slam_toolbox mapping

This lesson uses a Raspberry Pi 5 as an example.

For Raspberry Pi and Jetson Nano boards, you need to open a terminal on the host computer and enter the command to enter the Docker container. Once inside the Docker container, enter the commands mentioned in this lesson in the terminal. For instructions on entering the Docker container from the host computer, refer to **[01. Robot Configuration and Operation Guide] -- [5.Enter Docker (For JETSON Nano and RPi 5)]**.

For Orin boards, simply open a terminal and enter the commands mentioned in this lesson.

1. Introduction to slam-toolbox

slam-toolbox is an open-source 2D SLAM (Simultaneous Localization and Mapping) algorithm package based on ROS. It is primarily used for mapping and localization of mobile robots in unknown environments. Based on the Graph Optimization framework, it combines sensor data from laser lidar (such as 2D LiDAR) and inertial measurement units (IMUs) to construct a 2D grid map of the environment and update the robot's position and pose in real time.

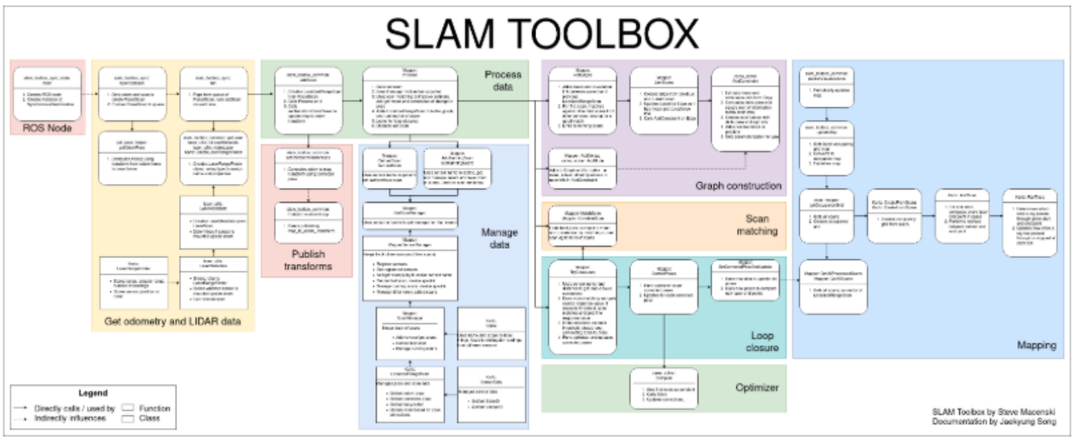
1.1 Core Technology Principles

1.1.1 Front-end Processing (Laser Scan Matching)

- **GICP Algorithm**: Estimates the robot's relative pose change by iteratively calculating the optimal match between the current laser scan and the map. Compared to traditional ICP, GICP introduces a probabilistic model, making it more robust to outliers.
- **Motion Compensation**: Utilizes IMU data to compensate for laser scan distortion during robot motion, improving matching accuracy in dynamic environments.

1.1.2 Back-end Optimization (Graph Optimization)

- **Pose Graph Construction**: Constructs a graph structure from the robot's pose nodes and the relative pose constraints obtained from scan matching.
- **Global Optimization**: Optimizes the pose graph using a nonlinear optimization algorithm (such as the Ceres Solver) to eliminate accumulated errors and achieve global consistency of the map.



GitHub Project Address: https://github.com/SteveMacenski/slam_toolbox

2. Program Functionality

After running the program, the map creation interface will appear in rviz. Use the keyboard or gamepad to control the robot's movements until the map is created. Then run the save map command to save the map. **(For the Ackermann-type robot, if you want to create a good map, try adding some stops when turning, rather than making a sharp turn.)**

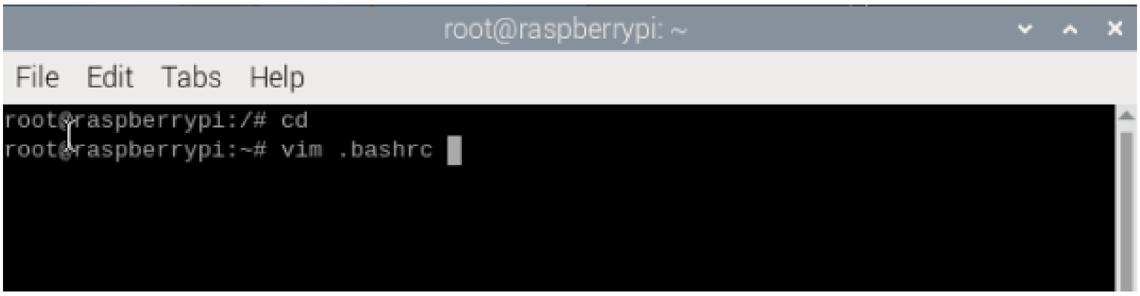
3. Pre-Use Configuration

This robot is equipped with a USB camera, a depth camera, and two different types of lidar. However, since it cannot automatically identify the robot, you need to manually set the robot type and lidar model.

