10. rrt_exploration mapping algorithm

rrt_exploration： http://wiki.ros.org/rrt_exploration

rrt_exploration/Tutorials： http://wiki.ros.org/rrt_exploration/Tutorials

map_server： https://wiki.ros.org/map_server

10.1. Introduction

RRT exploration is a search algorithm implemented based on the RRT path planning algorithm. The reason why the RRT algorithm is used is that RRT has a strong tendency for unknown areas. In RRT exploration, RRT is mainly used to generate boundary points, which is very beneficial for exploring boundary points. The so-called boundary point is the intersection point between the explored and unknown areas. Here is a definition: for all areas, if they have been explored, the area without obstacles is recorded as 0, and the area with obstacles is recorded as 1 , the unknown area is recorded as -1, and at the beginning, the entire area is recorded as -1.

The framework of the Rapid Exploration Random Tree (RRT) algorithm is as follows:

- Global RRT frontier point detector node.
- Local RRT frontier point detector node.
- OpenCV-based frontier detector node.
- Filter node.

- Assigner node.

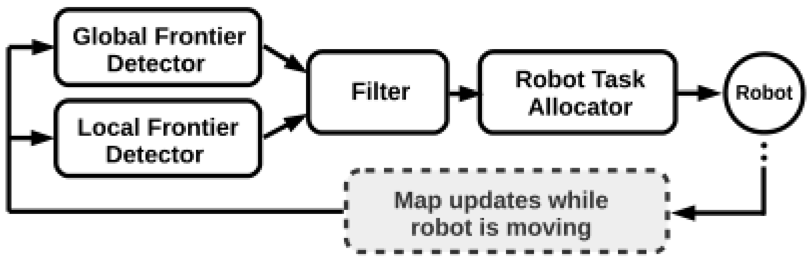


Fig. 1: Overall schematic diagram of the exploration algorithm

There are 3 types of nodes: nodes for detecting boundary points in the occupied raster map, nodes for filtering detected points, and nodes for assigning points to robots.

10.2. Use

Note: When building a map, the slower the speed, the better the effect (note that the rotation speed should be slower). If the speed is too fast, the effect will be poor.

According to different models, you only need to set the purchased model in [.bashrc], X1(ordinary four-wheel drive) X3(Mike wheel) X3plus(Mike wheel mechanical arm) R2(Ackerman differential) and so on. Section takes X3 as an example

Open the [.bashrc] file

```
sudo vim .bashrc
```

Find the [ROBOT_TYPE] parameters and modify the corresponding car model

```
export ROBOT_TYPE=X3   # ROBOT_TYPE: X1 X3 X3plus R2 X7
```

10.2.1. Start

Start the command (robot side). For the convenience of operation, this section takes [mono + laser + yahboomcar] as an example.

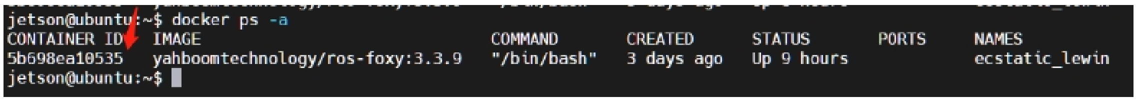
```
#You need to enter docker first, perform this step more
#If running the script to enter docker fails, please refer to 07.Docker-orin/05,
Enter the robot's docker container
~/run_docker.sh
roslaunch yahboomcar_nav laser_bringup.launch           # laser + yahboomcar
roslaunch yahboomcar_nav laser_usb_bringup.launch       # mono + laser +
yahboomcar
roslaunch yahboomcar_nav laser_astrapro_bringup.launch  # Astra + laser +
yahboomcar
```

Mapping command (robot side)

