https://fr.mathworks.com/help/ros/ug/ros-log-files-rosbags.html

ROSbag :

2. Examining and playing the bag file

Now that we've recorded a bag file using **rosbag record** we can examine it and play it back using the commands **rosbag info** and **rosbag play**. First we are going to see what's recorded in the bag file. We can do the **info** command -- this command checks the contents of the bag file without playing it back. Execute the following command from the bagfiles directory:

```
rosbag info <your bagfile>
```

You should see something like:

```
path:        2014-12-10-20-08-34.bag
version:     2.0
duration:    1:38s (98s)
start:       Dec 10 2014 20:08:35.83 (1418270915.83)
end:         Dec 10 2014 20:10:14.38 (1418271014.38)
size:        865.0 KB
messages:    12471
compression: none [1/1 chunks]
types:       geometry_msgs/Twist [9f195f881246fdfa2798d1d3eebca84a]
             rosgraph_msgs/Log   [acffd30cd6b6de30f120938c17c593fb]
             turtlesim/Color     [353891e354491c51aabe32df673fb446]
             turtlesim/Pose      [863b248d5016ca62ea2e895ae5265cf9]
topics:      /rosout                  4 msgs    : rosgraph_msgs/Log   (2 connections)
             /turtle1/cmd_vel       169 msgs    : geometry_msgs/Twist
             /turtle1/color_sensor 6149 msgs    : turtlesim/Color
             /turtle1/pose         6149 msgs    : turtlesim/Pose
```

This tells us topic names and types as well as the number (count) of each message topic contained in the bag file. We can see that of the topics being advertised that we saw in the **rostopic** output, four of the five were actually published over our recording interval. As we ran **rosbag record** with the -a flag it recorded all messages published by all nodes.

The next step in this tutorial is to replay the bag file to reproduce behavior in the running system. First kill the teleop program that may be still running from the previous section - a Ctrl-C in the terminal where you started turtle_teleop_key. Leave turtlesim running. In a terminal window run the following command in the directory where you took the original bag file:

```
rosbag play <your bagfile>
```

In this window you should immediately see something like:

```
[ INFO] [1418271315.162885976]: Opening 2014-12-10-20-08-34.bag

Waiting 0.2 seconds after advertising topics... done.

Hit space to toggle paused, or 's' to step.
```

In its default mode rosbag play will wait for a certain period (.2 seconds) after advertising each message before it actually begins publishing the contents of the bag file. Waiting for some duration allows any subscriber of a message to be alerted that the message has been advertised and that messages may follow. If **rosbag play** publishes messages immediately upon advertising, subscribers may not receive the first several published messages. The waiting period can be specified with the -d option.

Eventually the topic /turtle1/cmd_vel will be published and the turtle should start moving in turtlesim in a pattern similar to the one you executed from the teleop program. The duration between running **rosbag play** and the turtle moving should be approximately equal to the time between the original **rosbag record** execution and issuing the commands from the keyboard in the beginning part of the tutorial. You can have **rosbag play** not start at the beginning of the bag file but instead start some duration past the beginning using the -s argument. A final option that may be of interest is the -r option, which allows you to change the rate of publishing by a specified factor. If you execute:

```
rosbag play -r 2 <your bagfile>
```

