MOBILE ROBOTS (01416511)

Mobile Robot Chapter 6 : SLAM and Localization

SLAM

SLAM stands for Simultaneous Localization and Mapping. It is a fundamental problem in robotics and autonomous systems, where a robot explores an unknown environment, builds a map of that environment, and simultaneously determines its own position within that map.

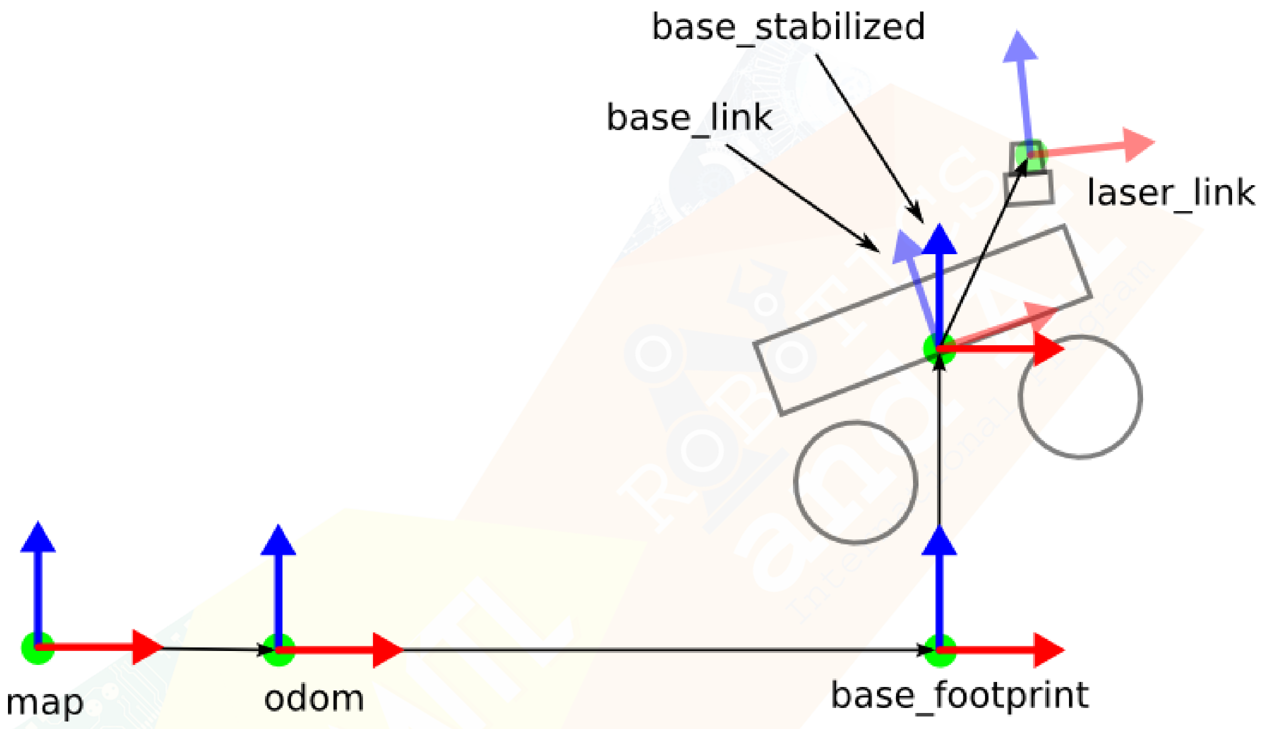
Brief Overview of SLAM:

