1. preface

From the previous quick start manual, we can understand that the mobile robot end and PC end here can actually be understood as the robot end in NoMachine in this device. The virtual machine runs on the PC end, so all subsequent terminal commands can be directly entered in NoMachine virtual machine.

2. Camera and Image

Camera Driver and Image Viewing

Using the startup camera on the mobile robot end:

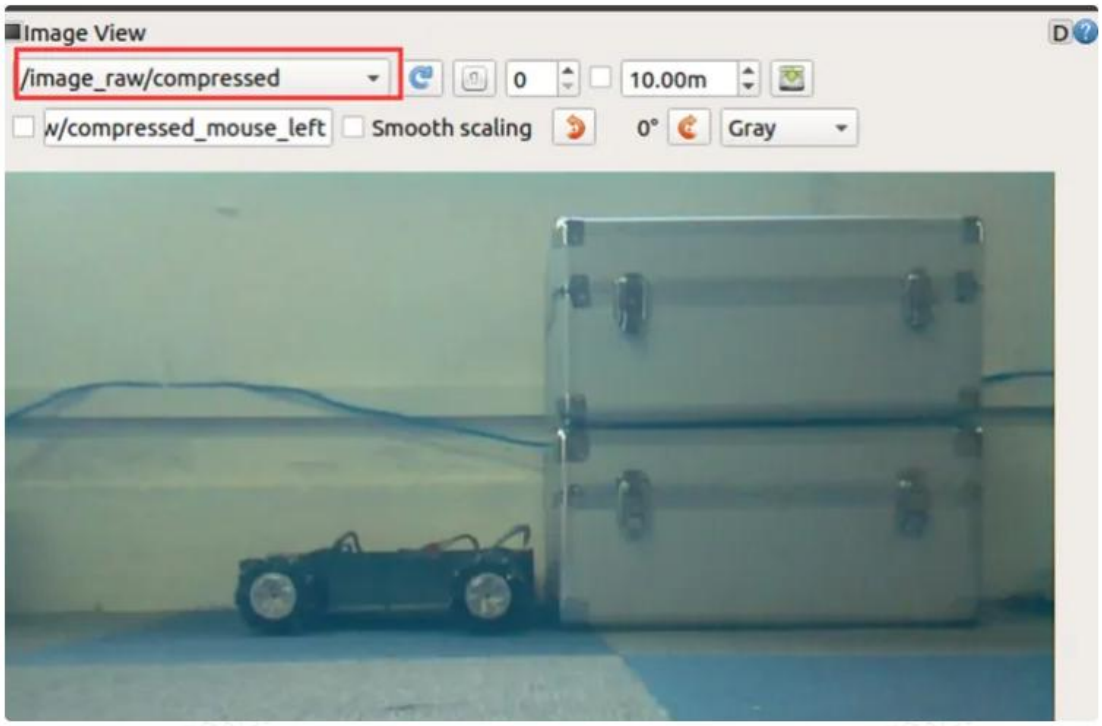
```
roslaunch robot_vision robot_camera.launch
```

After normal startup, the red indicator light on the camera will light up

Open the virtual machine and run rqt tool on the PC to view images

```
rqt_image_view
```

After starting the tool, the following window will pop up. Select the topic "/image_raw/compressed" in the red box to view the current camera image.



It is recommended to choose the **/image_raw/compressed** topic here because the images output from this topic will be compressed to avoid excessive bandwidth and lag. By using the **rostopic bw** command, it can be observed that there is a multiple difference in bandwidth between these two topics.

```
bingda@ubuntu:~$ rostopic bw /image_raw
subscribed to [/image_raw]
average: 8.27MB/s
        mean: 0.92MB min: 0.92MB max: 0.92MB window: 6
average: 7.72MB/s
        mean: 0.92MB min: 0.92MB max: 0.92MB window: 14
average: 4.83MB/s
        mean: 0.92MB min: 0.92MB max: 0.92MB window: 14
^Caverage: 1.94MB/s
        mean: 0.92MB min: 0.92MB max: 0.92MB window: 14
bingda@ubuntu:~$ rostopic bw /image_raw/compressed
subscribed to [/image_raw/compressed]
average: 1.10MB/s
        mean: 0.04MB min: 0.04MB max: 0.04MB window: 28
average: 1.06MB/s
        mean: 0.04MB min: 0.04MB max: 0.04MB window: 57
average: 1.08MB/s
        mean: 0.04MB min: 0.04MB max: 0.04MB window: 88
^Caverage: 1.02MB/s
        mean: 0.04MB min: 0.04MB max: 0.04MB window: 93
bingda@ubuntu:~$
```

