11. AMCL adaptive Monte Carlo positioning algorithm

11.1. Introduction

The full English name of amcl is adaptive Monte Carlo localization, which is a probabilistic positioning system for two-dimensional mobile robots. In fact, it is an upgraded version of the Monte Carlo positioning method, using the adaptive KLD method to update particles, and using particle filters to track the robot's posture based on known maps. As currently implemented, this node only works with laser scans and laser maps. It can be extended to handle other sensor data. amcl receives laser-based maps, laser scans, and transformation information, and outputs pose estimates. On startup, amcl initializes its particle filter based on the provided parameters. Note that due to the default settings, if no parameters are set, the initial filter state will be a medium-sized particle cloud centered at (0,0,0).

Monte Carlo

- Monte Carlo method is also called statistics [simulation method] (https://baike.baidu.com/item/simulation method/9794774), and statistical experimental method is an idea or method. It is a numerical simulation method that takes probability phenomena as the research object. It is a calculation method based on the sampling survey method to obtain statistical values to estimate unknown characteristic quantities.
- Example: There is an irregular shape inside a rectangle. How to calculate the area of the irregular shape? Not easy to calculate. But we can approximate. Take a bunch of beans, scatter them evenly on the rectangle, and then count the number of beans in the irregular shape and the number of beans in the remaining places. We know the area of rectangles, so we estimate the area of irregular shapes. Taking robot positioning as an example, it may be at any position on the map. In this case, how do we express the confidence of a position? We also use particles. Where there are more particles, it means that the robot is more likely to be there.

The biggest advantage of Monte Carlo method

- The error of the method has nothing to do with the dimensionality of the problem.
- Problems of statistical nature can be solved directly.
- There is no need to discretize the problem of continuity

Disadvantages of Monte Carlo method

- Deterministic problems need to be transformed into stochastic problems.
- Errors are probabilistic errors.
- Usually requires a larger number of calculation steps N.

particle filter

-The number of particles represents the probability of something. Through some evaluation method (the possibility of evaluating this thing), the distribution of particles is changed. For example, in robot positioning, for a certain particle A, I think there is a high possibility that this particle is at this coordinate (for example, this coordinate belongs to "this thing" mentioned before), then I will give it a high score. Next time you rearrange the positions of all the particles, arrange more near this position. After a few more rounds of this, the particles will all be concentrated in positions with high probability.

Adaptive Monte Carlo

- Solved the problem of robot kidnapping. It will re-sprinkle some particles globally when it finds that the average score of particles suddenly decreases (meaning that the correct particles were abandoned in a certain iteration).
- Solved the problem of fixed particle number, because sometimes when the robot is almost positioned, for example, the particles are all concentrated together, it is not necessary to maintain so many particles. At this time, the number of particles can be reduced.

11.2. Navigation and positioning test

According to different models, you only need to set the purchased model in [.bashrc], X1 (normal four-wheel drive) X3 (Mailun) X3 as an example.

```
#Raspberry Pi 5 master needs to enter docker first, please perform this step
#If running the script into docker fails, please refer to ROS/07, Docker tutorial
~/run_docker.sh
```

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
```

11.2.1. Start

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

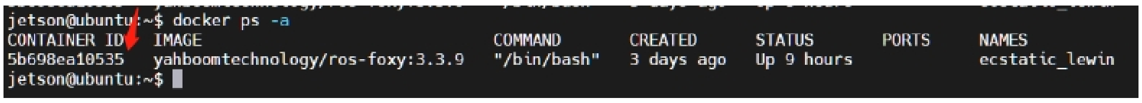
```
roslaunch yahboomcar_nav laser_bringup.launch # laser + yahboomcar
roslaunch yahboomcar_nav laser_usb_bringup.launch # mono + laser + yahboomcar
roslaunch yahboomcar_nav laser_astrapro_bringup.launch # Astra + laser +
yahboomcar
```

Start the navigation obstacle avoidance function (robot side), set parameters and modify the launch file according to needs.

<PI5 needs to open another terminal to 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:

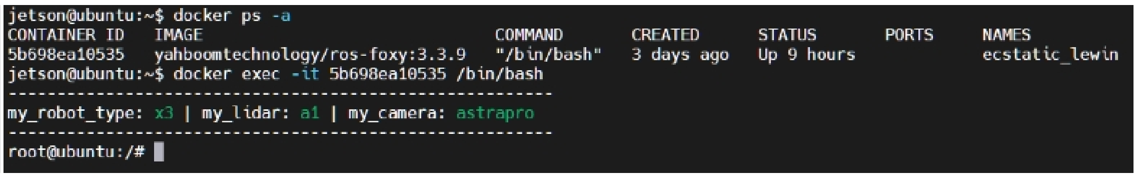
```
docker ps -a
```