1. **Localization:** Localization is the process of determining a robot's position and orientation (pose) within an environment. In a SLAM scenario, the robot does not have prior knowledge of its location, making it a challenging task. The robot uses various sensors such as odometry, GPS, inertial measurement units (IMUs), and visual odometry to estimate its pose.
2. **Mapping:** Mapping involves creating a representation of the environment, typically in the form of a 2D or 3D map. The robot uses sensor data, such as laser scanners (LiDAR), depth cameras, and cameras, to observe the surroundings and build the map. The map can include information about obstacles, landmarks, or other relevant features.
3. **Simultaneous Localization and Mapping:** The main challenge in SLAM is to perform both localization and mapping simultaneously. As the robot moves through the environment, it must update its estimated pose and build an accurate map simultaneously, all while dealing with uncertainties in sensor measurements, motion, and environmental features.
4. **Online and Offline SLAM:** SLAM algorithms can be categorized as online or offline. Online SLAM processes data in real-time, updating the robot's pose and map as it moves through the environment. In contrast, offline SLAM processes data after the robot has completed its exploration and stores the data for post-processing.
5. **Applications:** SLAM has numerous practical applications, such as autonomous vehicles, robotic exploration, indoor navigation, augmented reality, and even virtual reality. It is a critical technology for enabling robots and autonomous systems to operate in unknown or dynamic environments.
6. **Challenges:** SLAM is a complex problem due to sensor noise, uncertainties in motion, data association, and the computational burden of processing large amounts of data in real-time. Researchers have developed various SLAM algorithms, ranging from probabilistic methods like Extended Kalman Filter (EKF) and Particle Filter SLAM (PF-SLAM) to graph-based approaches like GraphSLAM and pose graph optimization techniques.

Overall, SLAM plays a crucial role in robotics and autonomous systems by enabling them to autonomously navigate and build a representation of their surroundings, even in unknown or changing environments.

Hector SLAM

In this lab sheet, use the Hector SLAM method to create a map for the robot. ***hector_slam*** uses the ***hector_mapping*** node for learning a map of the environment and simultaneously estimating the platform's 2D pose at the laser scanner frame rate. The frame names and options for ***hector_mapping*** have to be set correctly. This tutorial explains the different options. The image below shows all potential frames of interest in a simplified 2D view of a robot traveling through rough terrain, leading to roll and pitch motion of the platform.



The general relationship between the ***map***, ***odom***, and ***base_link*** frames is already described in Coordinate Frames for Mobile Platforms. We use two frames in between ***odom*** and ***base_link***:

1. ***base_footprint*** frame: Provides no height information and represents the 2D pose of the robot (position and orientation).
2. ***base_stabilized*** frame: Adds information about the robot height relative to the map/odom layer.

The ***base_link*** frame is rigidly attached to the robot and adds the roll and pitch angles compared to the ***base_stabilized*** frame. For this transformation, a system for estimation of the vehicle attitude like an AHRS or INS can be used (e.g., using the ***hector_imu_attitude_to_tf*** node). For a platform not exhibiting roll/pitch motion, the ***base_stabilized*** and ***base_link*** frames are equal. The transformation from ***base_link*** to ***laser_link*** is typically provided by a static transform publisher or the robot state publisher.

Install the requirement packages

```
$ sudo apt install ros-noetic-hector-mapping
```

Setup the SLAM package

```
$ cd ~/ired_ws/src
$ catkin_create_pkg ired_navigation roscpp rospy std_msgs
$ cd ~/ired_ws
$ catkin_make
$ source ~/.bashrc
```

Setup the SLAM launch file

```
$ roscd ired_navigation
$ mkdir launch
$ cd launch
$ touch slam.launch
$ code slam.launch
```