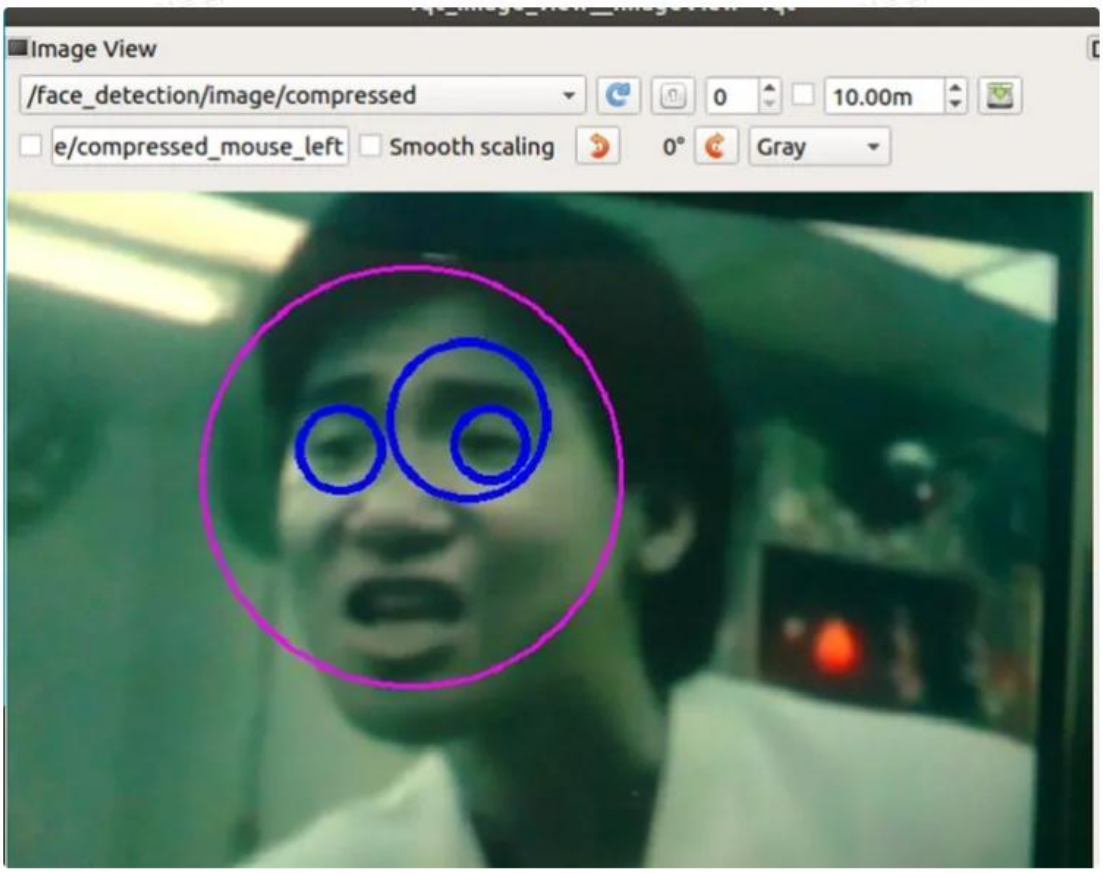
OpenCV - Human/Face Detection

Facial detection

Run the camera function package and face detection function package on the robot end.

```
roslaunch robot_vision robot_camera.launch
roslaunch robot_vision face_detection.launch
```

Open the rqt_image_view tool on the PC and subscribe to **/face_detection/image/compressed** to achieve face detection.



Facial recognition

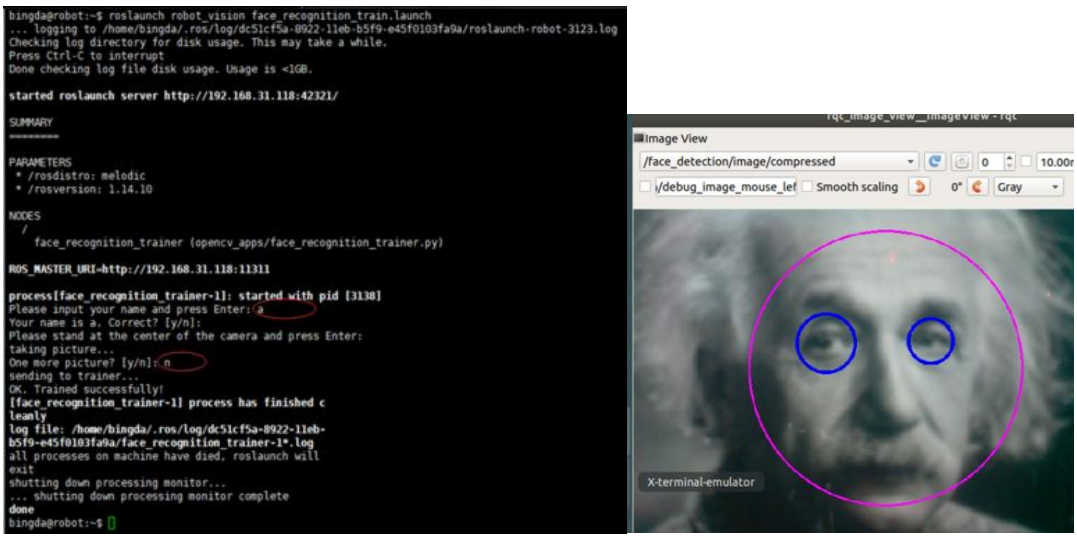
Run the following nodes in each of the three terminals on the robot end.

Start the camera roslaunch robot_vision robot_camera.launch

Start facial recognition roslaunch robot_vision face_recognition.launch

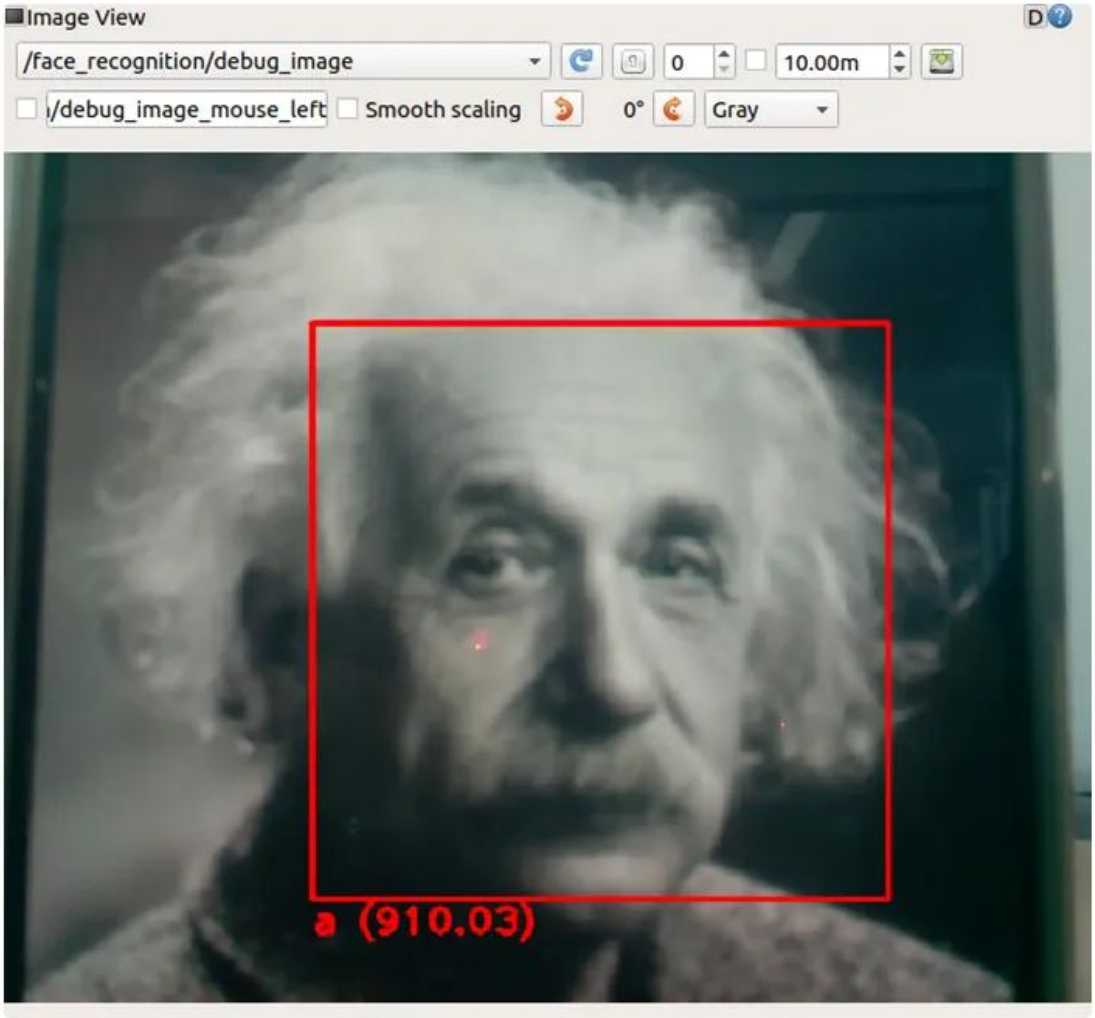
Start facial data entry in oslaunch robot_vision face_recognition_train.launch

