M. Huraira Anwer - 25100314 Sadaan Tahir - 25100224

EE 3002 L1 (Junior Design Studio - Robotics)

Spring 2025 - LAB 8

This week's lab builds on the previous lab that explored **mapping**, **S**imultaneous **L**ocalization and **M**apping (**SLAM**), and **navigation** in ROS, with a focus on **Hector SLAM**, **GMapping**, and the **Navigation Stack**.

You will gain hands-on experience with fundamental principles and practical implementations of mapping and navigation techniques crucial for autonomous robotic systems. Through ROS tools and algorithms, you will explore environment mapping, SLAM, and path planning, gaining the expertise to configure and deploy autonomous navigation systems. This lab offers a valuable opportunity to enhance your robotics knowledge and develop practical skills with industry-standard tools for real-world applications.

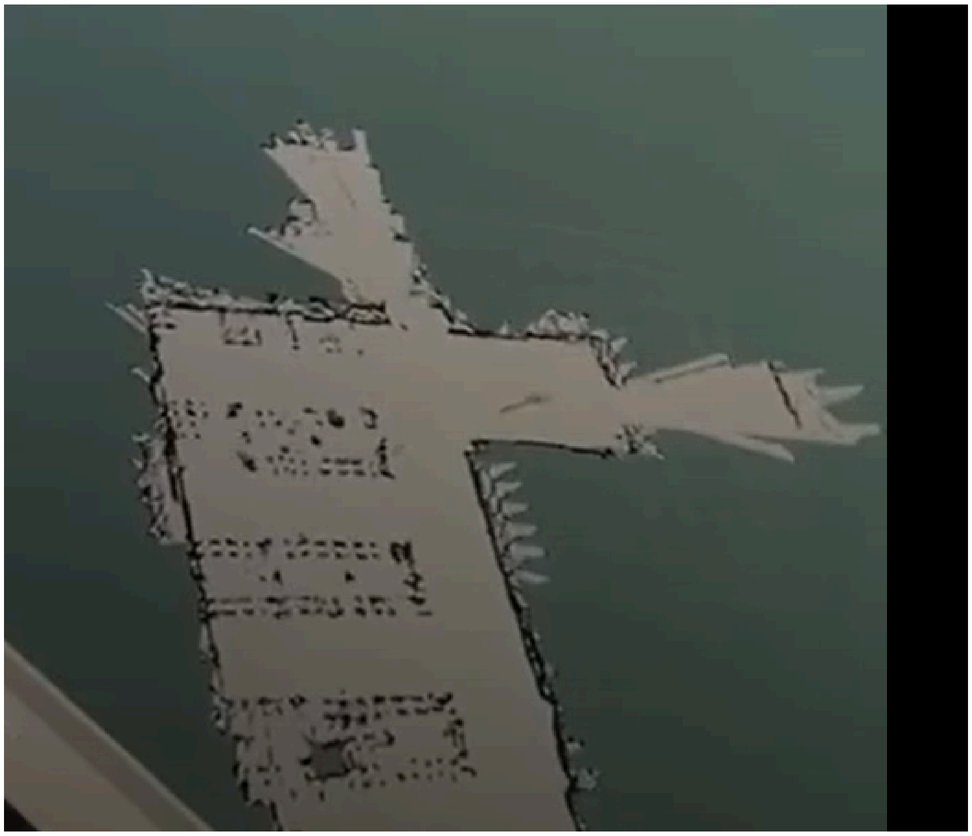
Using the map files from the previous lab, we will be first localizing the robot within that map and then navigating it to a goal using Lidar and the map such that it avoids the obstacles and reaches the goal correctly. To get an idea of what we'll be doing in this lab, check out this from last year's project: https://drive.google.com/file/d/1c3vXHVwJhJxHpstNNZdKRC_G2At98lqq/view

Note: This final lab will require A LOT of debugging. Use any resources you have under your disposal to get the tasks done and understand them completely as this lab will be pivotal to many of your projects.

While mapping if you are getting errors,

- Ensure you have followed steps **1-8** of Task 1 - Localization (mentioned below)
- Instead of recording a rosbag and playing it later which causes issues in rendering the map, it is better to map live, so connect remotely (with GUI) and launch Lidar and do **roslaunch hector_slam_launch tutorial.launch**
- **rosrun tf static_transform_publisher 0 0 0 3.1416 0 0 1.0 map scan 100**
- **Fixed frame** should be **'/map'** in **RVIZ**
- Now move the robot very slowly and turn it very carefully so it renders the obstacles in the map as accurately as possible, increasing the speed will result in a poor map quality. Keep doing this until you map the whole lab.

Ideally the **map** of the Feedback lab should look like this in **RVIZ …**



Task 1: Localization [25 MARKS]

Step 1. Make a new workspace called **lab8_ws**:

```
jdsrobo@ubuntu:~$ mkdir -p ~/lab8_ws/src
jdsrobo@ubuntu:~$ cd lab8_ws/
jdsrobo@ubuntu:~/lab8_ws$ catkin_make
jdsrobo@ubuntu:~/lab8_ws$ echo "source ~/lab8_ws/devel/setup.bash" >> ~/.bashrc
```

Step 2. Clone the following RPLidar ROS package into the **src** folder:

```
git clone https://github.com/Slamtec/rplidar_ros.git
```

Step 3. **Build** the workspace using **catkin_make** and **source** its **setup.bash** file.