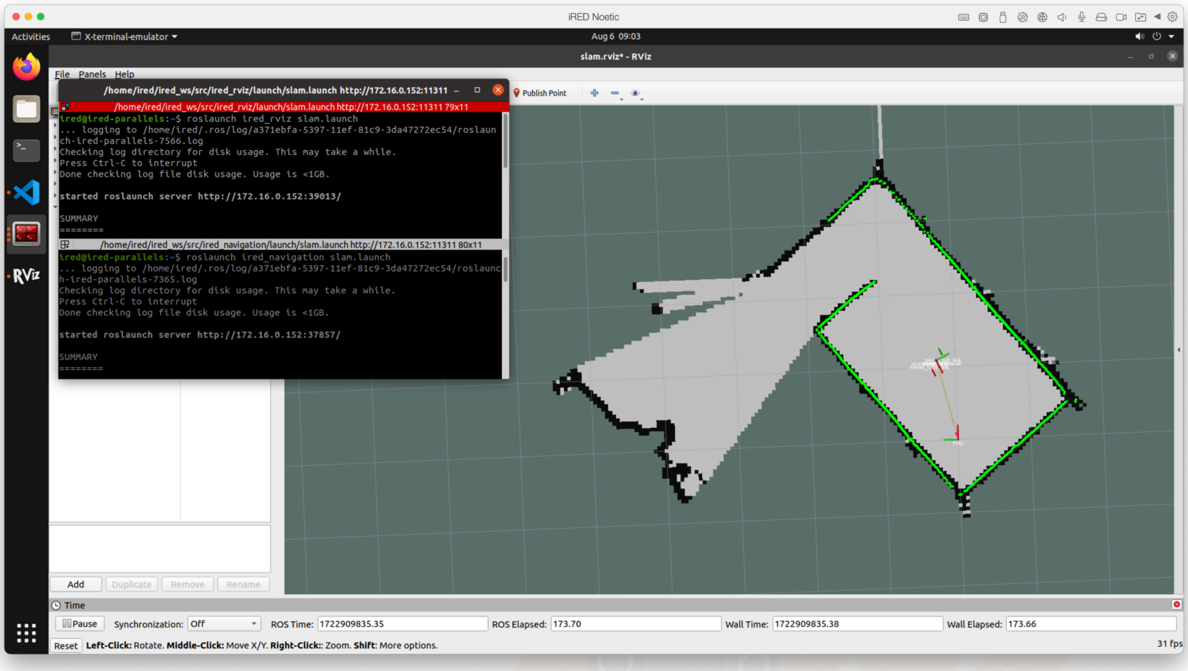
```
<?xml version="1.0"?>
<launch>
  <arg name="tf_map_scanmatch_transform_frame_name" default="scanmatcher_frame"/>
  <arg name="base_frame" default="base_link"/>
  <arg name="odom_frame" default="base_link"/>
  <arg name="pub_map_odom_transform" default="true"/>
  <arg name="scan_subscriber_queue_size" default="5"/>
  <arg name="scan_topic" default="scan"/>
  <arg name="map_size" default="2048"/>
  <include file="$(find hector_mapping)/launch/mapping_default.launch">
    <arg name="tf_map_scanmatch_transform_frame_name" value="$(arg
tf_map_scanmatch_transform_frame_name)"/>
    <arg name="base_frame" value="$(arg base_frame)"/>
    <arg name="odom_frame" value="$(arg odom_frame)"/>
    <arg name="pub_map_odom_transform" value="$(arg pub_map_odom_transform)"/>
    <arg name="scan_subscriber_queue_size" value="$(arg scan_subscriber_queue_size)"/>
    <arg name="scan_topic" value="$(arg scan_topic)"/>
    <arg name="map_size" value="$(arg map_size)"/>
  </include>
</launch>
```

If you don't have the ired_rviz package, please install it by using the following command(Optional)

```
$ cd ~/ired_ws/src
$ git clone https://github.com/aims-lab-kmitl/ired_rviz.git
$ cd ~/ired_ws
$ catkin_make
$ source ~/.bashrc
```

Test the SLAM package with the following command

```
$ roslaunch ired_bringup bringup.launch # terminal 1
$ roslaunch ired_navigation slam.launch # terminal 2
$ roslaunch ired_rviz slam.launch # terminal 3
```



```
/home/ired/ired_ws/src/ired_rviz/launch/slam.launch http://172.16.0.152:11311 79x11
ired@ired-parallels:~$ roslaunch ired_rviz slam.launch
... logging to /home/ired/.ros/log/a371ebfa-5397-11ef-81c9-3da47272ec54/roslaun
ch-ired-parallels-7566.log
Checking log directory for disk usage. This may take a while.
Press Ctrl-C to interrupt
Done checking log file disk usage. Usage is <1GB.

started roslaunch server http://172.16.0.152:39013/

SUMMARY
========
```

```
/home/ired/ired_ws/src/ired_navigation/launch/slam.launch http://172.16.0.152:11311 80x11
ired@ired-parallels:~$ roslaunch ired_navigation slam.launch
... logging to /home/ired/.ros/log/a371ebfa-5397-11ef-81c9-3da47272ec54/roslaunc
h-ired-parallels-7365.log
Checking log directory for disk usage. This may take a while.
Press Ctrl-C to interrupt
Done checking log file disk usage. Usage is <1GB.

started roslaunch server http://172.16.0.152:37857/

SUMMARY
========
```