After starting facial data entry, first name the face to be entered, and here name it '**a**'



Then take a photo of the face, open the rqt_image_view tool on the PC, subscribe to **/face_detection/image/compressed** to detect the face. After detecting the face, press

enter on the face input terminal to take a photo of the face. Choose whether to take another photo and simply enter **'n'** to complete the facial input without taking multiple photos. Returning to the rqt_image_view tool, subscribe to the /face_recognition/debug_image topic to perform facial recognition



Human body testing

Start the camera function package on the robot end first, and then start the human detection node

```
roslaunch robot_vision robot_camera.launch
roslaunch robot_vision people_detect.launch
```

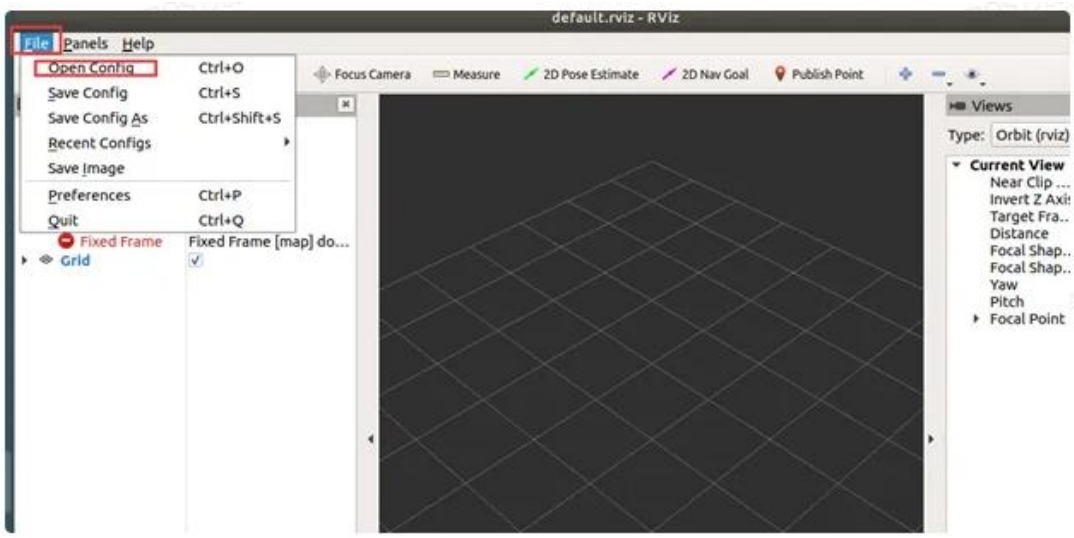
Open the rqt_image_view tool and subscribe to /people_detect/image/compressed to achieve human detection

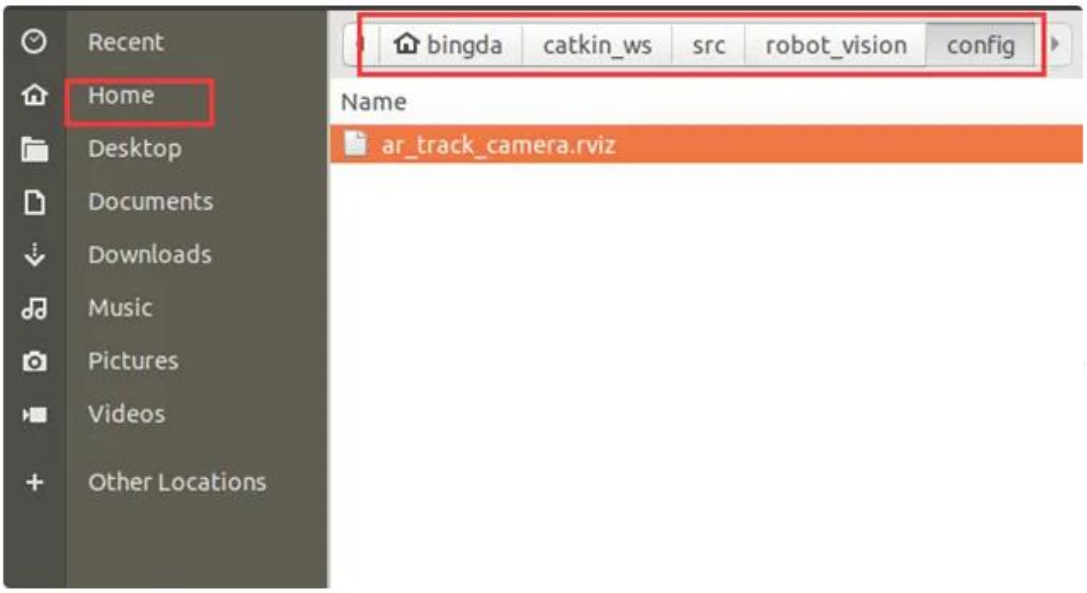
QR code AR tag pose detection

Robot end starts camera `roslaunch robot_vision robot_camera.launch`

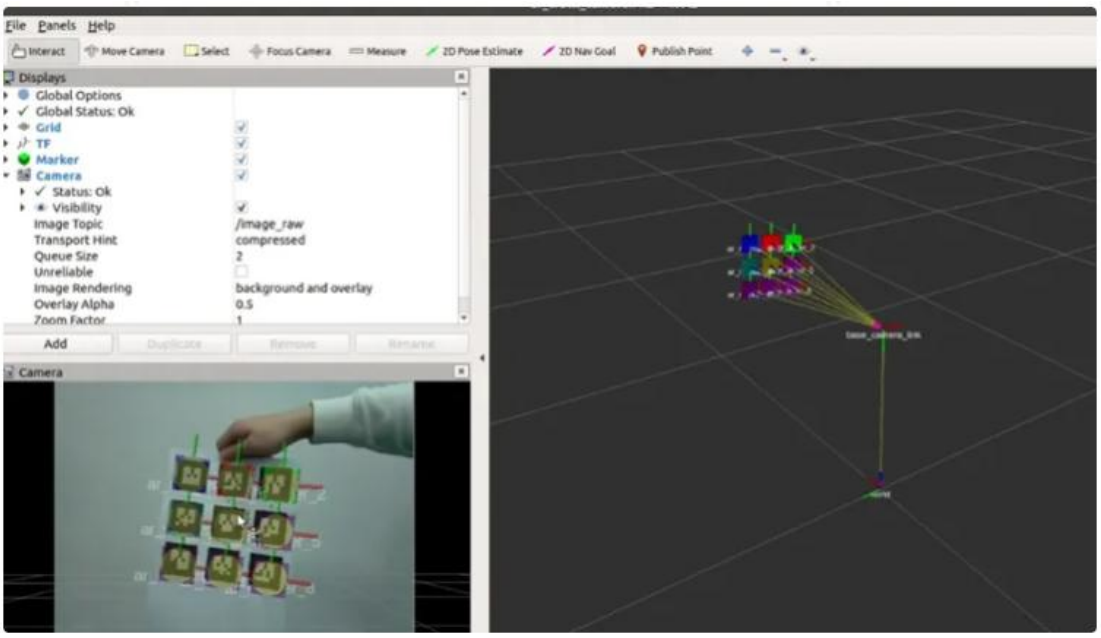
Start AR tag tracking on the robot end, `roslaunch robot_vision ar_track.launch`