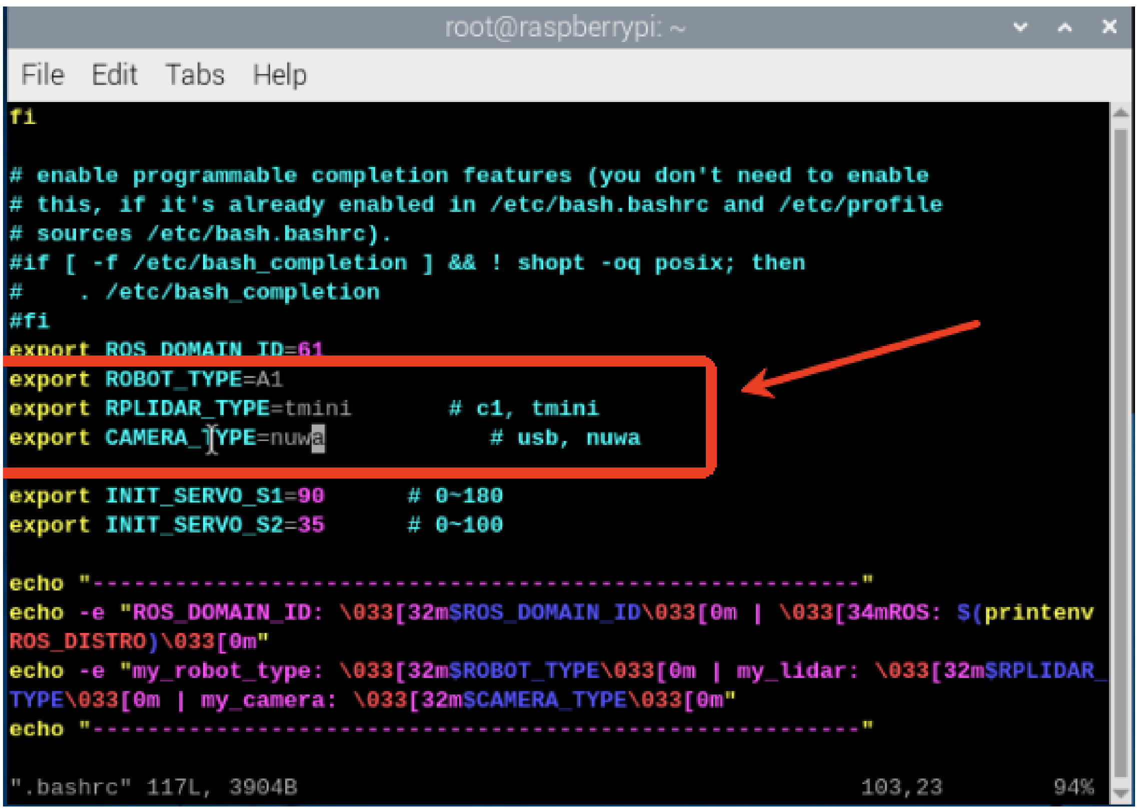
For the Raspberry Pi 5 controller, you need to first enter the Docker container. The Orin does not require this.

Change the following settings based on the car model, lidar type, and camera type.

```
root@ubuntu:/# cd
root@ubuntu:~# vim .bashrc
```

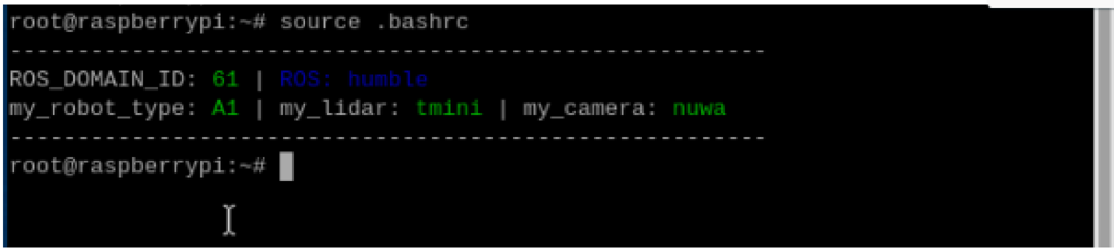


Find this location and press i on your keyboard to change the settings to the corresponding camera and lidar models. The default settings are tmini and nuwa.