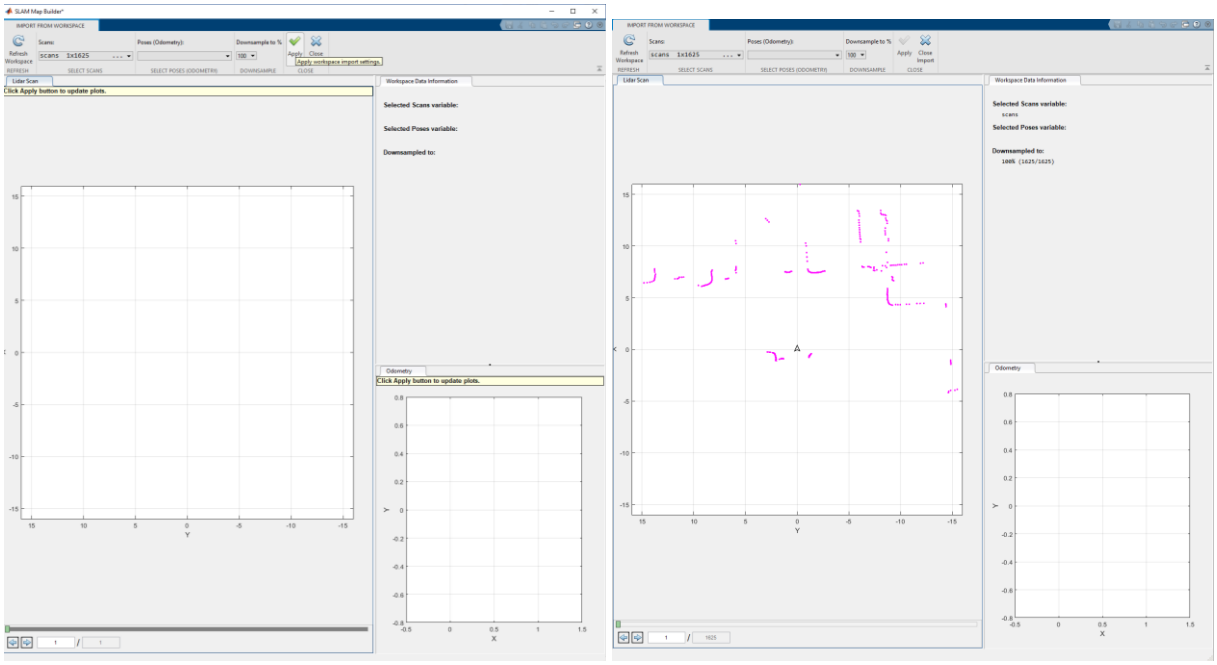
You should see the turtle execute a slightly different trajectory - this is the trajectory that would have resulted had you issued your keyboard commands twice as fast.

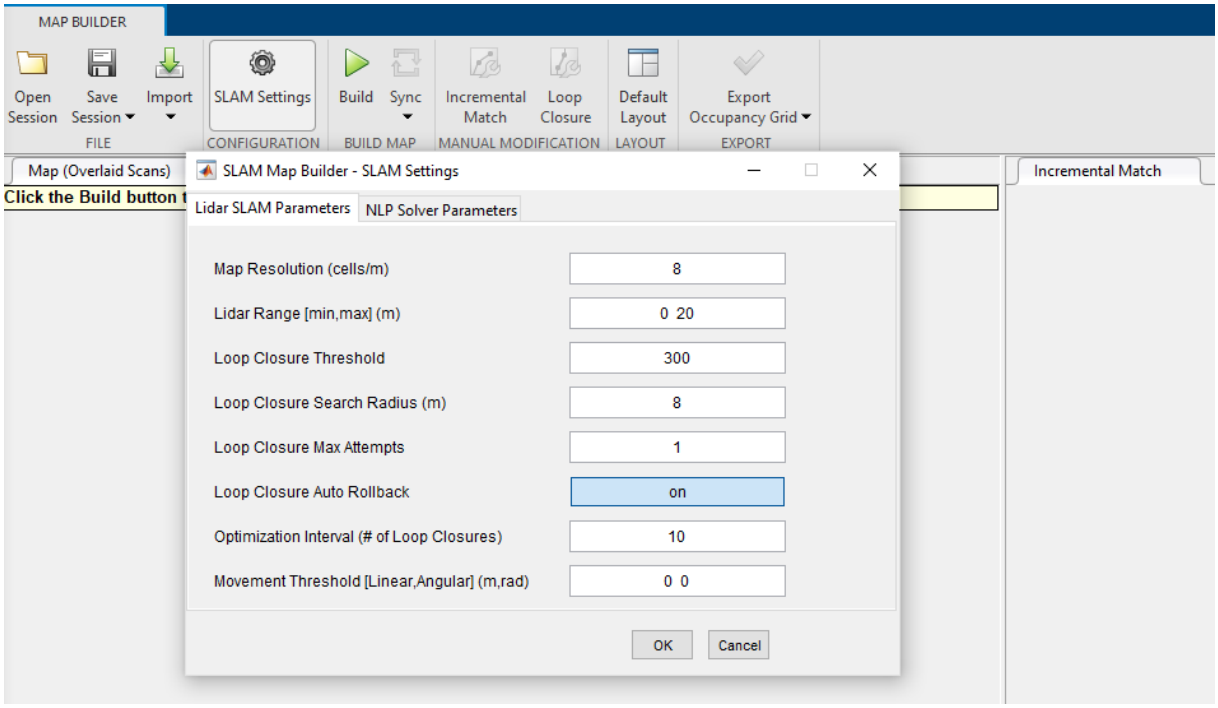
http://wiki.ros.org/rosbag/Tutorials/Recording%20and%20playing%20back%20data

