Table of Contents

1. Localization and navigation

The localization and navigation stack is collection of nodes which process lidar data and use it to navigate.

1.1. Repository

Windesheim-Willy/navigation_stack

1.2. How does it work

1.2.1. Navigation

Data flow in navigation stack

```
                                    [map_server|1] -----\  [map_server|2]---\
                                                         \                  \
[lidar node] -> [lidar_filters] -> [laser_scan_matcher] -> [AMCL] ------> [move_base]
-> [cmd_vel]
                      \                                    /               /
                       \-----------------------------------/---------------/
```

The lidar node captures the data from the hardware device and puts it on the service bus. However in the case of Willy the lidar data is not always perfect since the metal frame of the robot creates invalid data points. These invalid data points confuse the algorithms. This is fixed by adding the 'scan_to_scan_filter_chain' node from the 'laser_filters' package. Using a chain of two filters it is possible to "crop" the lidar data. This data is fed to the other nodes.

The 'laser_scan_matcher' node uses the filtered lidar data to create odometry data based on the moving data points. The odometry data however only tells the stack how much it has moved relative to its starting position. Odometry data is also prone to drift especially when using the 'laser_scan_matcher' since it can't handle big empty spaces. This is where the 'AMCL' node comes into play. The 'AMCL' node takes into account: the estimated starting position, odometry offset from the starting point, lidar data and a lidar map. It can determine the odometry drift based on the lidar data and the map and compensate with a second offset. Because the 'AMCL' also takes into account the map we now have localization.

'AMCL' uses the first instance of the 'map_server', this map server publishes the raw map which was generated by 'hector_map', just slightly cropped and rotated to improve performance. The 'move_base' node uses the second instance of the 'map_server' which publishes a modified map in which incorrectly mapped walls are manually filled in. This is done because the lidar can't see reflective materials like glass or mirrors. By filling in these areas the 'move_base' will not try the navigate through glass. The 'AMCL' node however needs a map is is as close to the lidar data so it can't localize itself.

This position is then fed into the 'move_base' node which uses the lidar data and map to create a maps of obstacles called costmaps. It then uses the position from 'AMCL', the costmaps and a goal to generate a path and start giving commands to the robot to follow that path as closely as possible

1.2.2. Mapping

Data flow while mapping

```
[lidar node] -> [lidar_filters] -> [hector_mapping] -> [map_saver]
    \---------> [rosbag capture]
```

The mapping process uses the same lidar data and lidar filters. The raw lidar data can also be captured in a bag-file so this mapping process can be redone offline. The lidar data is fed into the 'hector_mapping' node which will generate a 2d map based on the lidar data.

However this process is not always perfect since it has trouble with seeing transparent and reflective surfaces. Objects like chairs and tables can be problematic since the surface area of the legs are small and easy to miss if the lidar is not close enough. Furthermore if objects become larger above the height at which the lidar measures the object may be drawn to small on the map causing collisions.

The 'hector_mapping' node is not great at erasing objects from the map. So it is best to map a area when there are no moving objects like humans.

1.3. How to run?

To run the localization and navigation stack you need to have a instance of ROS master and the Lidar node running

To start autonomous navigation you need to use the 'start-live-navstack.sh'

```
./start-live-navstack.sh
```

The navigation stack requires a Lidar map of the environment it must navigate in. This map can be built by manually moving the robot using the keyboard node and running the mapping process. The mapping process can be started with the following command

```
./start-live-mapping.sh
```

In order to use this map we need to save a copy. To save the map that is on the service bus use the following command:

```
rosrun map_server map_saver -f {name of save file}
```

It has proven to be useful to test the software in a VM when access to the hardware is limited. To do so just run the 'start-sim-navstack.sh' or 'start-sim-mapping.sh' instead of the commands listed above. These commands will start a replay of a rosbag containing lidar data instread of the real thing. They also set a parameter which tells the nodes to accept out of date timestamps which are required to function in a simulated environment