After completing the changes, save and exit vim, then execute:

```
root@raspberrypi:~# source .bashrc
--------------------------------------------------------
ROS_DOMAIN_ID: 61 | ROS: humble
my_robot_type: A1 | my_lidar: tmini | my_camera: nuwa
--------------------------------------------------------
root@raspberrypi:~#
```



4. Program Startup

4.1. Startup Commands

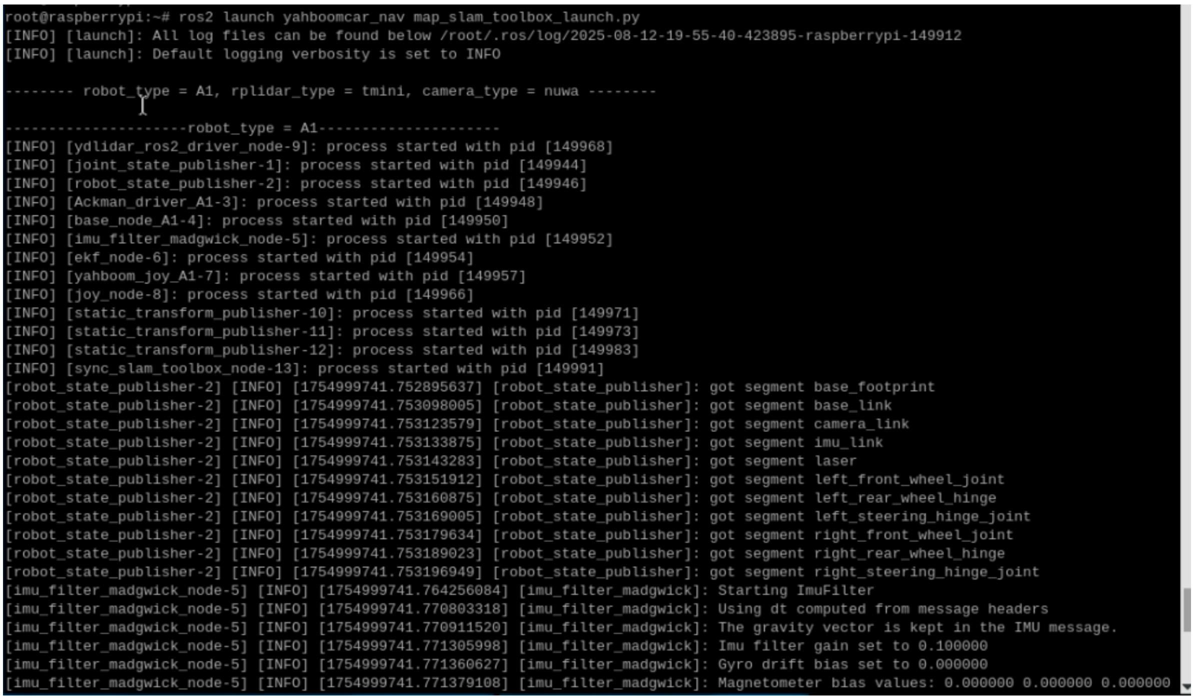
For the Raspberry Pi 5 controller, you must first enter the Docker container. For the Orin controller, this is not necessary.

Enter the Docker container (for steps, see [Docker Course] --- [4. Docker Startup Script]).

All the following commands must be executed from the Docker terminal within the same Docker container.**(For steps, see [Docker Course] --- [3. Docker Submission and Multi-Terminal Access]).**