```
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:/#
```

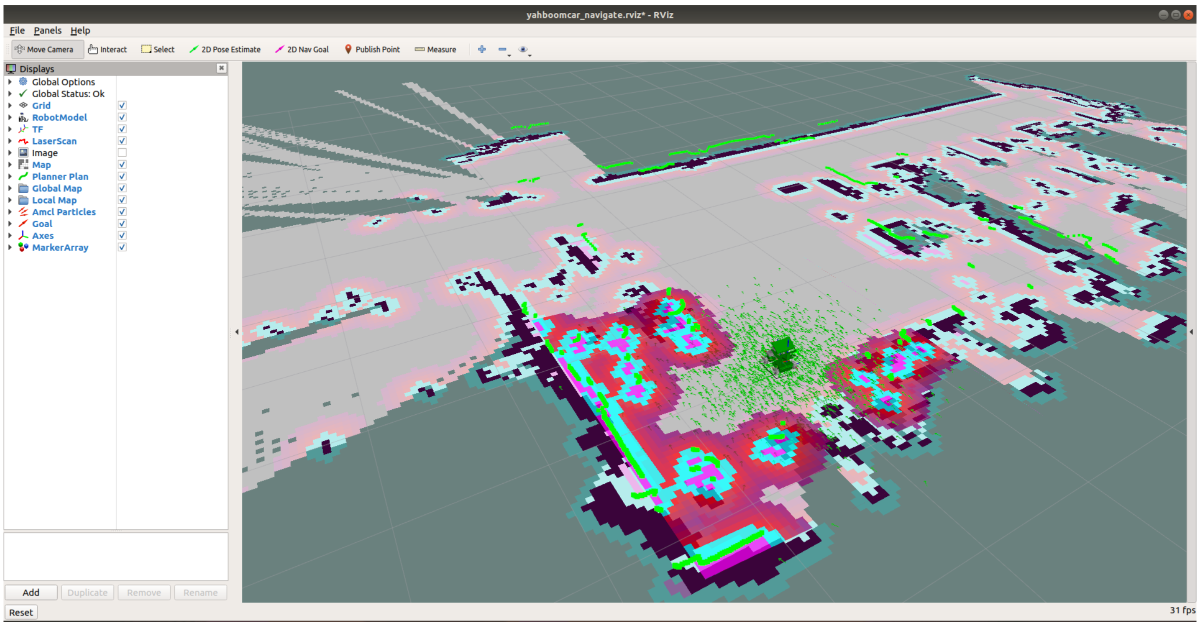
After successfully entering the container, you can open countless terminals to enter the container.