<Open another terminal and enter the same docker container

1. In the above steps, a docker container has been opened. You can open another terminal on the host (car) to view:

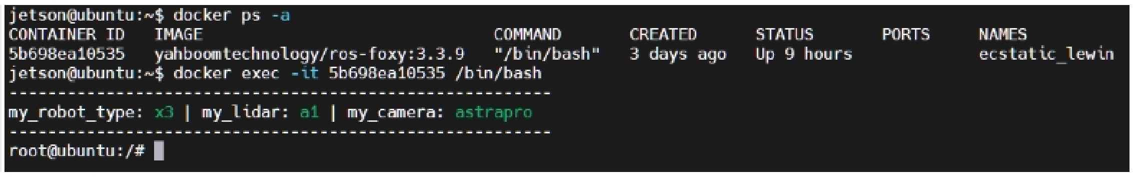
```
docker ps -a
```



```
jetson@ubuntu:~$ docker ps -a
CONTAINER ID   IMAGE                              COMMAND       CREATED      STATUS      PORTS     NAMES
5b698ea10535   yahboomtechnology/ros-foxy:3.3.9   "/bin/bash"   3 days ago   Up 9 hours            ecstatic_lewin
jetson@ubuntu:~$
```

2. Now enter the docker container in the newly opened terminal:

```
docker exec -it 5b698ea10535 /bin/bash
```



```
jetson@ubuntu:~$ docker ps -a
CONTAINER ID   IMAGE                              COMMAND       CREATED      STATUS      PORTS     NAMES
5b698ea10535   yahboomtechnology/ros-foxy:3.3.9   "/bin/bash"   3 days ago   Up 9 hours            ecstatic_lewin
jetson@ubuntu:~$ docker exec -it 5b698ea10535 /bin/bash
-----------------------------------------------------------
my_robot_type: x3 | my_lidar: a1 | my_camera: astrapro
-----------------------------------------------------------
root@ubuntu:/#
```

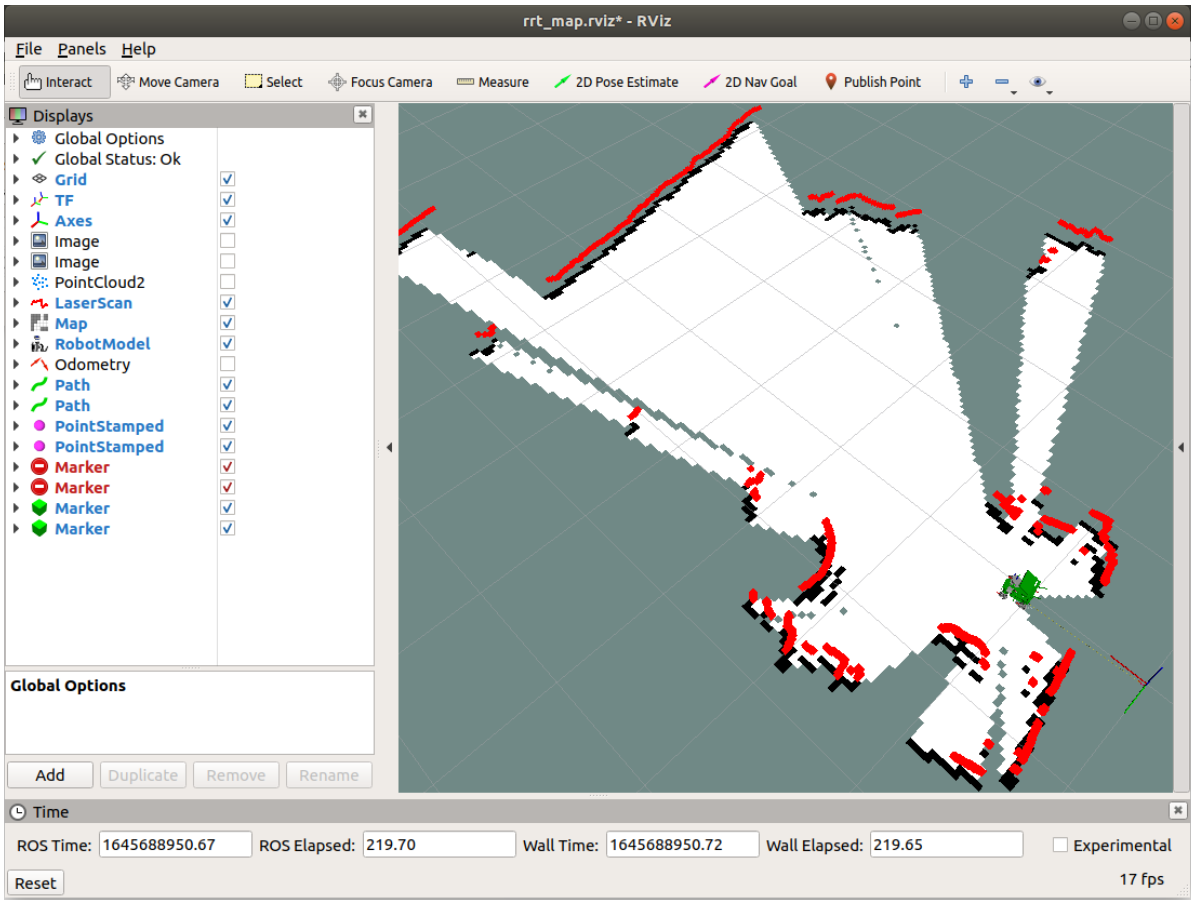
After successfully entering the container, you can open countless terminals to enter the container.

```
roslaunch yahboomcar_nav rrt_exploration.launch use_rviz:=false
```