Map Format

Maps manipulated by the tools in this package are stored in a pair of files. The ***YAML*** file describes the map metadata and names the image file. The image file encodes the occupancy data.

The image describes the occupancy state of each cell of the world with colors corresponding to the occupancy status of each pixel. Typically, whiter pixels denote free space, blacker pixels indicate occupied space, and pixels of intermediate shades represent areas of unknown occupancy. While color images are accepted, their color values are averaged to grayscale.

Image data is imported using ***SDL_Image***, and the supported formats depend on the ***SDL_Image*** library available on the platform. Generally, most common image formats are widely supported. However, PNG format is not supported on macOS, which is a notable exception.

Install the requirement packages

```
$ sudo apt install ros-noetic-map-server
```

Setup the map saver launch file

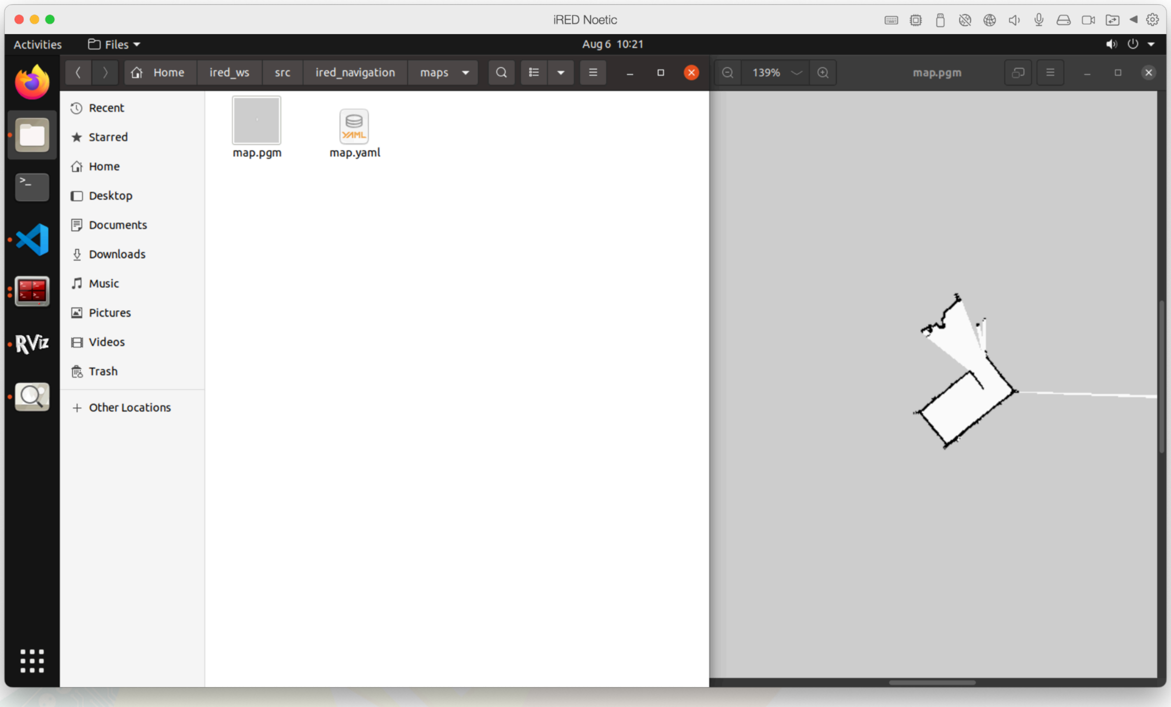
```
$ roscd ired_navigation
$ mkdir maps
$ cd launch
$ touch savemap.launch
$ code savemap.launch
```

```
<?xml version="1.0"?>
<launch>
    <arg name="map_name" default="map"/>
    <arg name="save_path" default="$(find ired_navigation)/maps/$(arg map_name)"/>
    <node name="map_server_node" pkg="map_server" type="map_saver"  args="-f $(arg
save_path)" output="screen"/>
</launch>
```

Test the save map package with the following command

```
$ roslaunch ired_navigation savemap.launch
```

You can view the map file at the path ***~/ired_ws/src/ired_navigation/maps***.