```
roslaunch yahboomcar_nav yahboomcar_navigation.launch use_rviz:=false map:=house
```

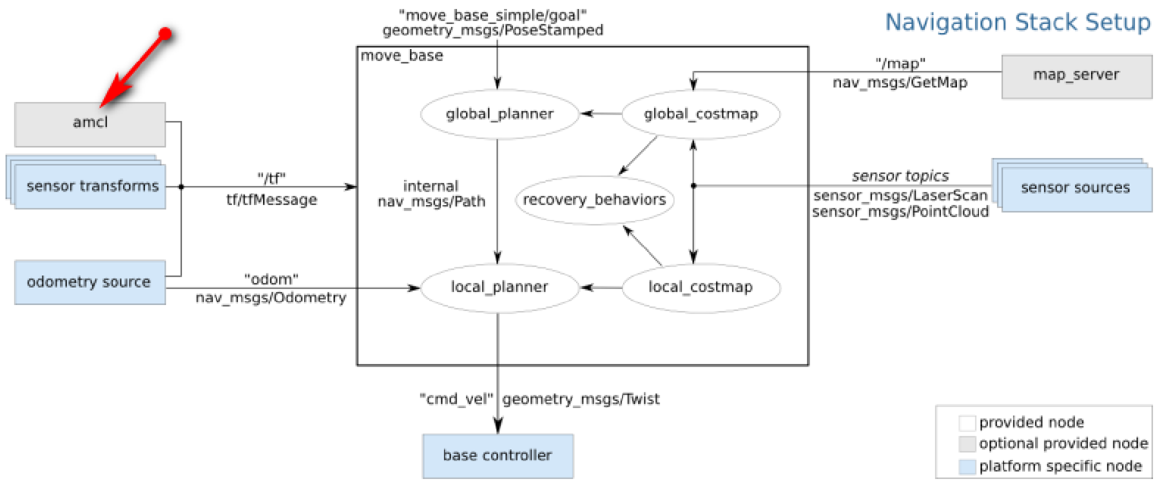
- [use_rviz] parameter: whether to open rviz.
- [map] parameters: map name, map to be loaded.

Turn on the visual interface (virtual machine side)

