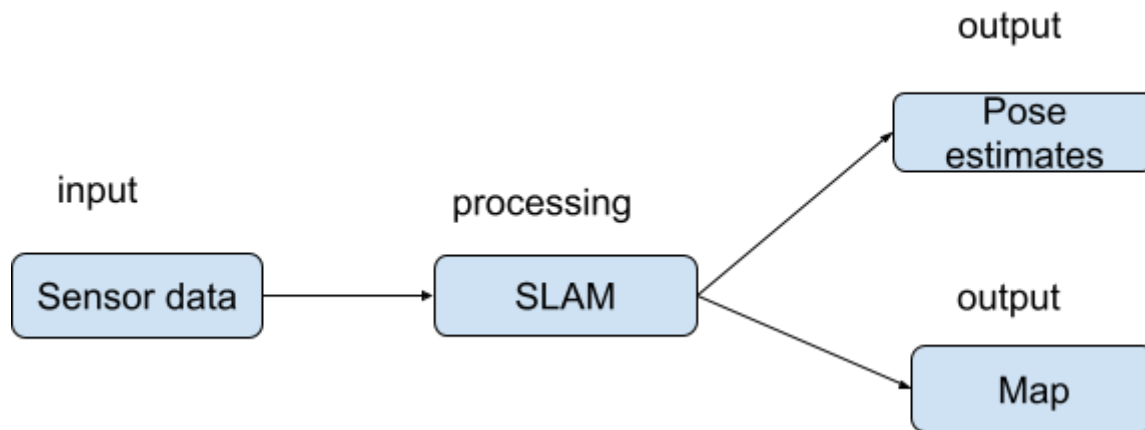


## SLAM

Simultaneous localization and mapping (SLAM) is an important technique for robot navigation. It is used to build a map in an unknown environment and simultaneously localize the robot in the map. It computes the robot poses and the map of the environment at the same time. Slam applications can be used in indoor, outdoor, in air and underwater environments for both manned and autonomous vehicles.

Examples:

- Home- vacuum cleaner
- Air- drones
- Underground- mining



**Fig1: SLAM**

**Image source: GES Created**

It has two stages:

- Localization
- Mapping

### **Localization:**

Localization is used to estimate the robot's position and orientation collectively called pose. There are different ways to track the robot's position, one way of finding the pose is by using the initial pose of the robot and orientation by wheel odometry, another way is by using the particle filter approach.

**Mapping:**

Mapping is a process of modelling the environment. Slam is used for creating the map using the sensor data by detecting the surroundings in an unknown environment. To build a map we need the laser scan data, odometry and the transform data while moving the robot in the environment.

**There are different slam algorithms to create a map:**

- **Gmapping:**

Slam\_gmapping ros node creates a 2D grid map with the help of sensor data and the pose estimates from the robot. Gmapping relies on the transform message between the starting point and the base of the robot to get the current position of the robot.

- **Cartographer:**

Cartographer has an individual configuration and launch file. The configuration file contains the information about the laser scan data, linear and angular update of the robot. Cartographer\_node ros node provides the slam 2D and 3D on multiple platforms. It is designed to work with the IMU(Inertial measurement data) but can also work with the odometry data.

- **HectorSLAM:**

The hector\_slam ros node works only with the laser data, this approach can be used without an odometry data. It uses scan matching in order to fit the current scan to the previous scan to determine the robot's movement. The map's moment is determined by the linear and angular movement of the robot.