Start rviz on the PC, load the configuration file, and click File ->OpenConfig. Find home ->catkin_ws ->src ->robot_vision ->config and select this configuration file.

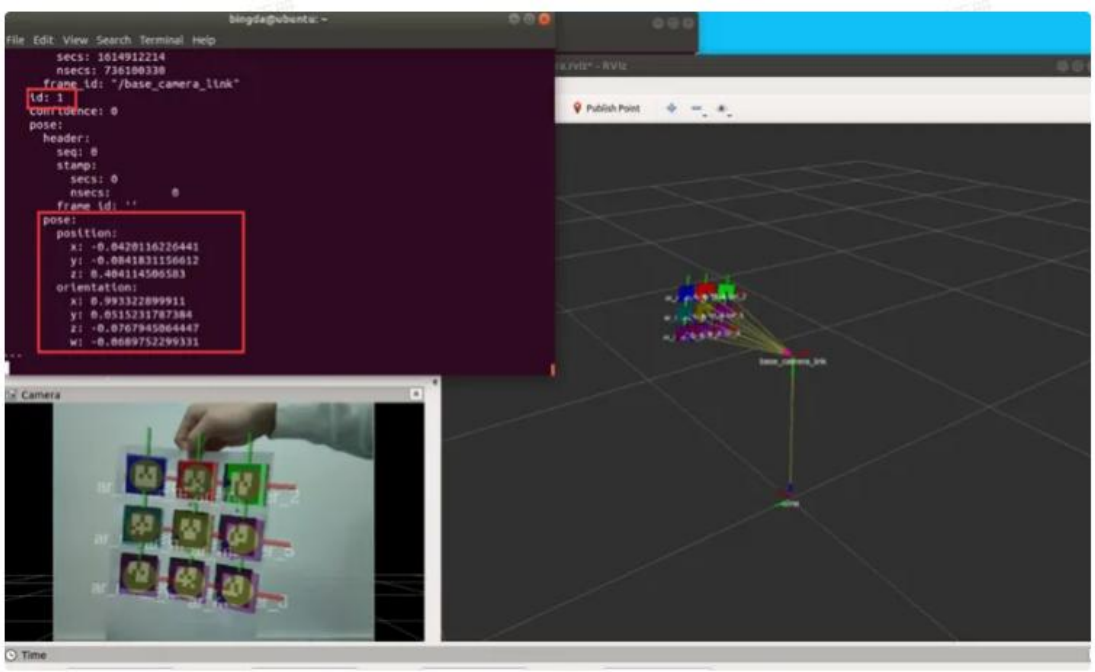




After loading the configuration file, take out the prepared QR code label. As can be seen, the position of the QR code is indicated on the left, and the distance of the QR code is indicated on the right



Open a new terminal on the robot or PC end to run rostopic echo /ar_pose_marker
You can see that here there is an output of the corresponding serial number for each QR code, as well as distance information relative to the camera origin and quaternion representation of heading and attitude



The concept of distributed operation: Distributed operation means that the nodes of ROS run on different robots in the same LAN through distributed communication. Let's first use CTRL+C to turn off the AR tag tracking node on the robot side;
Then, start AR tag tracking on the PC, and launch the roslaunch robot_vision ar_track.launch

At this point, when you return to the rviz interface on the PC side, you will find that the result is the same as running AR tag tracking on the robot side just now. But compared to the previous scene, there is a noticeable lag phenomenon this time. This is because AR tag tracking requires subscribing to the topic of image_raw, so the PC runtime needs to first transfer the content of image_raw to the PC. The bandwidth occupied by image_raw is very high, and it may cause lag during transmission due to data loss and other issues.

It should be noted that not all nodes can run on different devices within the same LAN. For example, devices like rviz that are related to displaying images must run on devices with displays, such as PCs, while robots that log in remotely through SSH do not have display conditions. And nodes related to hardware devices such as chassis and cameras must run on the robot end, because there is no hardware like chassis and cameras on the PC end.

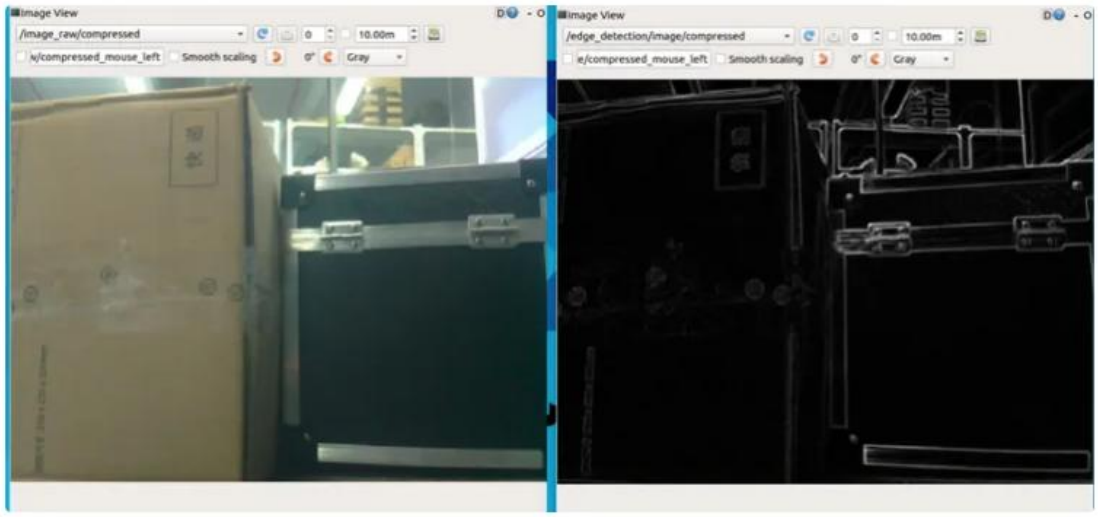
OpenCV - Edge Detection Class

edge detection

Robot end starts camera `roslaunch robot_vision robot_camera.launch`

Start edge detection on the robot end. `roslaunch robot_vision edge_detection.launch`