```
roslaunch yahboomcar_nav view_navigate.launch
```



11.2.2. Positioning analysis



The position of amcl in the upper left corner of the navigation frame is to tell the robot where it is.

After opening it, we can see a handful of green particles evenly scattered around the robot. When we move the robot randomly (keyboard or mouse control), all the particles also move with it. Use the position of each particle to simulate a sensor information and compare it with the observed sensor information (usually laser), thereby assigning a probability to each particle. Then the particles are regenerated according to the probability of generation. The higher the probability, the greater the probability of generation. After such iterations, all the particles will slowly converge together, and the exact position of the robot will be calculated.

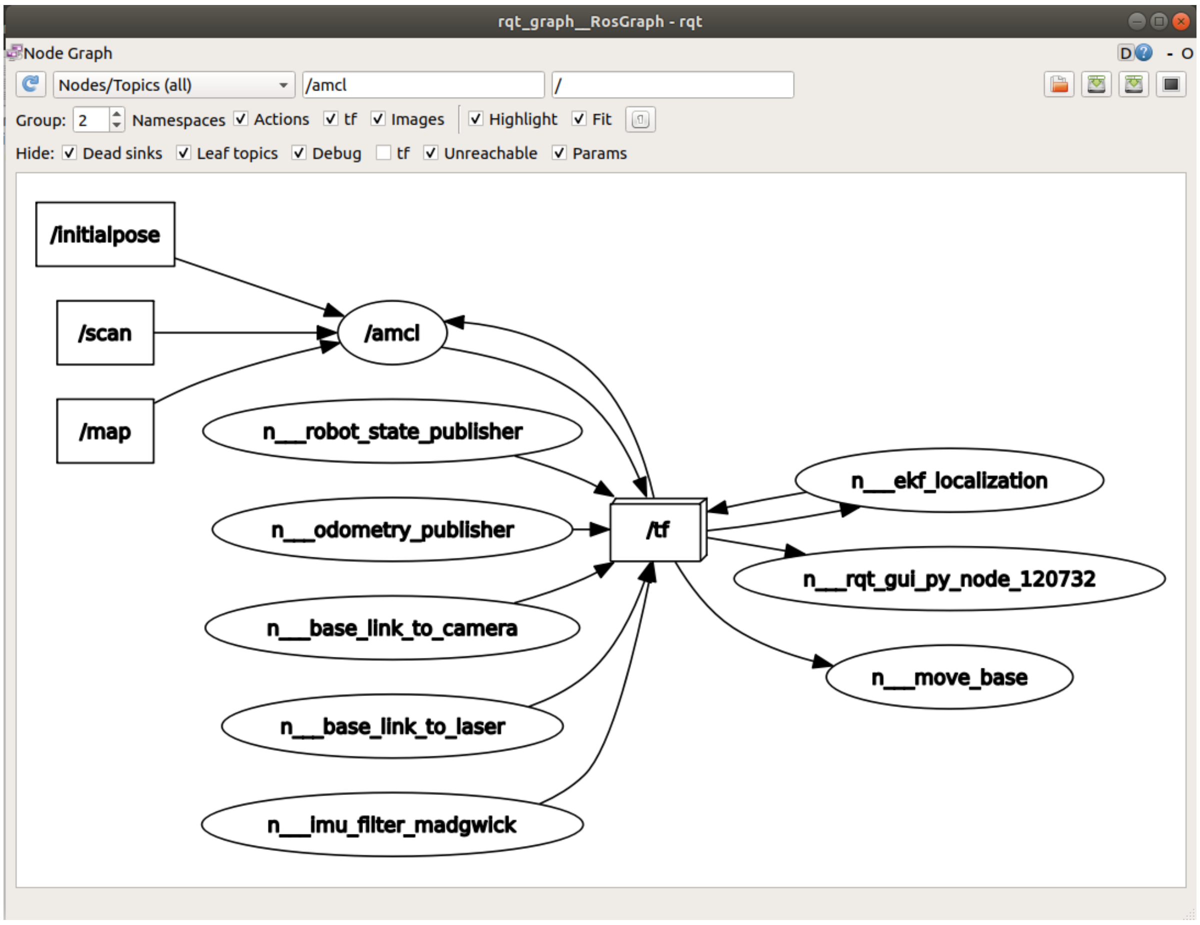
11.2. Topics and services

Subscription topic	**Type**	**Description**
scan	sensor_msgs/LaserScan	lidar data
tf	tf/tfMessage	Coordinate transformation information
initialpose	geometry_msgs/PoseWithCovarianceStamped	Used to (re)initialize the mean and covariance of the particle filter.
map	nav_msgs/OccupancyGrid	After setting the use_map_topic parameter, AMCL subscribes to this topic to retrieve maps for laser-based positioning. New features in Navigation 1.4.2.
Post Topic	**Type**	**Description**
amcl_pose	geometry_msgs/PoseWithCovarianceStatmped	Robot pose estimation in the map, with covariance information

Subscription topic	Type	Description
particlecloud	geometry_msgs/PoseArray	Collection of pose estimates maintained by particle filter
tf	tf/tfMessage	Publish conversion from odom to map
Server	**Type**	**Description**
global_localization	std_stvs/Empty	Initialize global positioning, all particles are randomly scattered in free areas on the map
request_normotion_update	std_stvs/Empty	Perform updates manually and publish updated particles
set_map	nav_msgs/SetMap	Service for manually setting new maps and poses.
Client	**Type**	**Description**
static_map	nav_msgs/GetMap	amcl calls this service to retrieve a map for laser positioning; enabling prevents retrieval of maps from this service.

View nodes