- 【use_rviz】parameter: whether to enable rviz visualization.

Turn on the visual interface (virtual machine side)

```
roslaunch yahboomcar_nav view_rrt_map.launch
```



ROS Time: 1645688950.67 ROS Elapsed: 219.70 Wall Time: 1645688950.72 Wall Elapsed: 219.65

The gap at the back of the robot is due to the obstruction caused by the installation position of the display screen, so a certain range of radar data is blocked. The shielding range can be adjusted, or it can not be blocked according to the actual situation. For specific operations, see [01. Radar Basic Course].

10.2.2. Mapping

Click [Publish Point] on the [rviz] interface, and then click on the map. Every time before clicking a point on the map, you must first select [Publish Point]. Click four points **clockwise or counterclockwise**, and the last point is near the car. After selecting five points as required, use [2D Nav Gloal] to give the robot a forward speed, and the robot begins to explore and build a map.

10.2.3. Map save

After the map construction is completed, the map will be automatically saved and returned to zero point. The map is saved to the ~/yahboomcar_ws/src/yahboomcar_nav/maps folder, the name is rrt_map, and can be modified in the [simple.launch] file.

rrt_map.yaml

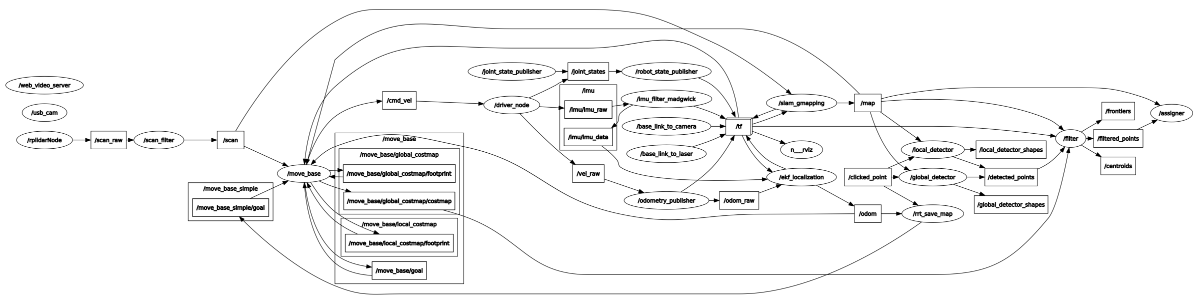
```
image: map.pgm
resolution: 0.05
origin: [-15.4,-12.2,0.0]
negate: 0
occupied_thresh: 0.65
free_thresh:  0.196
```

Parameter analysis:

- image: The path of the map file, which can be an absolute path or a relative path.
- resolution: Resolution of the map, meters/pixel
- origin: 2D pose (x, y, yaw) in the lower left corner of the map. The yaw here is rotated counterclockwise (yaw=0 means no rotation). Many parts of the current system ignore the yaw value.
- negate: Whether to reverse the meaning of white/black and free/occupied (the interpretation of the threshold is not affected)
- occupied_thresh: Pixels with an occupation probability greater than this threshold will be considered fully occupied.
- free_thresh: Pixels with occupancy probability less than this threshold will be considered completely free.