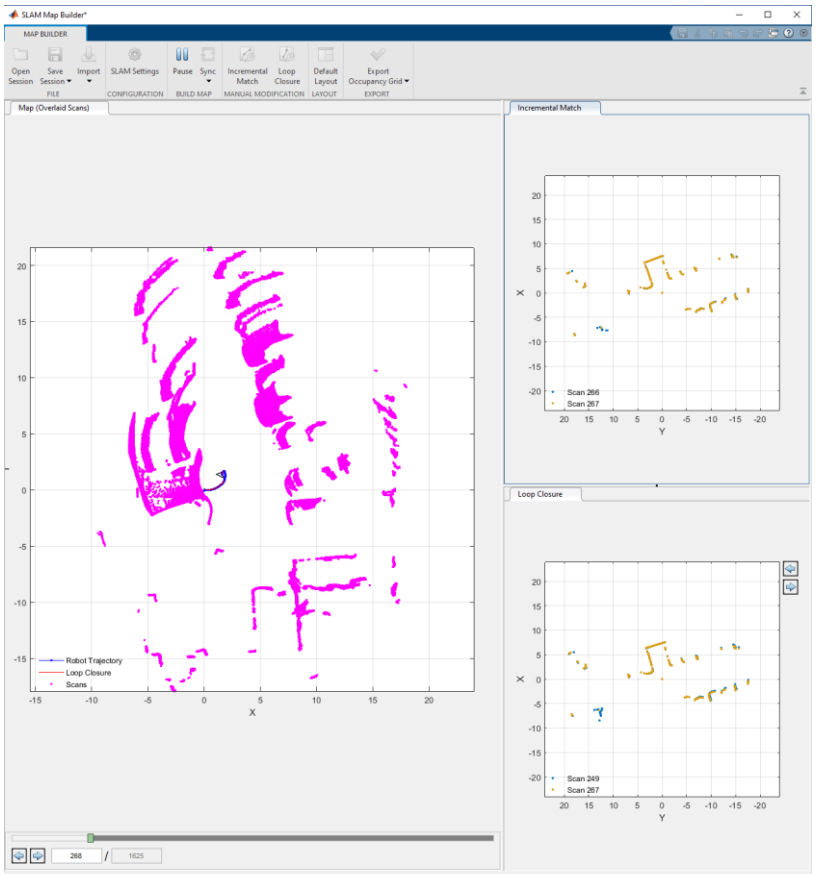
On crée un premier ROSbag à la main pour tester les fonctionnalités de SLAM.

