TP ROS2/ Humble

P. Gaussier

0) check the examples with turtlesim and gazebo to be sure you understand how communication is managed in ROS.

1) Test the communication example between two node examples (publisher/subscriber example). When it is working suppress the display for each message and instrument the code to measure the transmission speed and its reliability (mean duration between two transmissions and their variance). How can you control this speed?

https://docs.ros.org/en/humble/Tutorials/Beginner-Client-Libraries/Colcon-Tutorial.html

2a) Test the example of obstacle avoidance and wall following proposed in the archive.
For the lidar, be careful the number of sensors need to be changed and the number of zones.

2b) Transform the example in a classical Braitenberg vehicule for obstacle avoidance.

3) Write in matrix form a Braitenberg vehicle capable of following a wall while staying close to the wall (different from simple obstacle avoidance). The solution should not use if/then/else structures nor a threshold as it is the case in the examples proposed in 2a (think the solution as an equilibrium point).

4) Knowing the complexity of the environment, at what speed should the robot run to be sure to
be sure to avoid the obstacles correctly. Which "hidden" parameter should you take into account?
take into account?

5) Add a mechanism that allows your robot to go towards a light source (use the ambient light
ambient light sensors). If you prefer you could detect an object using the camera (could be more difficult).

5) Control a population of robots so that they follow each other (put a light source on each robot).

Install ROS2 Humble on your computer, then modify your .bashrc

```
echo "source /opt/ros/humble/setup.bash" >> ~/.bashrc
echo "source /usr/share/colcon_cd/function/colcon_cd.sh" >> ~/.bashrc
echo "export _colcon_cd_root=/opt/ros/humble/" >> ~/.bashrc
```

you can change your domain id to dialog with a specific robot:
https://docs.ros.org/en/humble/Concepts/About-Domain-ID.html

turtle sim:
```
> ros2 run turtlesim turtlesim_node
```

in another terminal:
```
> ros2 run turtlesim turtle_teleop_key
```

to see the different nodes on the ROS network:

```
> ros2 node list
> ros2 topic list
> ros2 service list
> ros2 action list
```

See details using graphical interface:
```
> rqt
```

use rqt to call the `/spawn` service.
`/spawn` will create another turtle in the turtlesim window.

```
> ros2 run turtlesim turtle_teleop_key --ros-args --remap
turtle1/cmd_vel:=turtle2/cmd_vel
```

https://docs.ros.org/en/humble/Tutorials/Beginner-CLI-Tools/Understanding-ROS2-Topics/Understanding-ROS2-Topics.html

create a workspace:

https://docs.ros.org/en/humble/Tutorials/Beginner-Client-Libraries/Creating-A-Workspace/Creating-A-Workspace.html

```
cd
mkdir -p ~/ros2_ws/src
cd ~/ros2_ws/src

git clone https://github.com/ros2/examples src/examples -b humble

colcon build --symlink-install

colcon test
```

Testing communication between two nodes:

https://docs.ros.org/en/humble/Tutorials/Beginner-Client-Libraries/Writing-A-Simple-Cpp-Publisher-And-Subscriber.html

```
ros2 pkg create --build-type ament_cmake cpp_pubsub
create : cpp_pubsub in ros2_ws/src
```

```
cd cpp_pubsub/src
wget -O publisher_member_function.cpp
https://raw.githubusercontent.com/ros2/examples/humble/rclcpp/topics/minimal_pub
lisher/member_function.cpp

come back in  ros2_ws/src to compile using:
colcon build --packages-select cpp_pubsub

source install/local_setup.bash

> ros2 run cpp_pubsub talker
```

and in another terminal:

```
> ros2 run cpp_pubsub listener

or more simply go in the build directory and launch the ./talker
and  ./listener…
Some thing to compile: you can use the make created by cmake in the build.
```

==========

Turtlebot and Gazebo simulator

https://ros2-industrial-workshop.readthedocs.io/en/latest/_source/navigation/ROS2-Turtlebot.html

install apt packages:
ros-humble-turtlebot3
ros-humble-turtlebot3-gazebo

default domain id=0

```
export ROS_DOMAIN_ID=11
```

in a first terminal:

```
source  /opt/ros/humble/setup.bash
export TURTLEBOT3_MODEL=burger

export GAZEBO_MODEL_PATH=$GAZEBO_MODEL_PATH:`ros2 pkg \
prefix turtlebot3_gazebo \
`/share/turtlebot3_gazebo/models/

ros2 launch turtlebot3_gazebo empty_world.launch.py

ros2 launch turtlebot3_gazebo turtlebot3_house.launch.py
```

in a second terminal:

```
export TURTLEBOT3_MODEL=burger
ros2 run turtlebot3_teleop teleop_keyboard

ros2 topic info /scan

sensor_msgs/msg/LaserScan
ros2 topic echo /scan

ros2 component types
```

```
workspace_folder/
    src/
      package_1/
          CMakeLists.txt
          package.xml

cd ros2_ws/src


//https://docs.ros.org/en/humble/Tutorials/Beginner-Client-Libraries/Writing-A-Simple-Cpp-Publisher-And-Subscriber.html


check laserscan turtlebot

ros2 topic list
ros2 topic type /scan

    sensor_msgs/msg/LaserScan

ros2 topic echo /scan

bandwith used
ros2 topic bw /scan


ros2 topic pub --once /cmd_vel geometry_msgs/msg/Twist "{linear: {x: 2.0, y: 0.0, z: 0.0}, angular: {x: 0.0, y: 0.0, z: 1.8}}"

ros2 topic pub --once /cmd_vel geometry_msgs/msg/Twist "{linear: {x: 0.0, y: 0.0, z: 0.0}, angular: {x: 0.0, y: 0.0, z: 0.0}}"

message structure for laserscan:

std_msgs/Header header
float32 angle_min
float32 angle_max
float32 angle_increment
float32 time_increment
float32 scan_time
float32 range_min
float32 range_max
float32[] ranges
float32[] intensities

info lidar:
https://emanual.robotis.com/docs/en/platform/turtlebot3/appendix_lds_01/

================================================================
base tp:


https://www.theconstructsim.com/exploring-ros2-with-a-wheeled-robot-4-obstacle-avoidance/

git clone https://bitbucket.org/theconstructcore/exploring-ros2-with-wheeled-robot.git

source install/local_setup.bash
```

```
source  /opt/ros/humble/setup.bash
export TURTLEBOT3_MODEL=burger

export GAZEBO_MODEL_PATH=$GAZEBO_MODEL_PATH:`ros2 pkg \
prefix turtlebot3_gazebo \
`/share/turtlebot3_gazebo/models/

ros2 launch turtlebot3_gazebo empty_world.launch.py
```
or
```
ros2 launch turtlebot3_gazebo turtlebot3_house.launch.py

cd exploring-ros2-with-wheeled-robot/
```

You can use the AttachLightPlugin, here's an example.

See also
https://github.com/jimfinnis/lightsensor_gazebo

use object detection in ros:
https://www.stereolabs.com/docs/ros2/object-detection/

http://docs.ros.org/en/melodic/api/sensor_msgs/html/msg/Illuminance.html