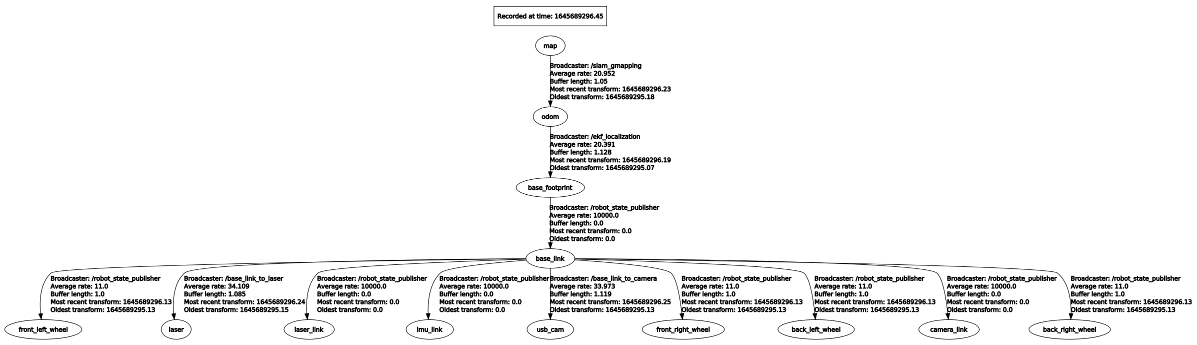
10.2.4. Node View

```
rqt_graph
```



10.2.5. View tf tree

```
rosrun rqt_tf_tree rqt_tf_tree
```


Recorded at time: 1645689296.45
map
Broadcaster: /slam_gmapping
Average rate: 20.952
Buffer length: 1.05
Most recent transform: 1645689296.23
Oldest transform: 1645689295.18
odom
Broadcaster: /ekf_localization
Average rate: 20.391
Buffer length: 1.128
Most recent transform: 1645689296.19
Oldest transform: 1645689295.07
base_footprint
Broadcaster: /robot_state_publisher
Average rate: 10000.0
Buffer length: 0.0
Most recent transform: 0.0
Oldest transform: 0.0
base_link
Broadcaster: /robot_state_publisher
Average rate: 11.0
Buffer length: 1.0
Most recent transform: 1645689296.13
Oldest transform: 1645689295.13
front_left_wheel
Broadcaster: /base_link_to_laser
Average rate: 34.109
Buffer length: 1.085
Most recent transform: 1645689296.24
Oldest transform: 1645689295.15
laser
Broadcaster: /robot_state_publisher
Average rate: 10000.0
Buffer length: 0.0
Most recent transform: 0.0
Oldest transform: 0.0
laser_link
Broadcaster: /robot_state_publisher
Average rate: 10000.0
Buffer length: 0.0
Most recent transform: 0.0
Oldest transform: 0.0
imu_link
Broadcaster: /base_link_to_camera
Average rate: 33.973
Buffer length: 1.119
Most recent transform: 1645689296.25
Oldest transform: 1645689295.13
usb_cam
Broadcaster: /robot_state_publisher
Average rate: 11.0
Buffer length: 1.0
Most recent transform: 1645689296.13
Oldest transform: 1645689295.13
front_right_wheel
Broadcaster: /robot_state_publisher
Average rate: 11.0
Buffer length: 1.0
Most recent transform: 1645689296.13
Oldest transform: 1645689295.13
back_left_wheel
Broadcaster: /robot_state_publisher
Average rate: 10000.0
Buffer length: 0.0
Most recent transform: 0.0
Oldest transform: 0.0
camera_link
Broadcaster: /robot_state_publisher
Average rate: 11.0
Buffer length: 1.0
Most recent transform: 1645689296.13
Oldest transform: 1645689295.13
back_right_wheel

10.3、global_rrt_frontier_detector

10.3.1. Introduction

The global_rrt_frontier_detector node takes the occupancy grid and finds boundary points (i.e. exploration targets) in it. It publishes detected points so that filter nodes can process them. In a multi-bot configuration, only one instance of this node is run. If needed, running additional instances of the global boundary detector can increase the speed of boundary point detection.