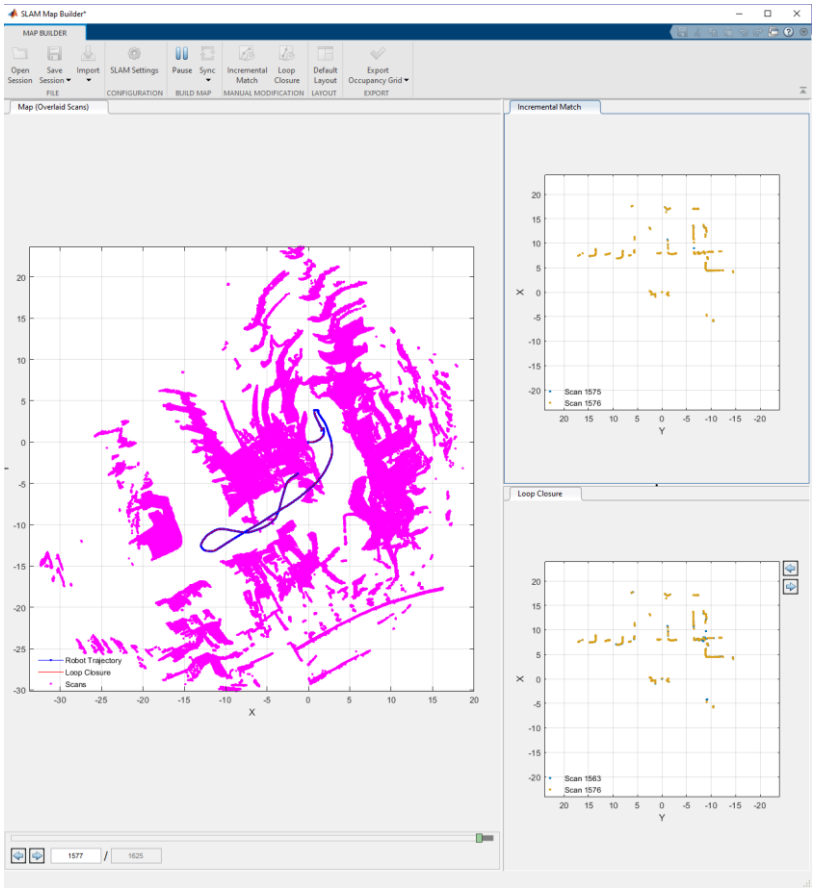
AUTRE METHODE QUE WORKSPACE :