AMCL

AMCL stands for ***Adaptive Monte Carlo Localization***. It's a probabilistic localization system for a robot moving in 2D. AMCL uses a particle filter to track the pose of a robot against a known map. Here's a brief overview of how it works:

1. ***Particle Filter:*** AMCL uses a set of particles to represent the possible positions and orientations (poses) of the robot on the map. Each particle has a weight that represents the likelihood of the robot being in that pose.
2. ***Motion Update:*** When the robot moves, each particle's position is updated based on the robot's motion model, which accounts for the control commands given to the robot and the uncertainty in its motion.
3. ***Sensor Update:*** The robot uses its sensors (e.g., laser scanner, LiDAR) to take measurements of its surroundings. These measurements are compared to the known map to update the weights of the particles. Particles that match the sensor data well receive higher weights.

4. ***Resampling:*** Particles are resampled based on their weights to focus more on the highly probable poses. This step helps to discard unlikely poses and concentrate the computational effort on the more probable areas.
5. ***Pose Estimation:*** The robot's estimated pose is calculated from the weighted average of the particles.

AMCL is widely used in robotic navigation for tasks that require the robot to determine its position and orientation within a known map, enabling it to navigate accurately.

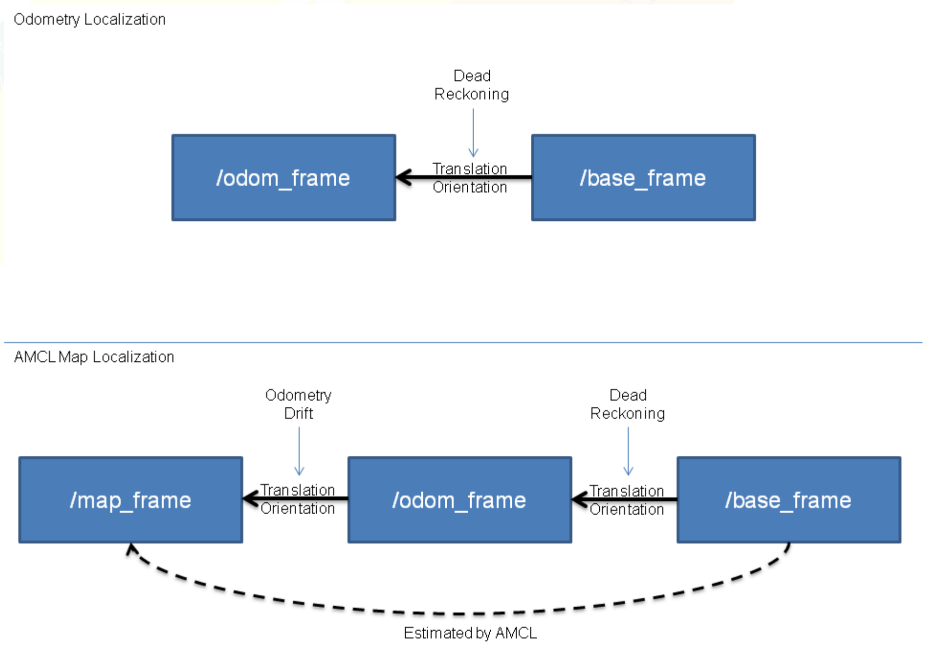
AMCL takes in a laser-based map, laser scans, and transform messages, and outputs pose estimates. On startup, ***AMCL*** initializes its particle filter according to the parameters provided. Note that, because of the defaults, if no parameters are set, the initial filter state will be a moderately sized particle cloud centered around (0,0,0).

