2 - ROS2 - Robotique mobile

- Turtlesim et modèle de Dubins
- TurtleBot3 - Bases en Simulation
- TurtleBot3 - Piloter le Robot
- Calibration de la caméra
- Suivi de ligne ROS2 Humble
- Behavior Trees Demo

Turtlesim et modèle de Dubins

https://docs.ros.org/en/humble/Tutorials/Beginner-CLI-Tools/Introducing-Turtlesim/Introducing-Turtlesim.html

TurtleBot3 - Bases en Simulation

Astuces

Gazebo

- Réinitialiser la pose du robot
  - `ctrl+Shift+R`
  - Edit --> Reset Model Poses

Le robot ne spawn pas

On a remarqué que parfois certains processus de gazebo continuent à tourner ou sont redémarrés malgré l'arrêt du noeud ROS principal. Il faut alors tuer le processus avecla commande `kill 1234`, voir commandes utiles ci-dessous.

Commandes utiles

- Lister les processus système (programmes) qui tournent actuellement
  `ps -ef`
- Lister les processus système (programmes) qui tournent actuellement sous forme d'arbre hiérarchisé : un processus enfant est rattaché à une branche d'un processus parent dont il dépend
  `ps -ef --forest`
- Parmi ces processus, sélectionner ceux qui contiennent le mot-clé `gazebo`
  `ps -ef --forest | grep ros`
- Repérer l'ID du processus
- Envoyer le signal d'arrêt du processus `SIGTERM` "mode gentil" : on demande au programme de s'arrêter
  `kill 1234`
- Si le processus continue tout de même à tourner, envoyer le signal de destruction du processus `SIGKILL` "mode méchant" :
  `kill -9 1234`

Ressources

- https://www.classes.cs.uchicago.edu/archive/2022/fall/20600-1/warmup_project.html

- https://emanual.robotis.com/docs/en/platform/turtlebot3/simulation/#gazebo-simulation

TurtleBot3 - Piloter le Robot

https://emanual.robotis.com/docs/en/platform/turtlebot3/navigation/#navigation

Les 2 ordinateurs dans le FabLab sur le mur de gauche sont installés sous Ubuntu 22.04 avec ROS2 Humble et les paquets nécessaires au pilotage des TurtleBot3. Il faut utiliser le compte étudiant (et non le compte fablab qui se login automatiquement au démarrage) avec le même mdp que celui utilisé en TP.

Vous pouvez emprunter un des 2 TurtleBot quand vous voulez en demandant à M. Carron ou M. Hentz dans le bureau en face :
- TurtleBot3 Burger
- TurtleBot3 Waffle + OpenManipulator X
Pensez bien à les recharger pour les suivants.

Un réseau Wifi fab-lab-5g est disponible et les TurtleBot configurés pour s'y connecter. **Attention ce réseau est limité à 20Go de données, donc ne pas l'utiliser pour autre-chose que la robotique**. Les PC doivent être connectés sur le même réseau que les TurtleBot. Vous devez ensuite vous connecter au Robot via ssh pour démarrer le noeud ROS2 côté robot :

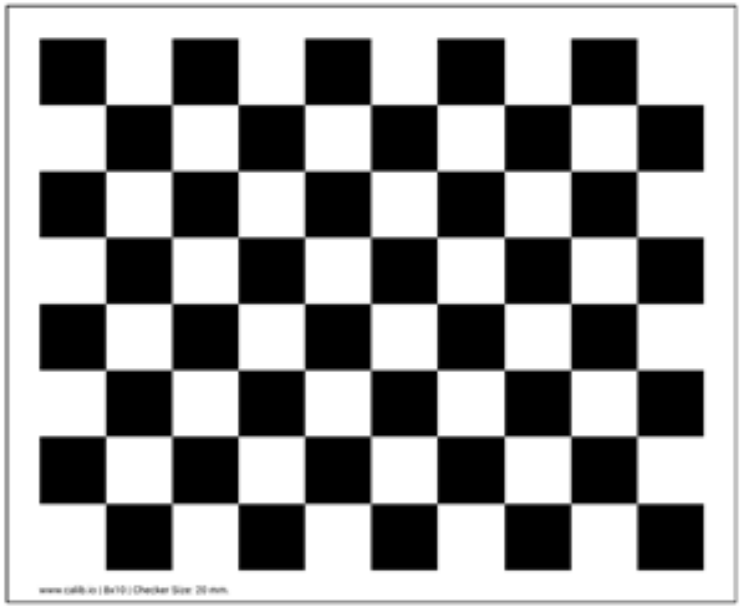
`ssh ubuntu@192.168.3.10` (burger) ou `ssh ubuntu@192.168.3.11` (waffle) avec le même mdp qu'en TP.