10.3.2 Topics and Services

Subscribe to topics	**type**	**describe**
map	nav_msgs/OccupancyGrid	The topic name is defined by the ~map_Topic parameter. It is the topic name that this node receives.
clicked_point	geometry_msgs/PointStamped	global_rrt_frontier_detector The area to be detected by the node. This topic is where nodes receive five points that define an area. The first four points are the four points that define the square area to be explored, and the last point is the starting point of the tree. After announcing these five points, RRT will start detecting border points.
Post a topic	**type**	**describe**
detected_points	geometry_msgs/PointStamped	Topics to publish detected boundary points.
shapes	visualization_msgs/Marker	Publish RRT imaginary line styles for viewing with Rviz.

10.3.3. Configuration parameters

parameter	type	defaults	describe
~map_topic	string	"/robot_1/map"	This node receives the map topic map name
~eta	float	5.0	This parameter controls the growth rate of the RRT used to detect boundary points, in meters. This parameter should be set according to the map size, very large values will cause the tree to grow faster and thus detect boundary points faster, but a large growth rate will also mean that the tree will lose small corner points in the map

10.4、local_rrt_frontier_detector

10.4.1. Introduction

This node is similar to global_rrt_frontier_detector. However, it works differently because the tree here is constantly reset every time a boundary point is detected. This node will run along the global boundary detector node, which is responsible for quickly detecting boundary points located near the robot.

In a multi-robot configuration, each robot runs a local detector instance. So, for a team of 3 robots, there will be 4 nodes for detecting boundary points; 3 local detectors and 1 global detector. If needed, running additional instances of the local boundary detector can increase the speed of boundary point detection. All detectors will publish detected boundary points ("/detected_points") on the same topic.

10.4.2. Topics and services

Subscribe to topics	type	describe
map	nav_msgs/OccupancyGrid	The name of the map topic subscribed to by this node.
clicked_point	geometry_msgs/PointStamped	Similar to the global_rrt_frontier_detector node
Post a topic	**type**	**describe**
detected_points	geometry_msgs/PointStamped	Post the detected boundary points to the topic.
shapes	visualization_msgs/Marker	Publish RRT imaginary line styles for viewing with Rviz.

10.4.3. Configuration parameters

parameter	type	defaults	describe
~/robot_1/base_link	string	"/robot_1/base_link"	Frame connected to the robot. Each time the RRT tree is reset, it will start from the current robot position taken from this coordinate system.
~map_topic	string	"/robot_1/map"	This node receives the map topic map name
~eta	float	5.0	This parameter controls the growth rate of the RRT used to detect boundary points, in meters. This parameter should be set according to the map size, very large values will cause the tree to grow faster and thus detect boundary points faster, but a large growth rate will also mean that the tree will lose small corner points in the map

10.5、frontier_opencv_detector

10.5.1. Introduction

This node is another boundary detector, but it is not based on RRT. This node uses OpenCV tools to detect boundary points. It is designed to run standalone, and in a multi-bot configuration, only one instance should be run (running additional instances of this node will not make any difference).

Initially, this node was implemented for comparison with RRT-based boundary detectors. Running this node along RRT detectors (local and global) improves the speed of boundary point detection.

Note: You can run any type and number of detectors, all detectors will be published on the same topic to which the filter node (explained in the next section) is subscribed. The filter, on the other hand, passes the filtered boundary points to the allocator in order to command the robot to explore these points.