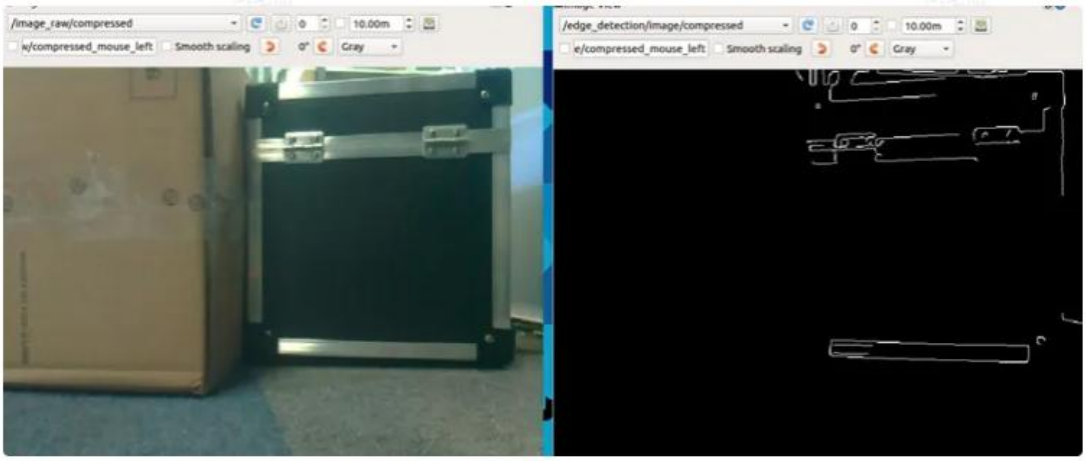
On the PC side, open two rqt_image_view tools on two terminals and subscribe to **/image_raw/compressed** and **/edge_detection/image/compressed** respectively



The normal image on the left shows two boxes, while the edge lines of the two boxes are displayed on the right.

Edge detection can extract image edges with significant changes in brightness or color. Edge detection can transmit parameters, with a default value of 0. The parameters passed in can be modified .

`roslaunch robot_vision edge_detection.launch edge_type:=2`



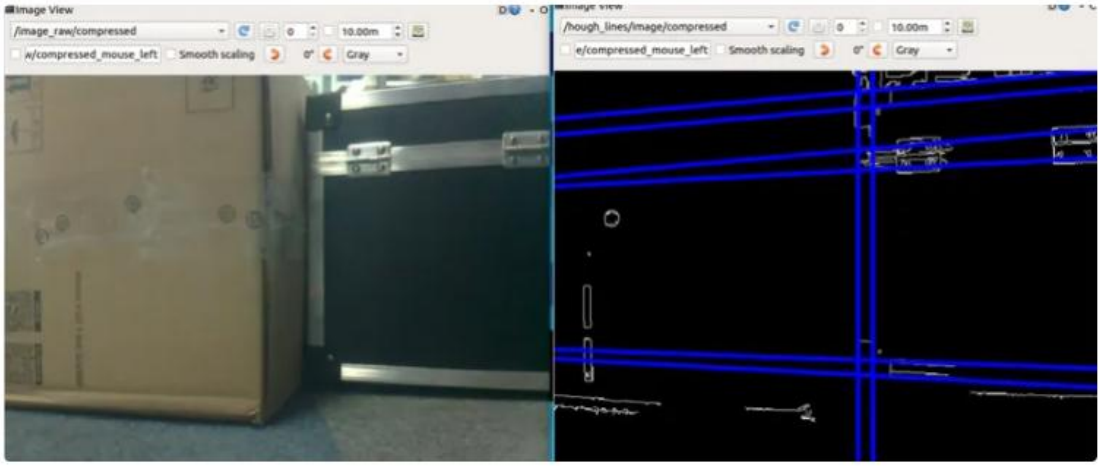
It can be seen that the cardboard box is completely blurry, while the metal frame and black box of another box, which have a strong contrast in color, have been abstracted into some lines. After abstracting complex images into lines through edge detection, features can be extracted from them.

Hough line detection

Robot end starts camera `roslaunch robot_vision robot_camera.launch`

Robot end starts line detection `roslaunch robot_vision hough_lines.launch`

Two rqt_image_view tools on the PC side subscribe to /image_raw/compressed and /hough_lines/image/compressed

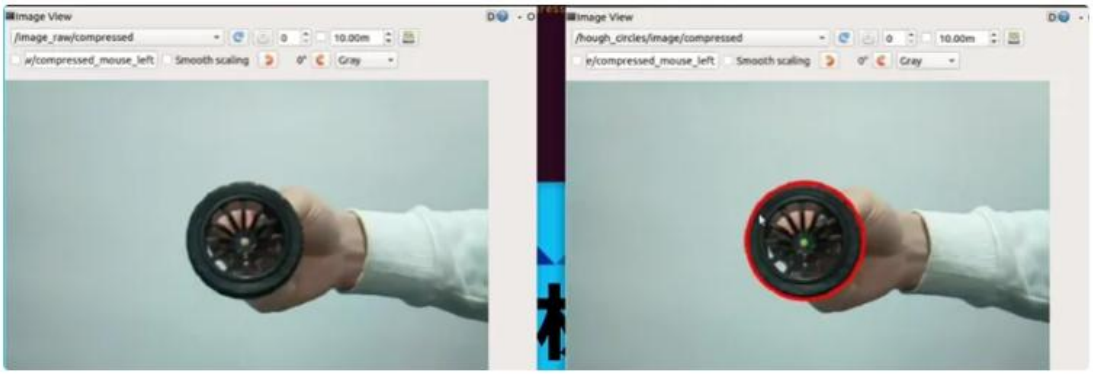


Hoff circle detection

Robot end starts camera `roslaunch robot_vision robot_camera.launch`

Robot end starts circular detection. `roslaunch robot_vision hough_circles.launch`

Two rqt_image_view tools on the PC side subscribe to /image_raw/compressed and /hough_circles/image/compressed.