```
rosrun rqt_graph rqt_graph
```



11.3. Parameter configuration

There are three categories of ROS parameters that can be used to configure amcl nodes: ensemble filters, laser models, and odometry models.

- Overall filter parameters

Parameters	**Type**	**Default value**	**Description**
~min_particles	int	100	Minimum number of particles allowed
~max_particles	int	5000	Maximum number of particles allowed
~kld_err	double	0.1	Maximum error between the true distribution and the estimated distribution
~kld_z	double	0.99	The upper standard normal quantile of (1-p), where p is the probability that the estimated distribution error is less than kld_err
~update_min_d	double	0.2(m)	The translation distance required to perform a filter update
~update_min_a	double	pi/6.0(rad)	The rotational movement required to perform a filter update

Parameters	Type	Default value	Description
~reseample_interval	int	2	Number of filter updates before resampling
~transform_tolerance	double	0.1(s)	The time the transform is issued to indicate that this transform is valid for the future
~recovery_alpha_slow	double	0.0	The exponential decay rate of the slow average weight filter, used to decide when to perform recovery operations by adding random poses, 0.0 means disabled
~recovery_alpha_fast	double	0.0	The exponential decay rate of the fast average weight filter, used to decide when to perform recovery operations by adding random poses, 0.0 means disabled
~initial_pose_x	double	0.0(m)	Initial pose average (x), used to initialize the Gaussian distribution filter
~initial_pose_y	double	0.0(m)	Initial pose average (y), used to initialize the Gaussian distribution filter
~initial_pose_a	double	0.0(m)	Initial pose average (yaw), used to initialize the Gaussian distribution filter
~initial_cov_xx	double	0.5*0.5(m)	Initial attitude average (x*x), used to initialize the Gaussian distribution filter
~initial_cov_yy	double	0.5*0.5(m)	Initial attitude average (y*y), used to initialize the Gaussian distribution filter
~initial_cov_aa	double	(pi/12)*(pi/12)(rad)	Initial attitude average (yaw*yaw), used to initialize the Gaussian distribution filter
~gui_publish_rate	double	-1.0(Hz)	The maximum rate of publishing information during visualization, -1.0 means disabled
~save_pose_rate	double	0.5(Hz)	The maximum rate at which pose estimates initial_pose_ and covariance initial_cov_ are stored in the parameter server for subsequent initialization of the filter. -1.0 means disabled
~use_map_topic	bool	false	When set to true, amcl will subscribe to the map topic instead of receiving the map through a service call