Transforms

AMCL transforms incoming laser scans to the odometry frame (~odom_frame_id). Therefore, there must be a path through the tf tree from the frame in which the laser scans are published to the odometry frame.

An implementation detail: upon receipt of the first laser scan, AMCL looks up the transform between the laser's frame and the base frame (~base_frame_id), and latches it forever. Thus, AMCL cannot handle a laser that moves with respect to the base.

The drawing below illustrates the difference between localization using odometry and AMCL. During operation, AMCL estimates the transformation of the base frame (~base_frame_id) in respect to the global frame (~global_frame_id), but it only publishes the transformation between the global frame and the odometry frame (~odom_frame_id). Essentially, this transform accounts for the drift that occurs using dead reckoning. The published transforms are future-dated.



Install the requirement packages

```
$ sudo apt install ros-noetic-amcl
```

Setup the AMCL launch file

```
$ roscd ired_navigation/launch
$ touch amcl.launch
$ code amcl.launch
```

```
<launch>
  <!-- Arguments -->
  <arg name="scan_topic"     default="scan"/>
  <arg name="initial_pose_x" default="0.0"/>
  <arg name="initial_pose_y" default="0.0"/>
  <arg name="initial_pose_a" default="0.0"/>

  <!-- AMCL -->
  <node pkg="amcl" type="amcl" name="amcl">

    <param name="min_particles"             value="500"/>
    <param name="max_particles"             value="3000"/>
    <param name="kld_err"                   value="0.02"/>
    <param name="update_min_d"              value="0.20"/>
    <param name="update_min_a"              value="0.20"/>
    <param name="resample_interval"         value="1"/>
    <param name="transform_tolerance"       value="0.5"/>
    <param name="recovery_alpha_slow"       value="0.00"/>
    <param name="recovery_alpha_fast"       value="0.00"/>
    <param name="initial_pose_x"            value="$(arg initial_pose_x)"/>
    <param name="initial_pose_y"            value="$(arg initial_pose_y)"/>
    <param name="initial_pose_a"            value="$(arg initial_pose_a)"/>
    <param name="gui_publish_rate"          value="50.0"/>

    <remap from="scan"                      to="$(arg scan_topic)"/>
    <param name="laser_max_range"           value="3.5"/>
    <param name="laser_max_beams"           value="180"/>
    <param name="laser_z_hit"               value="0.5"/>
    <param name="laser_z_short"             value="0.05"/>
    <param name="laser_z_max"               value="0.05"/>
    <param name="laser_z_rand"              value="0.5"/>
    <param name="laser_sigma_hit"           value="0.2"/>
    <param name="laser_lambda_short"        value="0.1"/>
    <param name="laser_likelihood_max_dist" value="2.0"/>
    <param name="laser_model_type"          value="likelihood_field"/>

    <param name="odom_model_type"           value="diff"/>
    <param name="odom_alpha1"               value="0.1"/>
    <param name="odom_alpha2"               value="0.1"/>
    <param name="odom_alpha3"               value="0.1"/>
    <param name="odom_alpha4"               value="0.1"/>
    <param name="odom_frame_id"             value="odom"/>
    <param name="base_frame_id"             value="base_footprint"/>
  </node>
</launch>
```