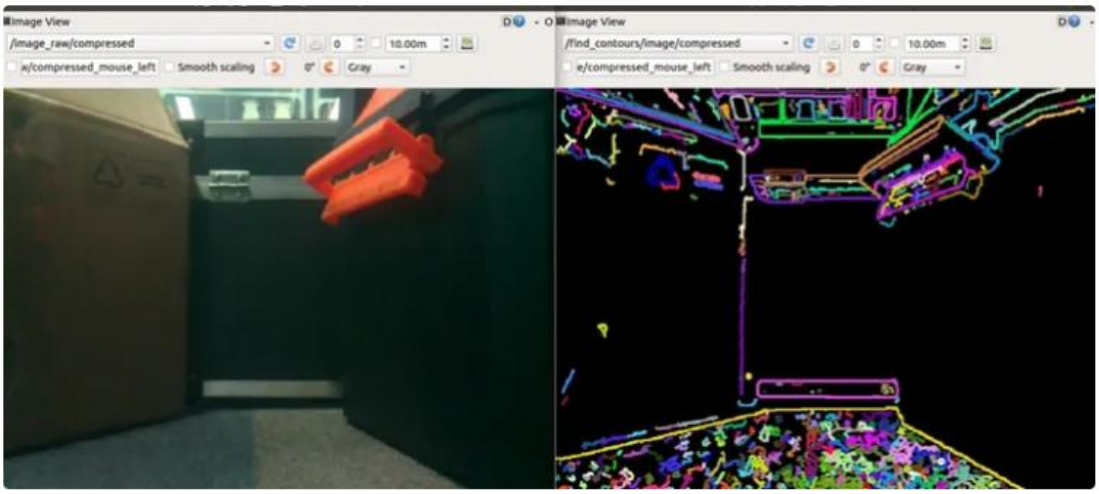


OpenCV - Structural Analysis Class

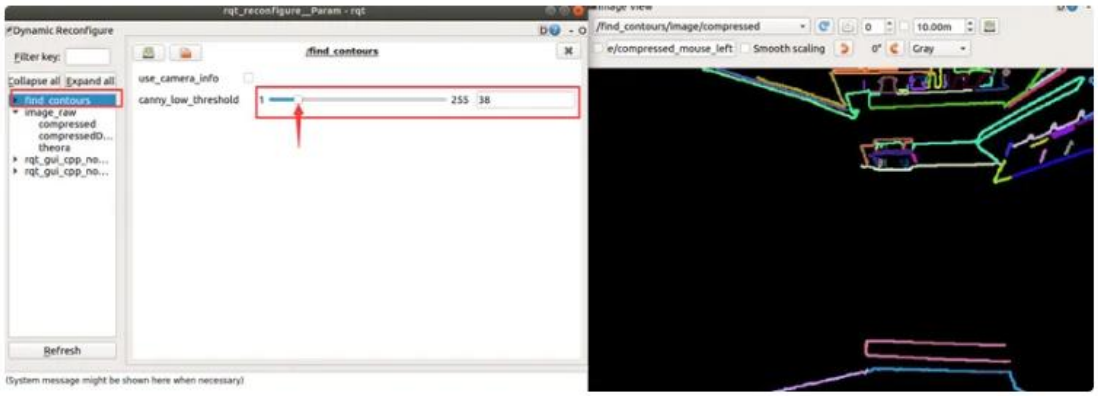
This section demonstrates the application of structural analysis class demos in OpenCV

Find_contours contour detection

Robot end starts camera `roslaunch robot_vision robot_camera.launch`
Robot end start contour detection `roslaunch robot_vision find_contours.launch`
Two `rqt_image_view` tools on the PC side subscribe to /image_raw/compressed and /find_contours/image/compressed.



It can be seen that contour detection depicts the entire outline of the palm, which is different from the edge detection introduced earlier. Edge detection depicts the lines of the edges, while contour detection displays the entire contour.
Here, we will reopen a dynamic parameter tuning tool called rosrun rqt_reconfigure rqt_reconfigure
Select find_contours on the right side to dynamically adjust parameters to change the image effect

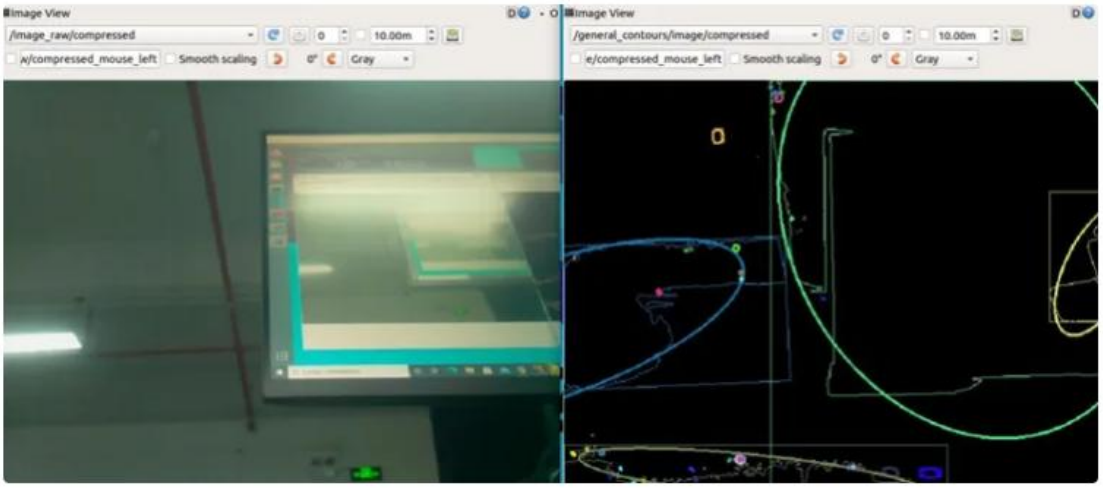


General_contours contour detection

Robot end starts camera `roslaunch robot_vision robot_camera.launch`

Start a universal contour detection `rroslaunch robot_vision general_contours.launch` on the robot end

rqt_image_view tool subscription /general_contours/image/compressed



This node no longer depicts a specific outline, but surrounds it with a rectangle. When we need to analyze images, we need to extract them. In the previous routine, the specific contour obtained was an irregular shape that cannot be extracted into an image. And the box depicted by this node can be intercepted to obtain a new image.