Parameters	Type	Default value	Description
~first_map_only	bool	false	When set to true, amcl will only use the first map it subscribes to, rather than the map received with each update
~selective_resampling	bool	false	When set to true, will reduce the resampling rate when not needed and help avoid particle deprivation. Resampling will only occur when the effective number of particles (N_eff=1/(sum(k_i^2))) is less than half of the current number of particles.

- Laser model parameters

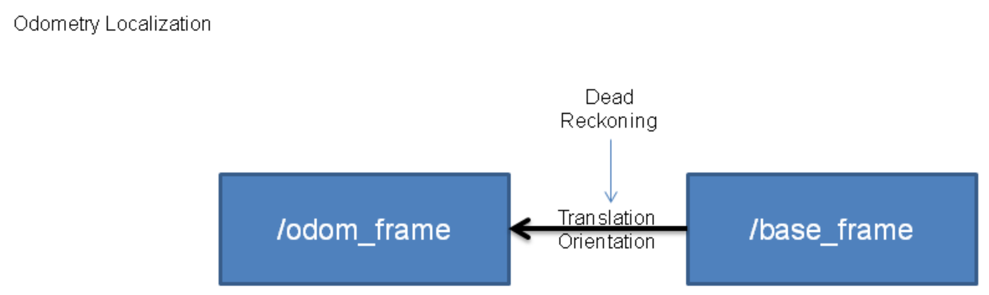
Parameters	Type	Default value	Description
~laser_min_range	double	-1.0	Minimum scan range, set to -1, the minimum usage range reported by lidar.
~laser_max_range	double	-1.0	Maximum scan range, set to -1, the maximum usage range reported by lidar.
~laser_max_beams	int	30	How many evenly spaced beams to use in each scan when updating the filter
~laser_z_bit	double	0.95	Mixing weight for the z_bit part of the model
~laser_z_short	double	0.1	Mixing weights for the z_short part of the model
~laser_z_max	double	0.05	Mixing weights for the z_max part of the model
~laser_z_rand	double	0.05	Mixing weights for the z_rand part of the model
~laser_sigma_hit	double	0.2(m)	The standard deviation of the Gaussian model used in the z_hit part of the model
~laser_lambda_short	double	0.1	Exponential decay parameter for the z_short part of the model
~laser_likelihood_max_dist	double	2.0(m)	Maximum distance on the map to measure obstacle expansion

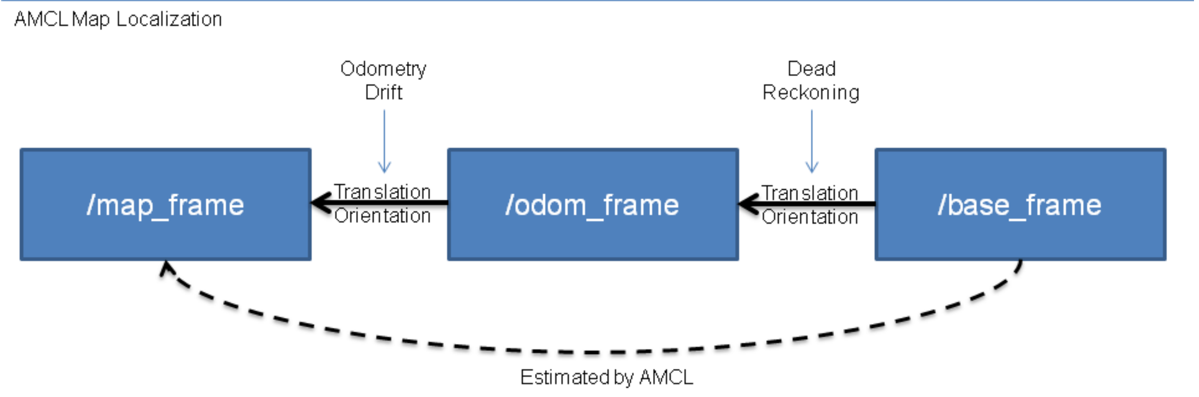
Parameters	Type	Default value	Description
~laser_model_type	string	"likelihood_field"	Model selection, bean, likelihood_field or likelihood_field_prob

- Odometer model parameters

Parameters	Type	Default value	Description
~odom_model_type	string	"diff"	Model selection, diff, omni, diff-corrected or omni-corrected
~odom_alpha1	double	0.2	Specifies the expected noise in odometry rotation estimates based on the rotational component of robot motion
~odom_alpha2	double	0.2	Specifies the expected noise in odometry rotation estimates based on the translational component of robot motion
~odom_alpha3	double	0.2	Specifies the expected noise in odometry translation estimates based on the translational component of robot motion
~odom_alpha4	double	0.2	Specifies the expected noise in odometry translation estimates based on the rotational component of robot motion
~odom_alpha5	double	0.2	Translation related noise parameter (only used in model omni)