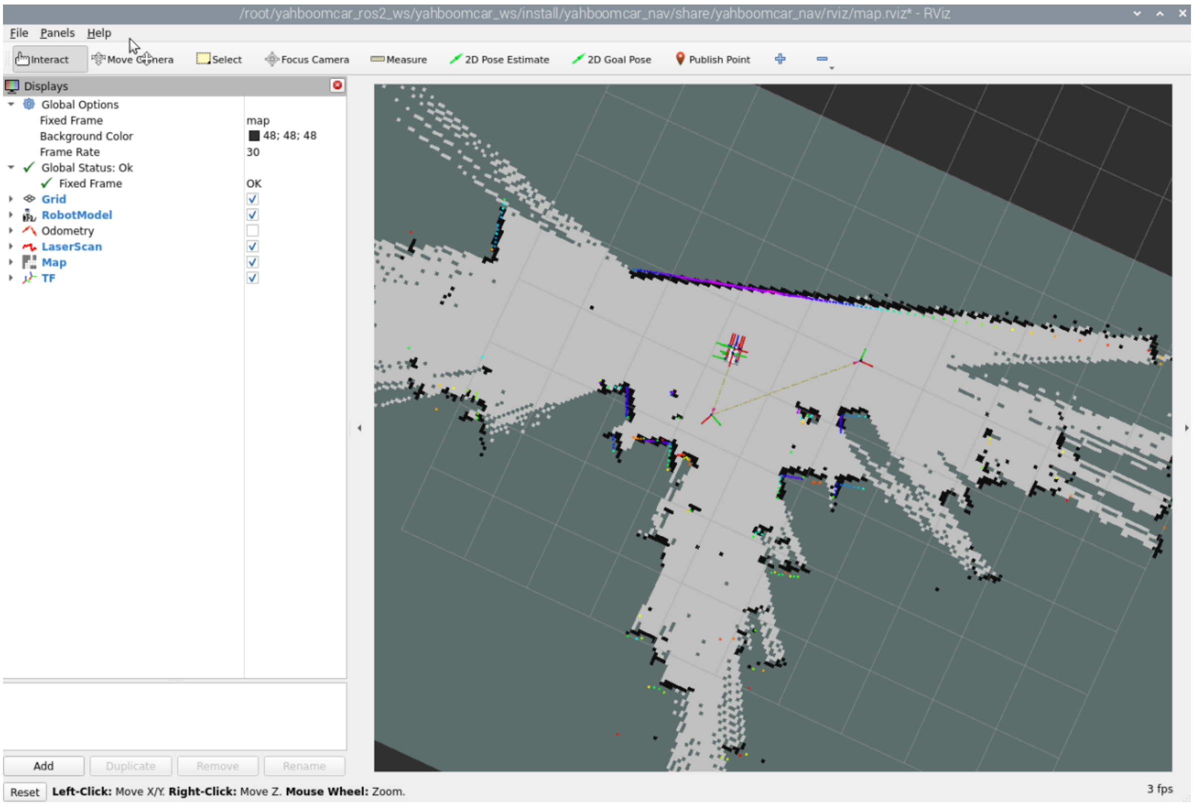
Start Mapping

```
ros2 launch yahboomcar_nav map_slam_toolbox_launch.py
```



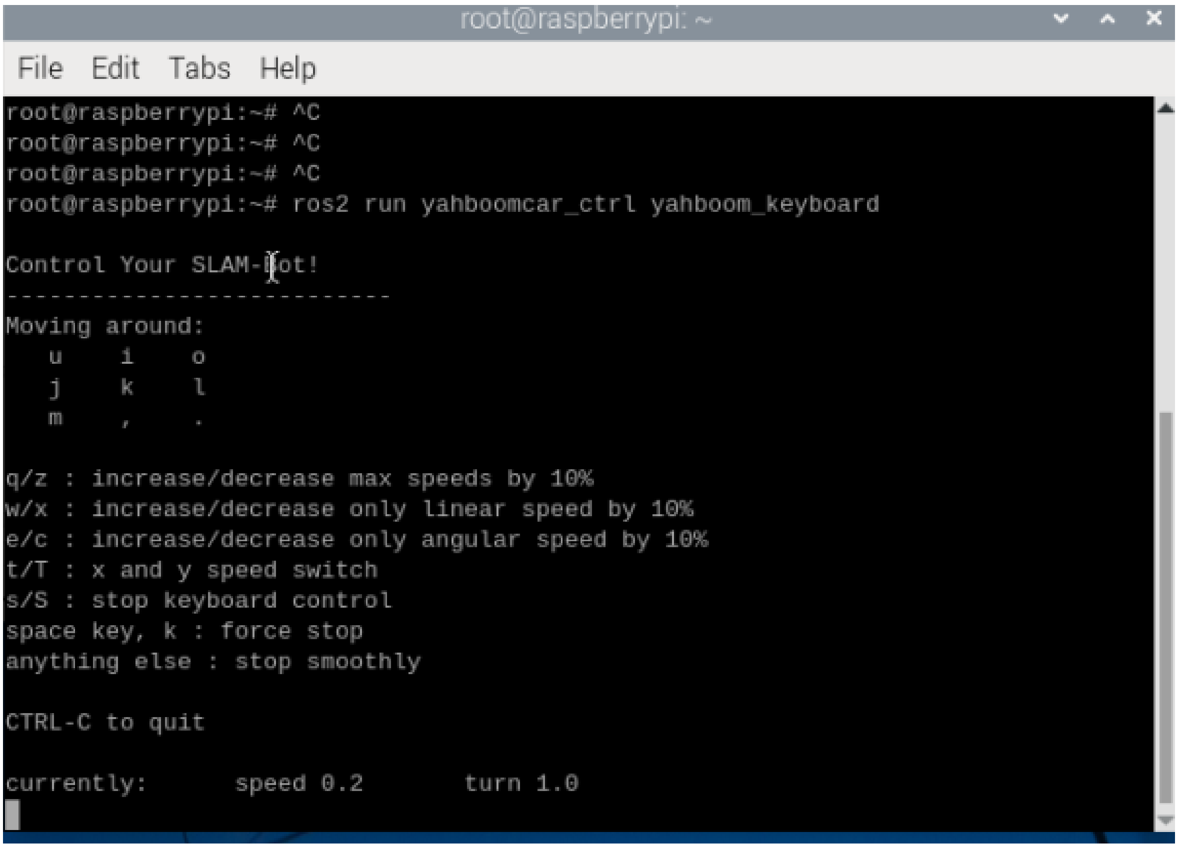
Enter the command to start rviz visualization mapping.

```
ros2 launch yahboomcar_nav display_map_launch.py
```