Les autres noeuds sont lancés sur le PC : télémanipulation, cartographie, navigation autonome, planification de trajectoire avec évitement d'obstacle etc.

Calibration de la caméra

https://docs.nav2.org/tutorials/docs/camera_calibration.html

- Générer un damier de calibration : 8 x 10 carrés de 20m
- avec https://calib.io/pages/camera-calibration-pattern-generator



- Ce sont les sommets intérieurs des carrés qui sont utilisés, donc 7x9 sommets

Ressources

- Noeud ROS2 pour Raspberry Cam https://github.com/christianrauch/camera_ros
- https://index.ros.org/p/camera_ros/
- https://medium.com/swlh/raspberry-pi-ros-2-camera-eef8f8b94304
-

Suivi de ligne ROS2 Humble

Le suivi de ligne repose surtout sur du traitement de l'image avec OpenCV, et l'envoi de commandes de vitesse au robot. Le code sous ROS1 devrait donc être portable assez directement sous ROS2.

Environnement de simulation Gazebo

Modélisation

Créer un paquet

- Créer un paquet ROS2 python « autonomy » qui implémente follower_p.py
- Adapter le CMakeLists.txt et package.xml en vous inspirant de :
  - turtlebot3_behavior_demos (/tb3_autonomy/scripts/test_vision.py)
  - turtlebot3/turtlebot3_example/turtlebot3_example/turtlebot3_position_control/turtlebot3_position_control.py
- Lancer ros2 launch turtlebot3_gazebo turtlebot3_circuit_competition.launch.py
- Démarrer la simulation en plaçant le robot au début de la piste
  - Caméra en vue de la ligne
- Lancer votre nœud python avec ros2 run autonomy follower_p.py

Ressources

Pour le suivi d'une ligne blanche :

- https://github.com/gabrielnhn/ros2-line-follower/blob/main/follower/follower/follower_node.py
  - `sudo apt install python3-cv-bridge python3-opencv`
  - ajouter au `~/.bashrc` : `export GAZEBO_MODEL_PATH=~/turtlebot3_ws/src/ros2-line-follower/follower/models`
- Le TP2 de robotique de Loïc Cuvillon donné à Telecom Physique Strasbourg
- ROS1 / OpenCV : https://github.com/osrf/rosbook/blob/master/followbot/follower_line_finder.py

Behavior Trees Demo

Concepts

https://docs.nav2.org/concepts/index.html#behavior-trees

https://docs.nav2.org/behavior_trees/overview/nav2_specific_nodes.html

https://docs.nav2.org/behavior_trees/overview/detailed_behavior_tree_walkthrough.html

Démo avec le Turtlebot3

https://github.com/sea-bass/turtlebot3_behavior_demos

Usage sans docker

- Installation

  https://github.com/sea-bass/turtlebot3_behavior_demos?tab=readme-ov-file#local-setup

- Vérifier que Gazebo fonctionne en lançant le noeud de base :
  `ros2 launch turtlebot3_gazebo turtlebot3_world.launch.py`
- Éteindre le noeud

On lance l'environnement de simulation lié à la démo de Behavior Tree :

- `ros2 launch tb3_worlds tb3_demo_world.launch.py`
  Le robot navigue en des positions connues avec pour but de trouver un cube d'une couleur spécifiée (rouge, vert ou bleu). La détection d'objets est faite par un simple seuillage en couleurs HSV avec des valeurs calibrées.

Démos de Behavior Trees en Python

On regarde le fichier `turtlebot3_behavior_demos/docker-compose.yaml` pour déterminer les commandes Bash correspondant aux commande docker indiquées dans le dépôt.

Dans un second terminal, on lance une des démos suivantes :

- `ros2 launch tb3_autonomy tb3_demo_behavior_py.launch.py tree_type:=queue enable_vision:=true target_color:=green`

Démo avec `py_trees`

Les fichiers source de la démo sont :

- tb3_demo_behavior_py.launch.py
- autonomy_node.py
  - navigation.py
    - test_move_base.py
  - vision.py
    - test_vision.py