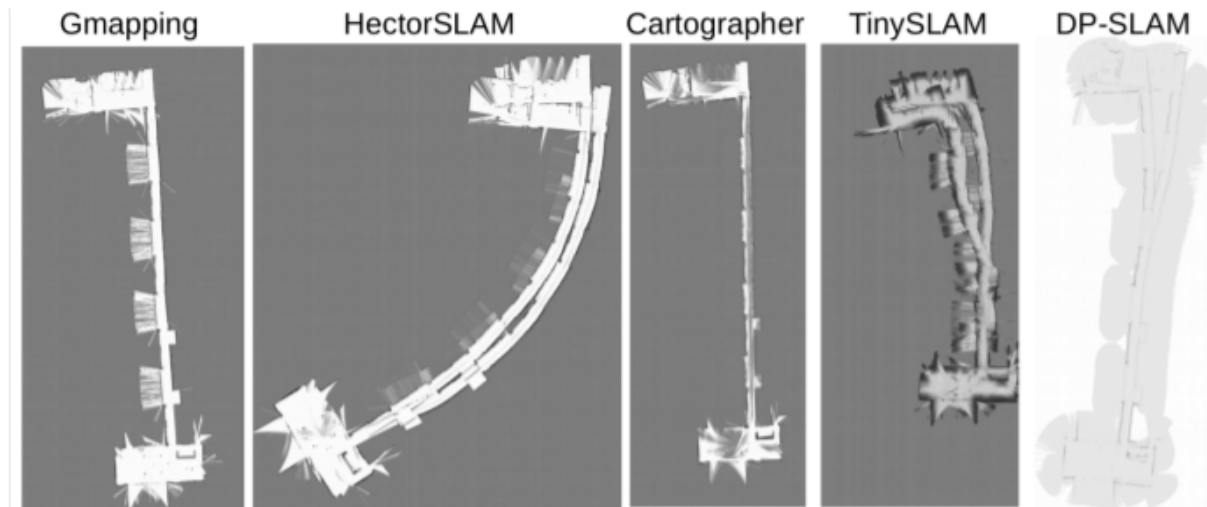
- **TinySLAM**

Tiny\_slam ros node is the simple slam algorithm. It takes in laser scan data and odometry for map creation. It is basically an ad hoc application provided for the specific robot model.

- **DP-SLAM**

DP\_Slam uses a log file to run offline. It assumes that the laser data and the odometry are written into the file at the same time.

**Following are maps created by various slam algorithms:**



**Fig 2 : Different slam algorithms**

**Image Source:** [http://www.acro.be/downloadvrij/Benchmark\\_2D\\_SLAM.pdf](http://www.acro.be/downloadvrij/Benchmark_2D_SLAM.pdf)

**Disadvantages of the few algorithms:**

- HectorSLAM the route isn't straight
- DP-SLAM gives segmentation faults few times
- Tiny SLAM the thickness of walls is more.

In conclusion HectorSLAM, DP-SLAM and Tiny SLAM algorithms are not accurate for map creation. Gmapping and cartographer algorithms are more robust and create decent maps. These algorithms handle the noise data much better than the other algorithms. Cartographer algorithm typically provides low dispersion value so it is more robust compared to other algorithms.

**Procedure to create a map using SLAM Cartographer:**

**Steps to set up turtlebot3 packages inside RB5:**

1. Creating new directory for TurtleBot 3

```
mkdir -p ~/turtlebot3_ws/src && cd ~/turtlebot3_ws/src
```

2. Clone necessary repositories & accessing TurtleBot Folder

```
git clone -b ros2 https://github.com/ROBOTIS-GIT/hls_ifcd_lds_driver.git
```

```
git clone -b ros2 https://github.com/ROBOTIS-GIT/turtlebot3_msgs.git
```

```
git clone -b ros2 https://github.com/ROBOTIS-GIT/turtlebot3.git
```

```
git clone -b ros2 https://github.com/ROBOTIS-GIT/DynamixelSDK.git
```

```
cd ~/turtlebot3_ws/src/turtlebot3
```

3. Source the TurtleBot3 Setup file

```
source /opt/ros/dashing/setup.bash
```

4. Build the TurtleBOT packages

```
cd ~/turtlebot3_ws/
```

```
colcon build
```

### Steps to set up host for RVIZ

1. Install colcon:

```
sudo apt install python3-colcon-common-extensions
```

Install cartographer:

```
sudo apt install ros-dashing-cartographer
```

```
sudo apt install ros-dashing-cartographer-ros
```

2. Install Turtlebot3 packages:

```
mkdir -p ~/turtlebot3_ws/src
```

```
cd ~/turtlebot3_ws
```

```
wget https://raw.githubusercontent.com/ROBOTIS-GIT/turtlebot3/ros2/turtlebot3.repos
```

```
vcs import src < turtlebot3.repos
```

```
colcon build --symlink-install
```