~odom_frame_id	string	"odom"	Odometer coordinate system
~base_frame_id	string	"base_link"	The coordinate system of the robot chassis
~global_frame_id	string	"map"	Coordinate system published by the positioning system
~tf_broadcast	bool	true	When set to false, amcl will not publish the coordinate system transformation between map and odom

11.4. Coordinate transformation





Compare the two methods Odometry Localization and AMCL Map Localization:

Odometry Localization: The rotation speed of the wheel can be known through the encoder feedback of the motor, the current wheel speed can be known according to the radius of the wheel, the overall moving speed of the robot can be obtained according to the forward kinematics model, and the movement can be known by integrating according to time. total mileage.

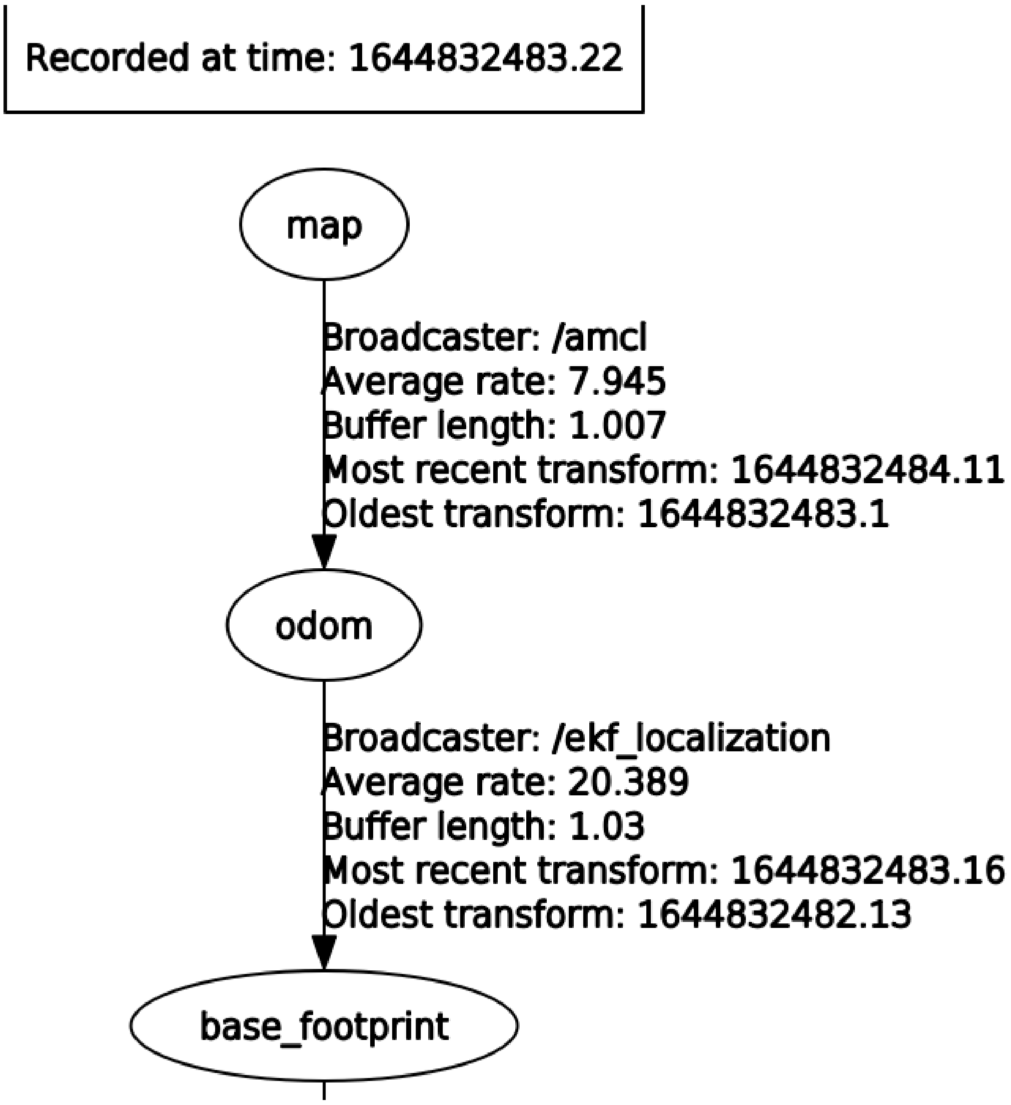
AMCL Map Localization: In amcl, the pose information of base_frame corresponding to map_frame has been obtained through particle filter positioning. However, in order to ensure that base_frame has only one parent node, this conversion is not directly released, because the parent node of base_frame is already odom_frame. Then amcl can only publish the conversion from map_frame to odom_frame. Another advantage of this is that we can get the cumulative error of odom.

We have always said that the distance measurement information obtained by the wheel odometer will have a cumulative error, which is similar to a car skidding. If only the wheel odometer is used to calculate the moving distance, it willError:

For example: the robot walks forward in a straight line, and the result obtained from the wheel odometer should be that it has walked 1 meter, so the distance to odom_frame should be 1 meter. However, due to slipping on the spot, the robot actually found that it did not move at all through amcl positioning, then amcl needs to publish the conversion from map_frame to odom_frame to remove this 1-meter error, so that the entire tf tree can be maintained normally.

The conversion relationship between coordinates.

```
rosrun rqt_tf_tree rqt_tf_tree
```


Recorded at time: 1644832483.22
map
Broadcaster: /amcl
Average rate: 7.945
Buffer length: 1.007
Most recent transform: 1644832484.11
Oldest transform: 1644832483.1
odom
Broadcaster: /ekf_localization
Average rate: 20.389
Buffer length: 1.03
Most recent transform: 1644832483.16
Oldest transform: 1644832482.13
base_footprint

As can be seen from the above figure, the coordinate conversion from map to odom is completed by the amcl node.