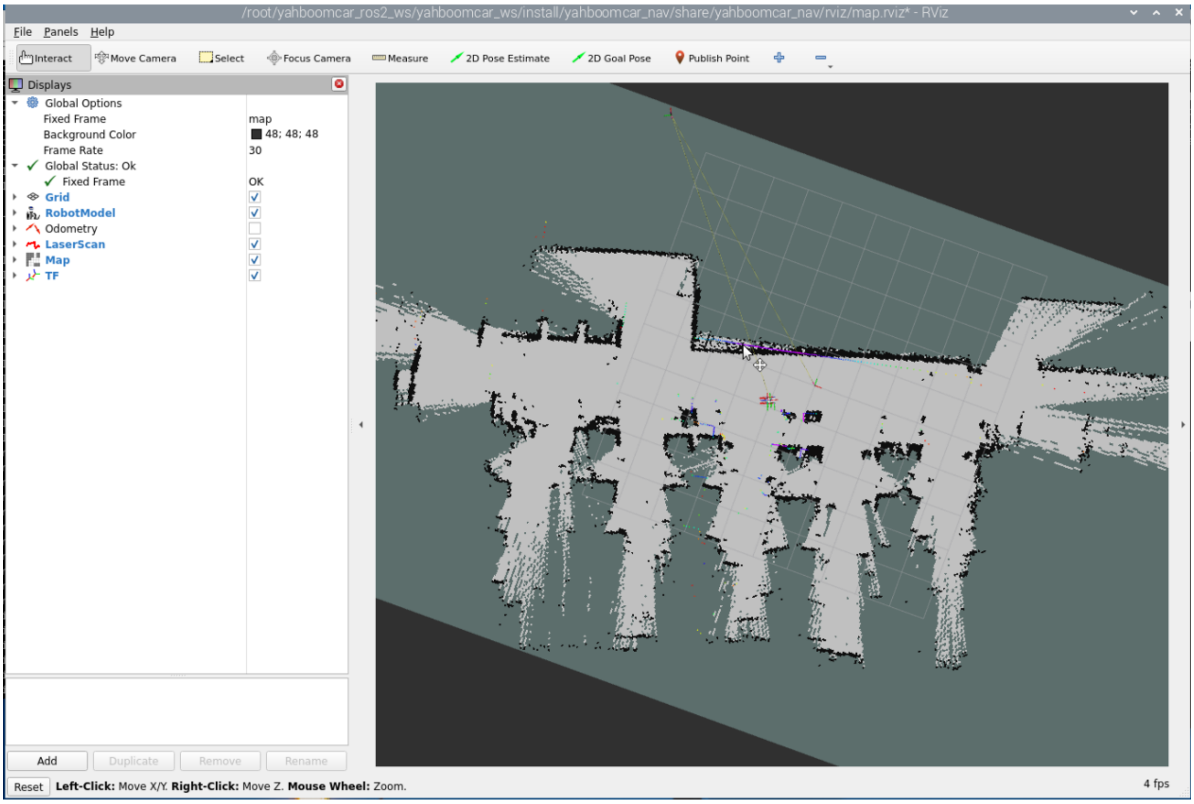


The program has controller control enabled by default. If you're using a controller, you can now connect it directly to the receiver for control. To control it using the keyboard, enter the following in the terminal:

```
#keyboard
ros2 run yahboomcar_ctrl yahboom_keyboard
```