### Steps to create a map using cartographer:

1. In turtlebot3 run the bring-up command to start the applications

```
export TURTLEBOT3_MODEL=burger
```

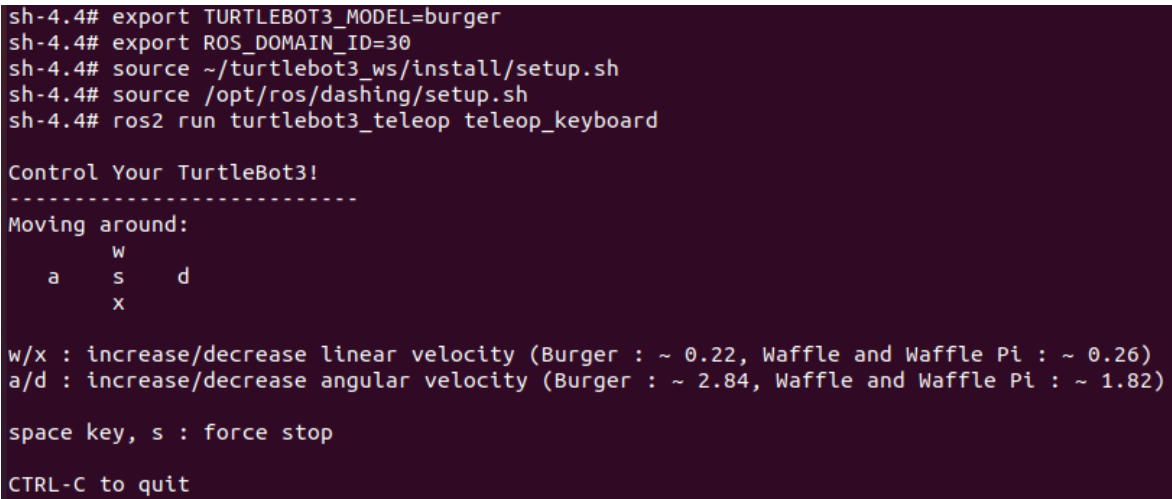
```
ros2 launch turtlebot3_bringup robot.launch.py
```

2. Launch the slam file in remote pc

```
ros2 launch turtlebot3_cartographer cartographer.launch.py
```

3. Run the teleop command in remote pc

```
ros2 run turtlebot3_teleop teleop_keyboard
```



```
sh-4.4# export TURTLEBOT3_MODEL=burger
sh-4.4# export ROS_DOMAIN_ID=30
sh-4.4# source ~/turtlebot3_ws/install/setup.sh
sh-4.4# source /opt/ros/dashing/setup.sh
sh-4.4# ros2 run turtlebot3_teleop teleop_keyboard

Control Your TurtleBot3!
-----
Moving around:
      w
  a   s   d
      x

w/x : increase/decrease linear velocity (Burger : ~ 0.22, Waffle and Waffle Pi : ~ 0.26)
a/d : increase/decrease angular velocity (Burger : ~ 2.84, Waffle and Waffle Pi : ~ 1.82)

space key, s : force stop

CTRL-C to quit
```

**Fig4 : TurtleBot3 Control Command**

**Image Source: GES Created**

### **Control the TurtleBot**

You can control the TurtleBot using W, S, A, D & X keys , after movement you can able to see the linear & angular velocity in the terminal

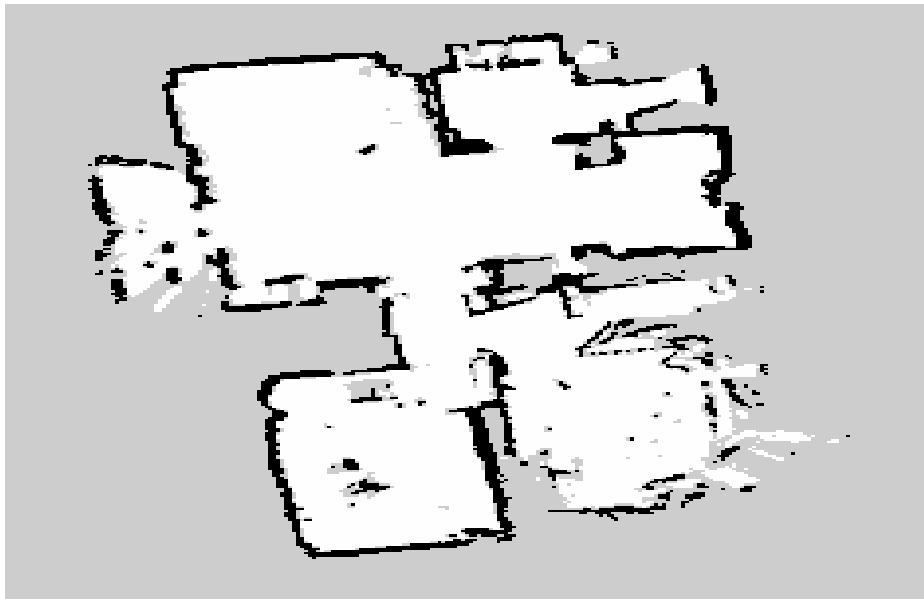
4. Launch the **map\_saver** node in the nav2\_map\_server package to create map files. Save the map.

Two map files gets created:

map.pgm file - This is an image file that represents the blueprint of the environment.

map.yaml file - This configuration file gives the meta information about the map.

```
ros2 run nav2_map_server map_saver -f ~/map
```



**Fig 5 : Created a MAP file**

**Image Source: GES Created**