urgence, a principalement pour but de reprendre la main sur les commandes du robot. Il faut donc qu'il soit à la fois en mesure de tuer les nodes envoyant les commandes et d'arrêter le robot.

Pour cela j'ai créé le node /emergency_stop. Ce node a pour rôle de tuer les nodes, de couper les moteurs en envoyant « false » sur le topic /motor_power, puis d'envoyer une commande nulle sur /cmd_vel. Sans cette dernière commande, le robot était en effet arrêté le temps que le robot soit réactivé mais une fois qu'il était remis en état de marche, la commande problématique reprenait.

Difficultés et résolution

Ma première idée était de tuer tous les nodes sans distinction pour remettre les choses à plat avant de relancer l'intégralité de l'architecture ROS. Cependant je me suis confronté à un problème que je n'avais pas soupçonné. Sans quitter manuellement les terminaux utilisés pour activer les nodes et l'interface web, je ne pouvais pas remettre le robot en état de marche. Même en utilisant **xdotool** et la commande **windowkill** il m'était impossible de cibler certains terminaux à fermer. Et pour cause, cette commande, même combinée à des commandes de sélection de terminaux fermait tous les terminaux sans distinctions, m'empêchant ainsi de relancer un terminal sans intervention au clavier. Il n'était pas non plus possible d'utiliser une commande du type `xdotool key ctrl+d` car cela n'avait pas d'effet

J'ai donc finalement pris la décision de n'éliminer que les nodes ayant un effet sur /cmd_vel. C'est-à-dire les nodes cités plus haut.

Réactivation des nodes

Une fois que les nodes sont arrêtés il me faut être en mesure de rétablir les fonctionnalités du turtlebot. Pour cela j'ai créé un autre node activé par /emergency_stop. Je l'ai appelé

/emergency_activation. La logique voudrait que je mette en place un bouton d'activation pour éviter que le robot ne revienne se placer dans un cas de figure où il est amené à bugger (pour le docking notamment). Mais dans un souci de rapidité ce node est activé à la fin de l'utilisation de /emergency_stop.

Le but de ce node est de lister les autres nodes restés actifs pour déterminer quelle fonctionnalité a pu être lancée. Par exemple, si /ar_track_alvar est actif, cela signifie que /automatic_parking_vision vient d'être tué. Il faut donc le relancer pour que la fonctionnalité soit de nouveau disponible

Remarques

L'arrêt d'urgence est amené à évoluer si d'autres nodes ont la permission de publier sur /cmd_vel.

Il pourrait également être judicieux de rajouter le bouton « activation » pour que l'application de docking ne se relance pas automatiquement (ou alors il faut supprimer d'autres nodes supplémentaire et trouver d'autres témoins pour remarquer quelle application a été lancée)

Nécessité de séparer les deux fonctionnalités de l'arrêt d'urgence

Plus haut, j'expliquais avoir mis un bouton d'arrêt d'urgence me permettant de tuer les nodes agissant sur le topic **/cmd_vel** puis de les réactiver au bout de quelques instants. Ce dispositif avait pour avantage d'être rapide mais assez contre intuitif concernant le but premier de l'arrêt d'urgence. Le docking par exemple se contentait de reprendre depuis le début et faisait tourner le robot dès sa réactivation. Alors certes, le node **/emergency_stop** pouvait être utilisé à la fois depuis l'interface web et depuis un terminal mais dans le cadre d'une utilisation normale, il était essentiel de pouvoir activer le node **/emergency_activation** à n'importe quel instant, en fonction de la volonté de l'utilisateur.

C'est pourquoi j'ai créé un bouton [Réactivation] assurant le rôle de remise en fonctionnement du robot.

Modifications de l'architecture

Afin d'effectuer les changements induits par le choix de réactivation passant par le bouton d'arrêt d'urgence L'architecture de l'arrêt d'urgence était dépendante du node **/simple_navigation_goals** qui recevait les messages provenant des boutons de l'interface web. Or, ce node étant détruit à l'issue de la première phase de l'arrêt d'urgence, il fallait trouver un autre moyen de déclencher la réactivation.

De plus, une fois les moteurs coupés, le joystick devenait inutilisable. Ce qui n'est pas le plus pratique lorsque l'arrêt d'urgence a été déclenché à cause d'une mauvaise trajectoire nécessitant la prise en main manuelle pour remettre le robot en place. Il fallait donc être en mesure de retrouver la commande manuelle sans réactiver les autres nodes

Pour cela j'ai laissé le node **/emergency_stop** prendre en charge le lancement du node **/emergency_activation** pour réactiver les moteurs. J'ai ensuite créé un subscriber capable de recevoir les informations provenant de l'interface web avec le topic **/choix**. Ainsi, lorsque l'utilisateur utilise le bouton [Réactivation] une nouvelle phase démarre pendant laquelle les nodes sont réactivés.

Duplication du package d'arrêt d'urgence

Une fois que la réactivation des nodes ne reposait plus que sur les interactions avec les boutons de l'interface web, il était alors impossible de d'utiliser l'arrêt d'urgence pour les tests simples. Ayant besoin de tester le docking uniquement j'avais besoin d'un nouvel arrêt d'urgence ne faisant appel qu'à lui même pour retrouver les mêmes fonctionnalités que celui qui venait d'être mis en place. C'est pourquoi j'ai créé le package « Emergency_stop_for_tests ».