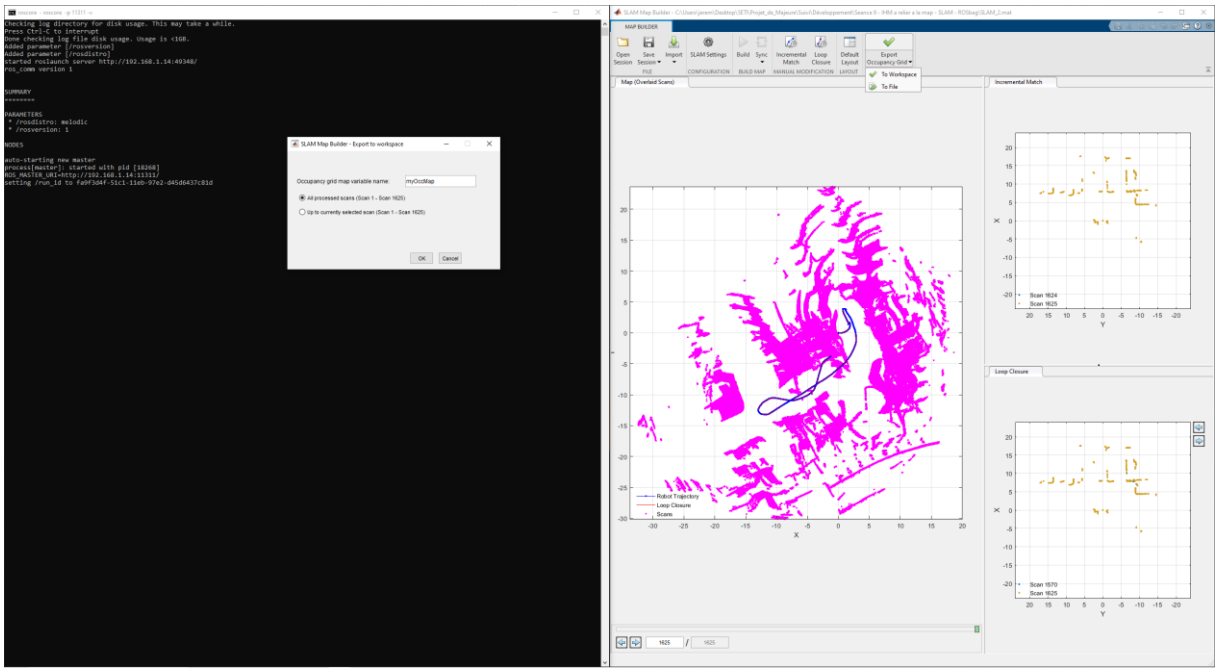
```
>> load slamLidarScans.mat
```



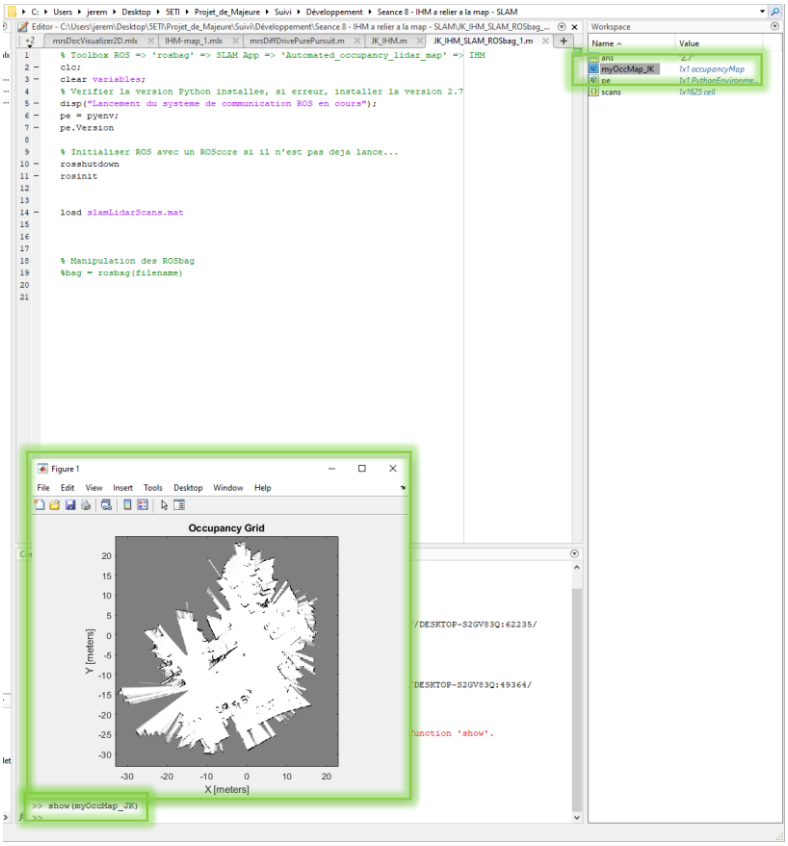



SLAM Map Builder*
MAP BUILDER
Open Session
Save Session
Import
SLAM Settings
Pause
Sync
Incremental Match
Loop Closure
Default Layout
Export Occupancy Grid
FILE
CONFIGURATION
BUILD MAP
MANUAL MODIFICATION
LAYOUT
EXPORT
Map (Overlaid Scans)
Incremental Match
Scan 266
Scan 267
Loop Closure
Scan 249
Scan 267
Robot Trajectory
Loop Closure
Scans
X
Y
268
1625


SLAM Map Builder*
MAP BUILDER
Open Session
Save Session
Import
SLAM Settings
Pause
Sync
Incremental Match
Loop Closure
Default Layout
Export Occupancy Grid
FILE
CONFIGURATION
BUILD MAP
MANUAL MODIFICATION
LAYOUT
EXPORT
Map (Overlaid Scans)
Incremental Match
Scan 1575
Scan 1576
Loop Closure
Scan 1563
Scan 1576
Robot Trajectory
Loop Closure
Scans
X
Y
1577
1625



Par exportation de variable dans le Workspace :



Par exportation en tant que fichier OcupancyMap en « .mat » :

En attendant voici ce que donne Slam avec l'exemple du tuto (sans passer par rosbag). Slam prend environ 30 min à construire la carte. (Il y a des exemples "temps réel" sur Mathworks donc j'espère que ça se basera sur la carte déjà construite.) Quoi qu'il en soit, si j'ai bien compris Slam construit la carte et met la trajectoire du robot enregistrée à partir respectivement des données de scan (lidar) et d'odométrie. Ce qui signifie que si l'on veut éviter de devoir faire passer le robot sur le terrain pour avoir la map, Slam devra s'ajouter à mon système de conception de map que j'avais commencé sur mon IHM. En bref : SLAM permet assez simplement de **visualiser** la trajectoire du robot par odométrie et son observation Lidar de son environnement, et il faudra une IHM indépendante pour le **contrôle et le test** des autres paramètres (comme la détection d'image...) et **la conception de la map en amont à la main**.

- ➔ Taille de la map : juste les coordonnées des traits
- ➔ Chemin guidé préenregistré avec les points d'intérêt…
- ➔ Modèle 3D potentiellement à réaliser…