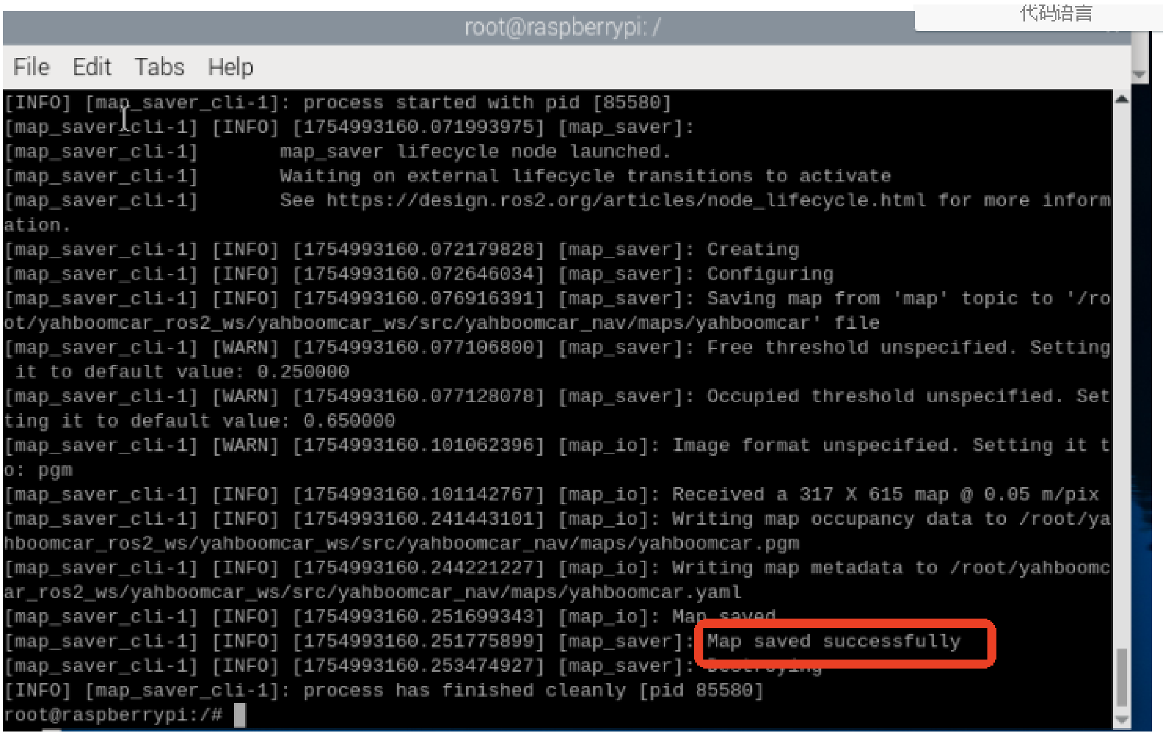


Then control the car and slowly navigate the area to be mapped.



After mapping is complete, enter the following command to save the map. In the terminal, enter:

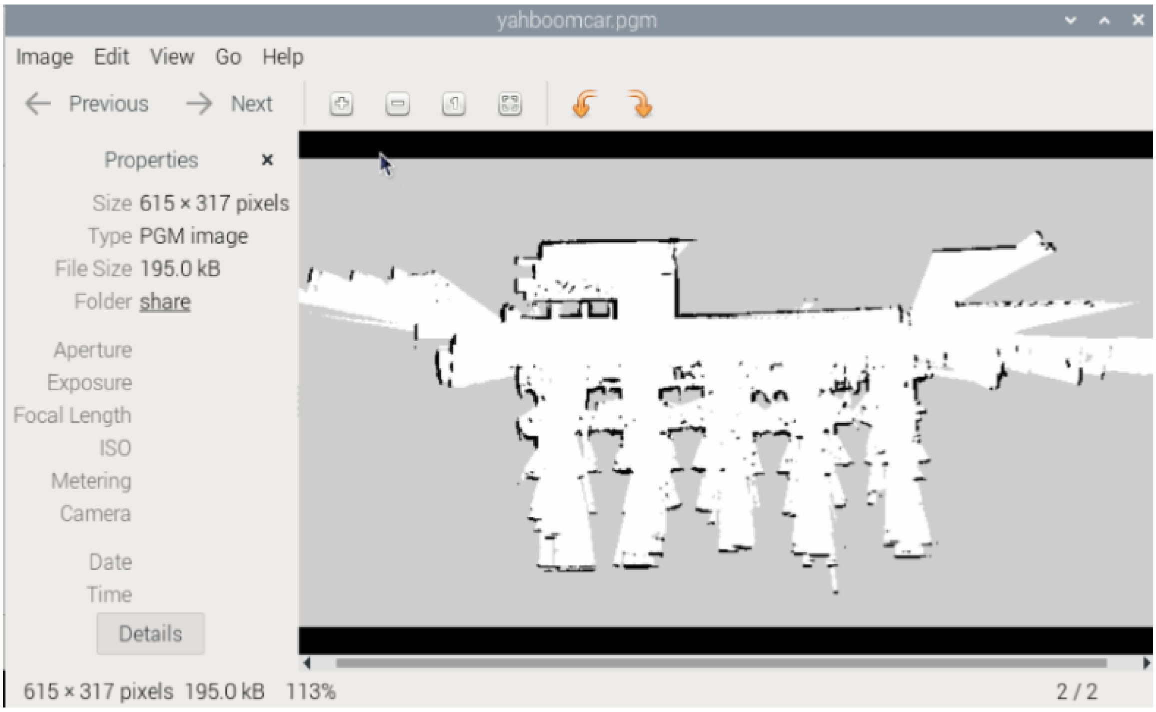
```
ros2 launch yahboomcar_nav save_map_launch.py
```



A map named yahboomcar will be saved. This map is saved in:

```
~/yahboomcar_ros2_ws/yahboomcar_ws/src/yahboomcar_nav/maps/yahboomcar.yaml
```

Two files will be generated, one named yahboomcar.pgm. The map is as follows (for Orin boards, you can directly double-click the map. For Raspberry Pi boards, you need to copy the file to /root/share in Docker). (This is the shared folder, then view it on the host machine.)



yahboom_map.yaml, take a look at the contents of the yaml file.

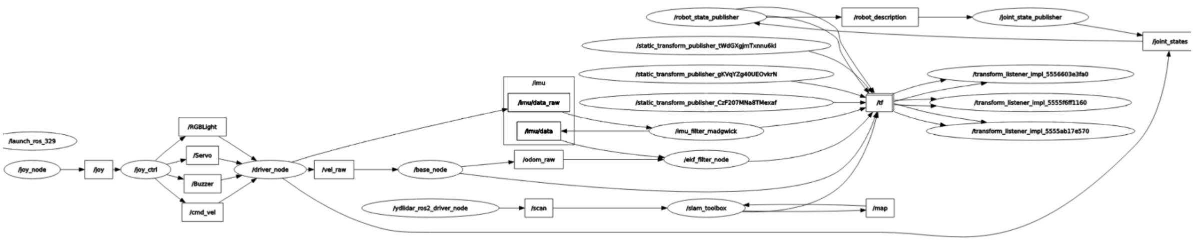
```
image: yahboomcar.pgm
mode: trinary
resolution: 0.05
origin: [-9.02, -15.5, 0]
negate: 0
occupied_thresh: 0.65
free_thresh: 0.25
```

- image: The image representing the map, i.e., yahboomcar.pgm
- mode: This property can be one of trinary, scale, or raw, depending on the selected mode. Trinary is the default mode.
- resolution: The resolution of the map, in meters/pixels
- origin: The 2D coordinate system at the bottom left corner of the map. Pose (x, y, yaw), where yaw is counterclockwise rotation (yaw=0 means no rotation). Currently, many parts of the system ignore the yaw value.
- negate: Whether to invert the meaning of white/black and free/occupied (this does not affect the interpretation of thresholds).
- occupied_thresh: Pixels with an occupied probability greater than this threshold are considered fully occupied.
- free_thresh: Pixels with an occupied probability less than this threshold are considered completely free.