Let's output a rectangle to describe the relevant topic

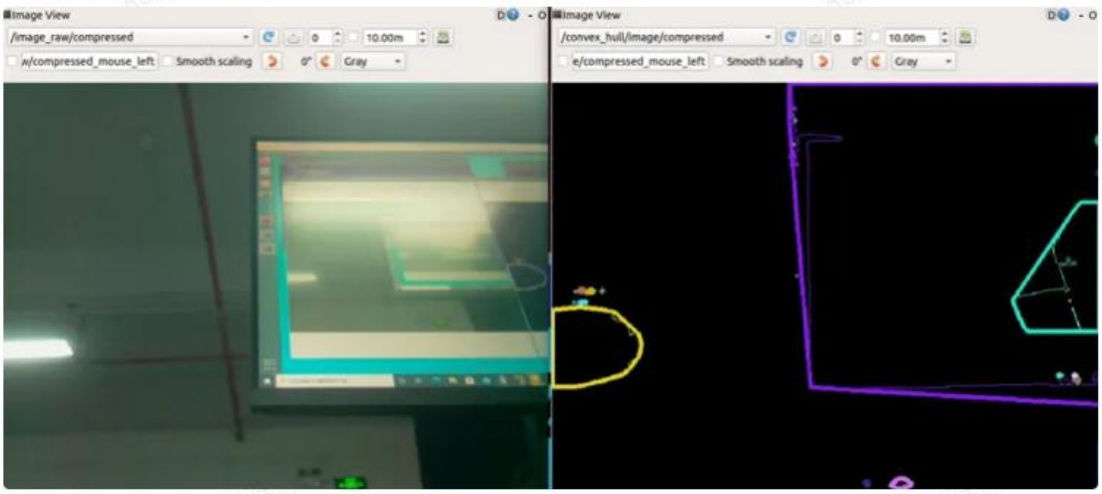
`rostopic echo /general_contours/rectangles`

```
-
  angle: -45.0
  center:
    x: 242.750015259
    y: 47.7500076294
  size:
    width: 4.2426404953
    height: 3.53553390503
-
```

Here, the center of the rectangle and the angle of its length and width placement will be output, so that the rectangle can be extracted from the image based on this information.

Convex hull detection convex_hull

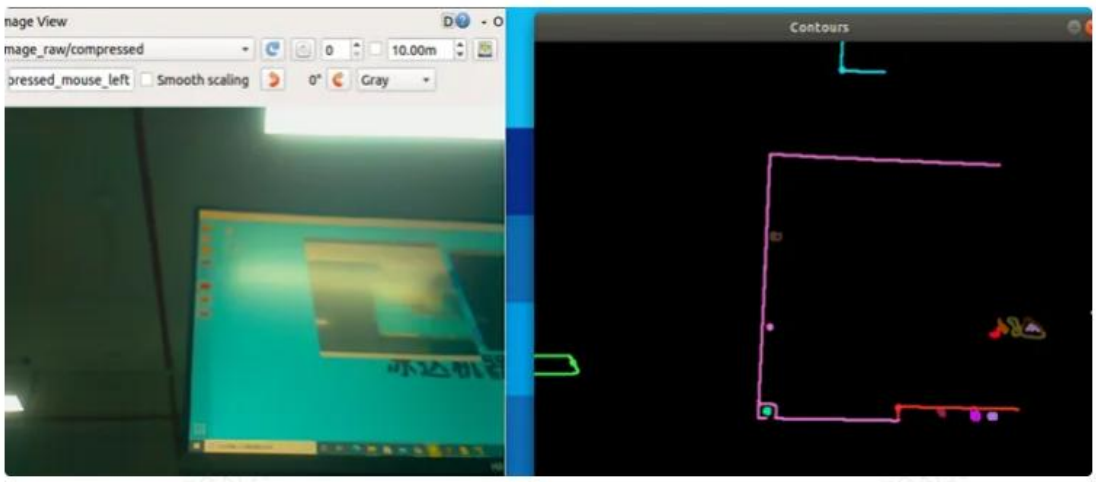
Robot end starts camera `roslaunch robot_vision robot_camera.launch`
Robot end starts convex hull inspection. `roslaunch robot_vision convex_hull.launch`
rqt_image_view tool subscription **/convex_hull/image/compressed**



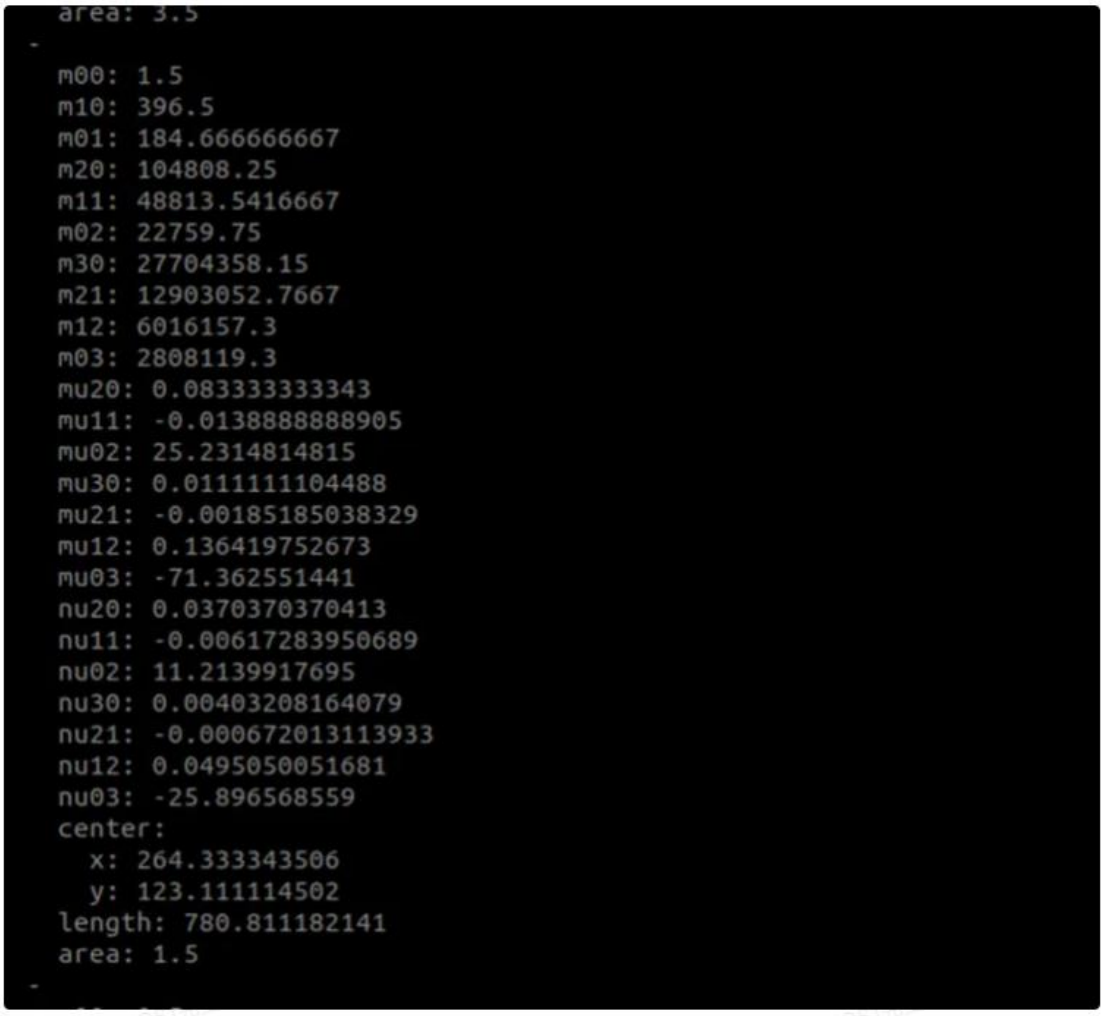
A convex hull is a method that connects the outermost points to form a convex polygon, which falls between contour detection and general contour detection. This polygon will not have any concavity, so it can enclose the area where the contour is located with the smallest area.