10.5.2. Topics and services

Subscribe to topics	type	describe
map	nav_msgs/OccupancyGrid	The name of the map topic subscribed to by this node.
Post a topic	**type**	**describe**
detected_points	geometry_msgs/PointStamped	Topics to which detected boundary points are published.
shapes	visualization_msgs/Marker	Publish RRT imaginary line styles for viewing with Rviz.

10.5.3. Configuration parameters

parameter	type	defaults	describe
~map_topic	string	"/robot_1/map"	This node receives the map topic map name

10.6、filter

10.6.1. . Introduction

This Node node receives detected boundary points from all detectors, filters these points, and passes them to the distribution node to command the robot. Filtering includes deleting old and invalid points, as well as deleting redundant points.

10.6.2. Topics and Services

Subscribe to topics	type	describe
map	nav_msgs/OccupancyGrid	The topic name is defined by the ~map_Topic parameter. It is the topic name that this node receives.
robot_x/move_base_node /global_costmap/costmap	nav_msgs/OccupancyGrid	x is the number of the robot. This node subscribes to the topic of all costmaps for all robots and therefore requires a costmap. Normally, the cost map should be published by navigation (after turning navigation on the robot, each robot will have a cost map). For example, if n_robots=2, the nodes will subscribe to: robot_1/move_base_node/global_costmap/costmap and robot_2/move_base_node/global_costmap/costmap. The cost map is used to remove invalid points. NOTE: The namespace of all nodes corresponding to robots should start with robot_x.
detected_points	geometry_msgs/PointStamped	The topic name defined by ~goals_topic. It is the topic where filter nodes receive boundary detection points.
Post a topic	**type**	**describe**
frontiers	visualization_msgs/Marker	The filter node publishes only the topics of filtered boundary points.
centroids	visualization_msgs/Marker	The filter node publishes the topic of the received boundary point.
filtered_points	MsgLink(msg/type)	All filtered points are sent to this topic's assigner node as a point array.

10.6.3. Configuration parameters

parameter	type	defaults	describe
~map_topic	string	"/robot_1/map"	This node receives the map topic map name
~costmap_clearing_threshold	float	70.0	Cost map cleanup threshold
~info_radius	float	1.0	The information radius used to calculate the information gain of boundary points.
~goals_topic	string	/detected_points	Defines the topic for nodes to receive detected boundary points
~n_robots	float	1.0	Number of robots
~namespace	string		Namespaces
~namespace_init_count	float	1.0	Namespace index
~rate	float	100.0	Node cycle rate in Hz.

10.7、Assigner

10.7.1. Introduction

This node receives target detection targets, i.e. filtered boundary points published by the filtering node, and commands the robot accordingly. The evaluator node commands the robot through move_base_node. This is why navigation is enabled on the robot.

10.7.2. Topics and services

Post a topic	type	describe
map	nav_msgs/OccupancyGrid	The topic name is defined by the ~map_Topic parameter. It is the topic name that this node receives.
frontiers_topic	nav_msgs/OccupancyGrid	The topic name is defined by the ~frontiers_topic parameter

10.7.3. Configuration parameters

parameter	type	defaults	describe
~map_topic	string	"/robot_1/map"	This node receives the map topic map name
~info_radius	float	1.0	The information radius used to calculate the information gain of boundary points.
~info_multiplier	float	3.0	Unit is meter. This parameter is used to emphasize the importance of the information gain of a boundary point relative to the cost (expected travel distance to the boundary point).
~hysteresis_radius	float	3.0	Unit is meter. This parameter defines the hysteresis radius.
~hysteresis_gain	float	2.0	Unit is meter. This parameter defines the lag gain.
~frontiers_topic	string	/filtered_points	This node receives the topic of the boundary point.
~n_robots	float	1.0	Number of robots
~namespace	string		Namespaces
~namespace_init_count	float	1.0	Starting indexing of robot names.
~delay_after_assignement	float	0.5	The unit is seconds. It defines the amount of delay after each robot allocation.
~global_frame	string	"/map"	Used for global coordinate systems. In a single robot, it is the same as the "map_topic" parameter. In the case of multiple robots, the coordinate system name corresponds to the global coordinate system name.