Setup the navigation launch file

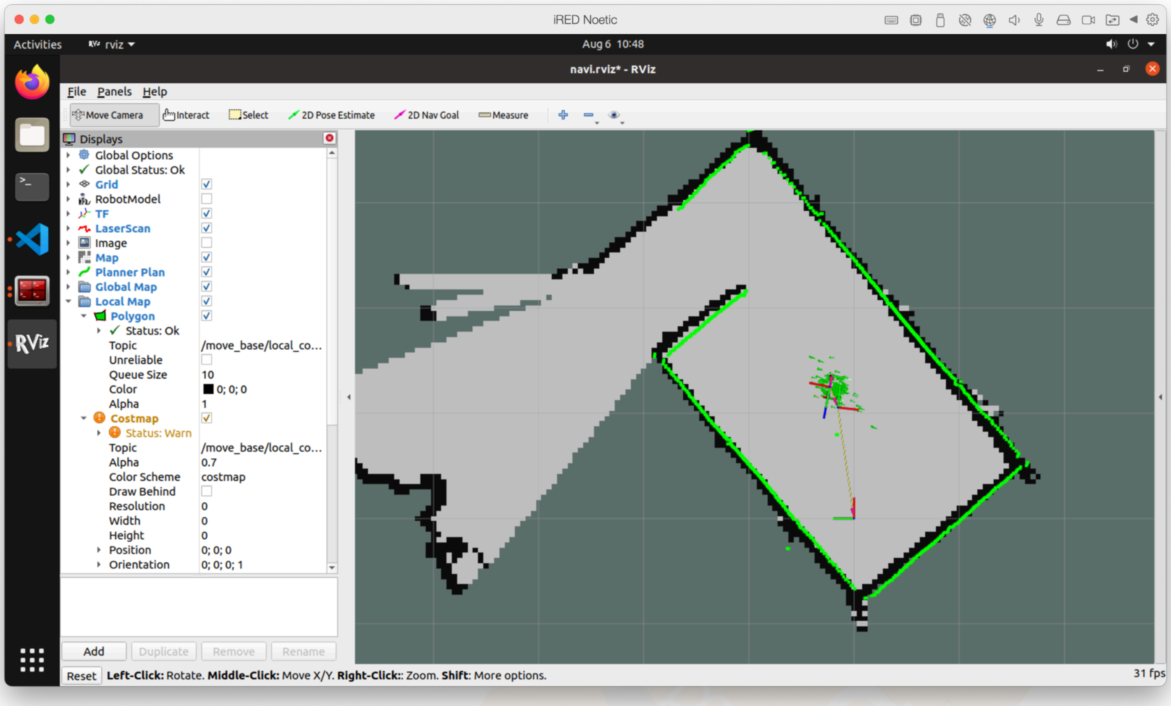
```
$ roscd ired_navigation/launch
$ touch navigation.launch
$ code navigation.launch
```

```
<?xml version="1.0"?>
<launch>
    <!-- map server -->
    <arg name="map_file" default="map"/>
    <arg name="map_path" default="$(find ired_navigation)/maps/$(arg map_file).yaml"/>
    <node name="map_server" pkg="map_server" type="map_server" args="$(arg map_path)"/>

    <!-- AMCL -->
    <arg name="scan_topic"     default="scan"/>
    <arg name="initial_pose_x" default="0.0"/>
    <arg name="initial_pose_y" default="0.0"/>
    <arg name="initial_pose_a" default="0.0"/>
    <include file="$(find ired_navigation)/launch/amcl.launch">
        <arg name="scan_topic"     value="$(arg scan_topic)"/>
        <arg name="initial_pose_x" value="$(arg initial_pose_x)"/>
        <arg name="initial_pose_y" value="$(arg initial_pose_y)"/>
        <arg name="initial_pose_a" value="$(arg initial_pose_a)"/>
    </include>
</launch>
```

Test the amcl package with the following command

```
$ roslaunch ired_navigation navigation.launch
```



Global Options	
Global Status: Ok	
Grid	✓
RobotModel	☐
TF	✓
LaserScan	✓
Image	☐
Map	✓
Planner Plan	✓
Global Map	✓
Local Map	✓
Polygon	✓
Status: Ok	
Topic	/move_base/local_co...
Unreliable	☐
Queue Size	10
Color	■ 0; 0; 0
Alpha	1
Costmap	✓
Status: Warn	
Topic	/move_base/local_co...
Alpha	0.7
Color Scheme	costmap
Draw Behind	☐
Resolution	0
Width	0
Height	0
Position	0; 0; 0
Orientation	0; 0; 0; 1

Left-Click: Rotate. **Middle-Click:** Move X/Y. **Right-Click:**: Zoom. **Shift**: More options.

Assignment

Group Assignment: The group representative should take a screenshot of the map file and submit it to Classroom.