5. View the Node Communication Graph

In the terminal, enter:

```
ros2 run rqt_graph rqt_graph
```

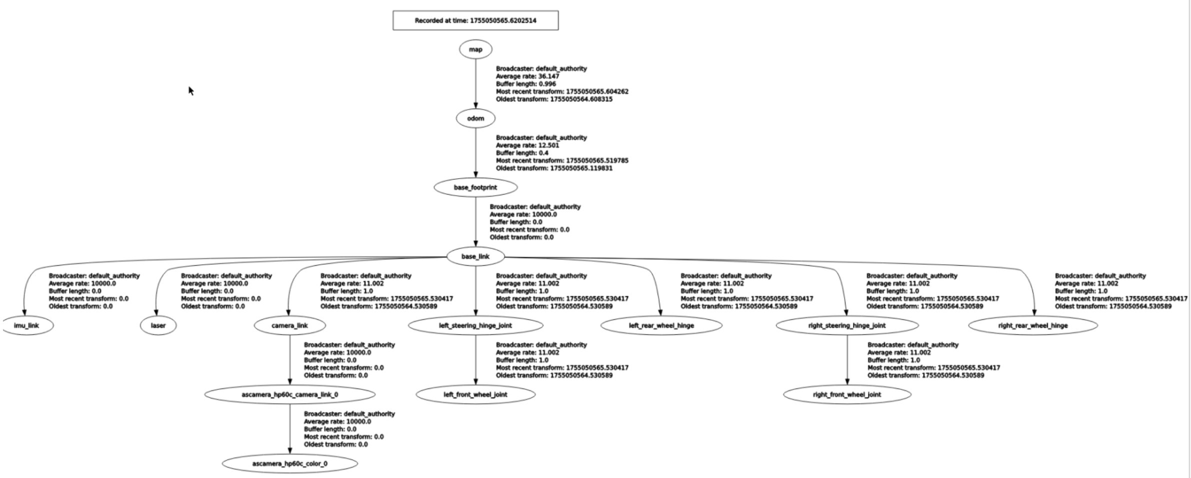


6. View the TF Tree

In the terminal, enter:

```
ros2 run rqt_tf_tree rqt_tf_tree
```

After the program finishes running, a TF conversion interface will appear.



7. slam-toolbox Node Details

```
ros2 node info /slam_toolbox
```

Enter the above command in the terminal to view the subscription and publication topics related to the gmapping node.

```
/slam_toolbox
  Subscribers:
    /map: nav_msgs/msg/OccupancyGrid
    /parameter_events: rcl_interfaces/msg/ParameterEvent
    /scan: sensor_msgs/msg/LaserScan
    /slam_toolbox/feedback: visualization_msgs/msg/InteractiveMarkerFeedback
  Publishers:
    /map: nav_msgs/msg/OccupancyGrid
    /map_metadata: nav_msgs/msg/MapMetaData
    /parameter_events: rcl_interfaces/msg/ParameterEvent
    /pose: geometry_msgs/msg/PoseWithCovarianceStamped
    /rosout: rcl_interfaces/msg/Log
    /slam_toolbox/graph_visualization: visualization_msgs/msg/MarkerArray
```

```
    /slam_toolbox/scan_visualization: sensor_msgs/msg/LaserScan
    /slam_toolbox/update: visualization_msgs/msg/InteractiveMarkerUpdate
    /tf: tf2_msgs/msg/TFMessage
  Service Servers:
    /slam_toolbox/clear_changes: slam_toolbox/srv/Clear
    /slam_toolbox/describe_parameters: rcl_interfaces/srv/DescribeParameters
    /slam_toolbox/deserialize_map: slam_toolbox/srv/DeserializePoseGraph
    /slam_toolbox/dynamic_map: nav_msgs/srv/GetMap
    /slam_toolbox/get_interactive_markers:
visualization_msgs/srv/GetInteractiveMarkers
    /slam_toolbox/get_parameter_types: rcl_interfaces/srv/GetParameterTypes
    /slam_toolbox/get_parameters: rcl_interfaces/srv/GetParameters
    /slam_toolbox/list_parameters: rcl_interfaces/srv/ListParameters
    /slam_toolbox/manual_loop_closure: slam_toolbox/srv/LoopClosure
    /slam_toolbox/pause_new_measurements: slam_toolbox/srv/Pause
    /slam_toolbox/save_map: slam_toolbox/srv/SaveMap
    /slam_toolbox/serialize_map: slam_toolbox/srv/SerializePoseGraph
    /slam_toolbox/set_parameters: rcl_interfaces/srv/SetParameters
    /slam_toolbox/set_parameters_atomically:
rcl_interfaces/srv/SetParametersAtomically
    /slam_toolbox/toggle_interactive_mode: slam_toolbox/srv/ToggleInteractive
  Service Clients:

  Action Servers:

  Action Clients:
```