Modifications du fonctionnement de /emergency_activation

En abandonnant le joystick disponible sur la page web, j'ai décidé de lancer le node de commande au clavier au moment où les moteurs sont réactivés.

D'autre part pour se passer du fonctionnement nécessitant l'utilisation d'un subscriber j'ai décidé de passer par une interruption au clavier pour réactiver les nodes. Cette interruption utilise un principe de threading qui attend n'importe quelle saisie au clavier pour finir la réactivation en tuant au passage le node de commande au clavier pour que les commandes n'interfèrent pas entre elles.

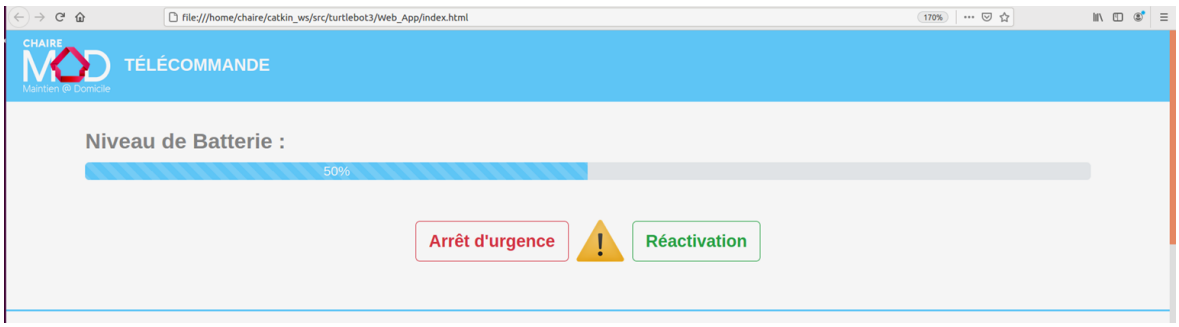

Illustration 2: Nouvelle configuration de l'arrêt d'urgence

Test d'autres démonstrateurs du turtlebot

Dans le cadre de la présentation pédagogique aux élèves de CM2, j'ai décidé de tester les autres applications disponibles sur le site Robotis :

https://emanual.robotis.com/docs/en/platform/turtlebot3/basic_examples/#move-using-interactive-markers

Turtlebot Follow demo :

Cette application peut s'avérer intéressante dans le cadre de la cohabitation avec une personne. En effet, une fonctionnalité permettant au robot de suivre son utilisateur peut avoir son utilité.

Cependant, en suivant le procédé lié au lancement de la commande j'obtiens cette erreur

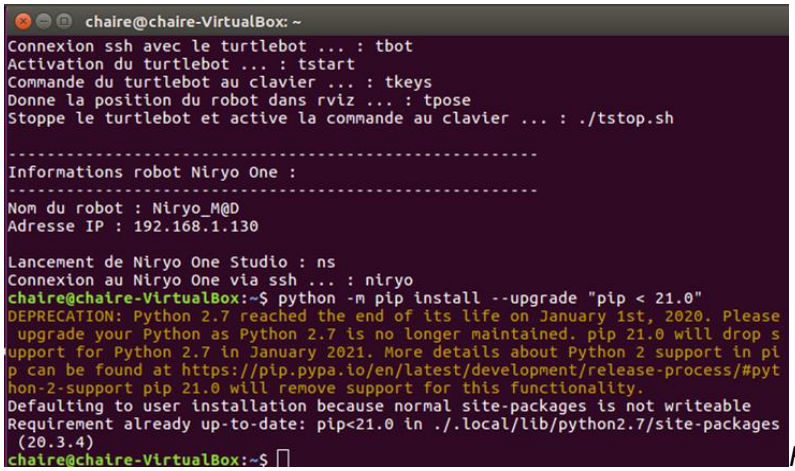

Figure 2: Screenshot du terminal avec l'erreur

En essayant de passer par pip3 je n'arrive pas non plus à installer les packages nécessaires.

Patrol :

Cette fonctionnalité permet d'ordonner au robot de suivre des chemins spécifiques se basant sur des formes. Elle n'a pas réellement d'intérêt, d'autant plus qu'elle a tendance à dysfonctionner. Cependant elle m'a été utile pour tester l'arrêt d'urgence.

Turtlebot Panorama demo :

J'ai réussi à lancer tous les nodes et à faire tourner le robot sur lui même pour prendre les photos mais je n'ai pas pu visualiser le résultat car aucune image n'est affichée après la commande : rqt_image_view image:=/turtlebot3_panorama/panorama

Obstacle description :

Je voudrais essayer d'utiliser cette fonctionnalité pour avoir une meilleure gestion du déplacement autonome du turtlebot. Le lidar permet de détecter la plupart des obstacles. Cependant, certains pieds de chaise ignorés par le repérage d'obstacles pourraient logiquement être évités grâce au retour du module de caméra