Step 4. **Check** the connected USB devices on your Jetson Nano (**make sure the RPLidar is connected**) using this terminal command:

```
ls /dev/ttyUSB*
```

Give read/write permissions to the USB port where the Lidar is connected:

```
sudo chmod 666 /dev/ttyUSB0 ## Assuming the Lidar is connected to USB0 ##
```

Step 5. **Install** necessary packages for Hector SLAM

```
sudo apt-get update

sudo apt-get install qt5-qmake qtbase5-dev
```

Step 6. Clone the Hector SLAM repository (given below) under **~/lab8_ws/src/**:

```
git clone -b noetic-devel https://github.com/tu-darmstadt-ros-pkg/hector_slam.git
```

Step 7. **Build** the workspace using **catkin_make** and **source** its **setup.bash** file.

Step 8. Now, **modify the launch file** for the **hector_mapping package** to configure the robot's **base frame** and **odometry frame**, **laser scan topic** and to publish a static **transform** between the Lidar and the robot's base frame to link the Lidar's position relative to the robot.

Open the launch file in a text editor:

```
gedit ~/lab8_ws/src/hector_slam/hector_mapping/launch/mapping_default.launch
```

Modify the following lines to set the **base frame** and **odom frame** to **base_link**:

```
<arg name="base_frame" default="base_link"/>
<arg name="odom_frame" default="base_link"/>
<arg name="scan_topic" default="scan"/>
```

Add a **static transform publisher node** (as a new line) to broadcast the transform from **base_link** to the **Lidar's frame**:

```
<node pkg="tf" type="static_transform_publisher" name="base_to_laser_broadcaster"
args="x_val y_val z_val yaw_val pitch_val roll_val base_link laser 100"/>
```

*Note: Replace **x_val y_val z_val yaw_val pitch_val** and **roll_val** with your measured values. **yaw_val** should be **3.1416** to avoid inversion of frame while localizing.*

Change this line from true to false in **hector_slam_launch** package , **launch** folder, filename is **tutorial.launch**

```
<param name="/use_sim_time" value="false"/>
```

Step 9. **Install** these necessary **packages** for **localization** and **navigation**

```
sudo apt-get update

sudo apt install ros-noetic-amcl

sudo apt install ros-noetic-navigation

sudo apt install ros-noetic-map-server
pip3 install pyyaml
```

Step 10. **Unzip** the **localization.zip** package provided to you and place it in the **~/lab8_ws/src** Make the **.py** files in **localization/src** and **.launch** file in **localization/launch** executable using the chmod +x syntax. These are sample files that you may use for localization setup. Do catkin_make & source the **lab8_ws** directory.

Running this launch file will give **'/amcl_pose'** topics which will provide the localized and dynamically updating pose of the robot in the map (this will go as input for the next part i.e. navigation).

Note: You may need to debug / edit the provided files or make changes to ensure they work correctly for the next step.

If you encounter a "yaml module not found" error, add launch-prefix='python3' to the amcl.launch localization file in the tfpub.py and odom.py nodes (if you are running them from within the launch file, else you can run them independently before the launch file as well)

Step 11. The **amcl** package requires 5 things to work:

1. The map (.**yaml** and .**pgm** files)
2. The robot's initial position that must be published once
3. The robot's odom data should be published continuously.
4. The Lidar '/scan' topic that should be publishing the laser scan data
5. The odometry transformations that should be published using a TF broadcaster

Step 12. **Verify** localization by launching **RVIZ** in a terminal, subscribe to **/odom** and /**amcl_pose** topics in **RVIZ** and see the robot pose update in the map, also do **rostopic echo /amcl_pose** to view the robot's pose updating dynamically in the terminal as you move it.

Some helpful resources:

- https://wiki.ros.org/amcl
- https://docs.ros.org/en/noetic/api/robot_localization/html/

M. Huraira Anwer - 25100314 Sadaan Tahir - 25100224

Task 2: Navigation [25 MARKS]

Step 1. For the navigation part, **unzip** the **navigation.zip** package provided to you and place it under the **~/lab8_ws/src/** directory.

Make the **.launch** file/s in **navigation/launch/** executable using the chmod +x syntax. Ignore the file in the navigation**/src/** directory, its functionality is not complete, you need to implement that from scratch (instructions in **Step 2**) Do catkin_make & source the **lab8_ws** directory.

Running this launch file will give **'/move_base'** topics which will provide a path to follow which leads to the goal (user-defined) while avoiding obstacles. Point to a goal in the RVIZ, this will be used by the **move_base** files to generate a plan which you'll use in **Step 2** to navigate towards the goal.

Note: You may need to debug / edit the provided files or make changes to ensure they work correctly for the next step.

Step 2. Design a **PID controller** yourself that subscribes to **"/move_base/NavfnROS/plan"** through a callback function and converts this plan to develop a trajectory in order to simulate a unicycle to follow the same path as the plan, with the Robomaster simply tracking it, essentially treating time as a variable to determine the goal rather than position. You'll also need to subscribe to **'/odom'** to get current position and orientation. When the callback function (**move_base** topic) publishes a new goal, it should terminate the previous goal tracking movement and move towards the next goal for a time-effective implementation.

*Note: You may need to write a **quaternion_to_yaw** function to extract theta from **data.pose.pose.orientation** from the odom callback function.*

Some helpful resources:

- http://wiki.ros.org/navigation
- http://wiki.ros.org/navigation/Tutorials/RobotSetup

Submission Requirements:

1. Include a **zip file** containing the following items:
   1.1. **All files for all Tasks inclusive of videos and screenshots wherever necessary**.
2. Submit a **LAB report** in PDF format (please do not submit word files). This report should provide explanations for all tasks performed along with screenshots, whenever suitable.