Image Moments Calculation Contour-Moments

Robot end starts camera `roslaunch robot_vision robot_camera.launch`
Starting Image Calculation on PC: `roslaunch robot_vision contour_moments.launch debug_view:=true`
The application of image computing will launch an image-based debugging window (on the right side)



Run the robot or PC version of the `rostopic echo /contour_moments/moments`



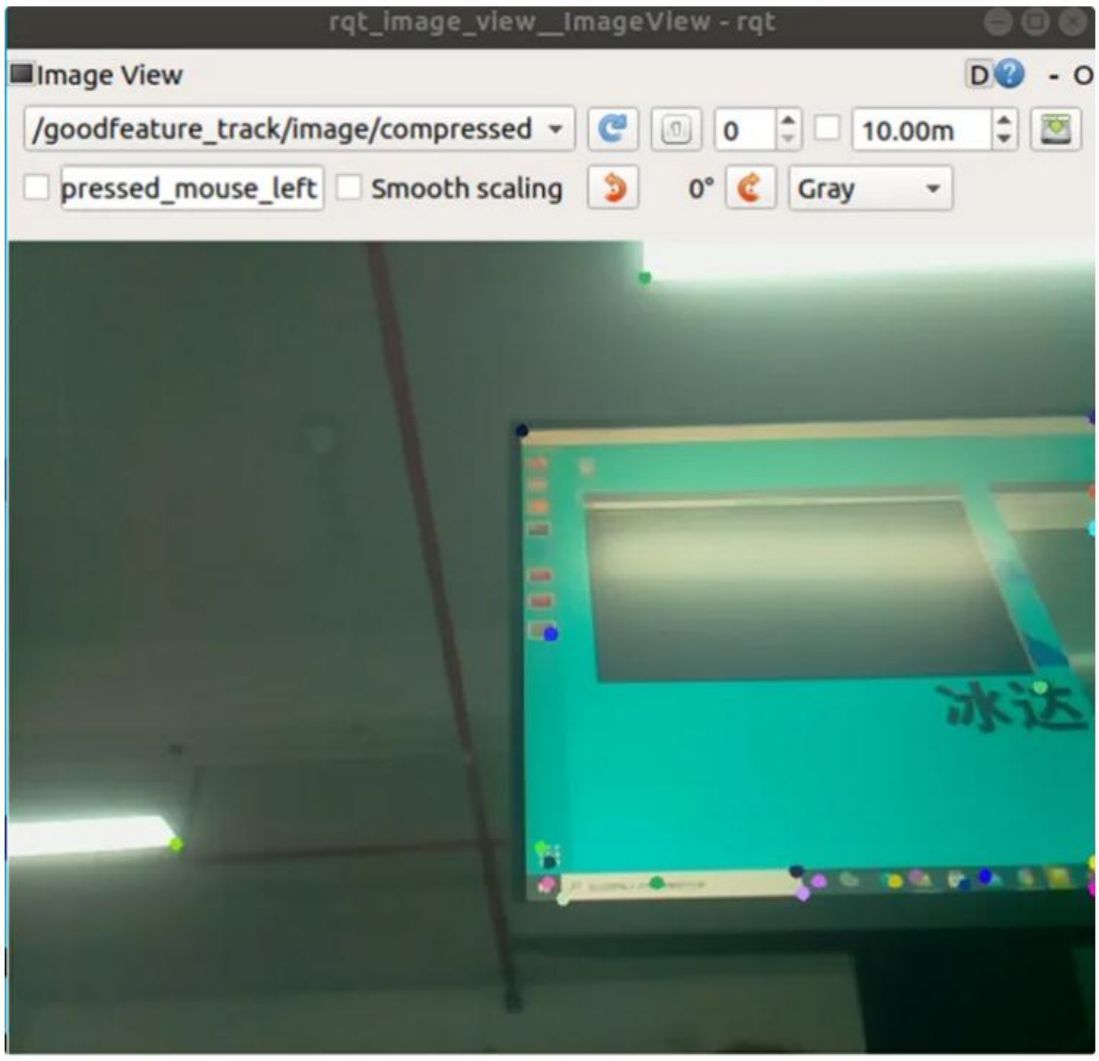
This topic will output some calculations for images, including center of gravity, center, perimeter, area, etc.

OpenCV - Motion Analysis Class

This section introduces the motion analysis class demo in OpenCV

Goodfeature_track corner detection

Robot end starts camera `roslaunch robot_vision robot_camera.launch`
Robot start corner detection `roslaunch robot_vision goodfeature_track.launch`
Start the `rqt_image_view` tool on the PC /goodfeature_track/image/compressed

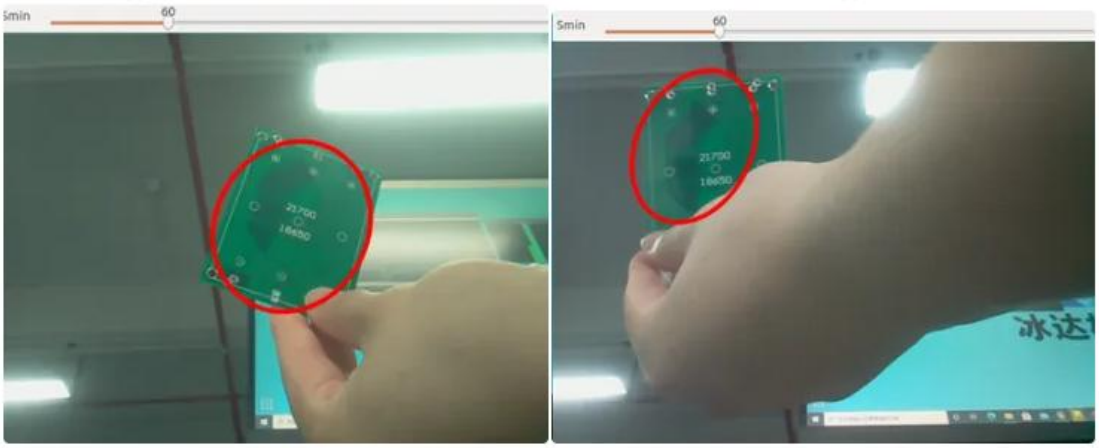


It can be seen that there are some obvious feature points and corners in the image.

Camshift object detection

Robot end starts camera `roslaunch robot_vision robot_camera.launch`

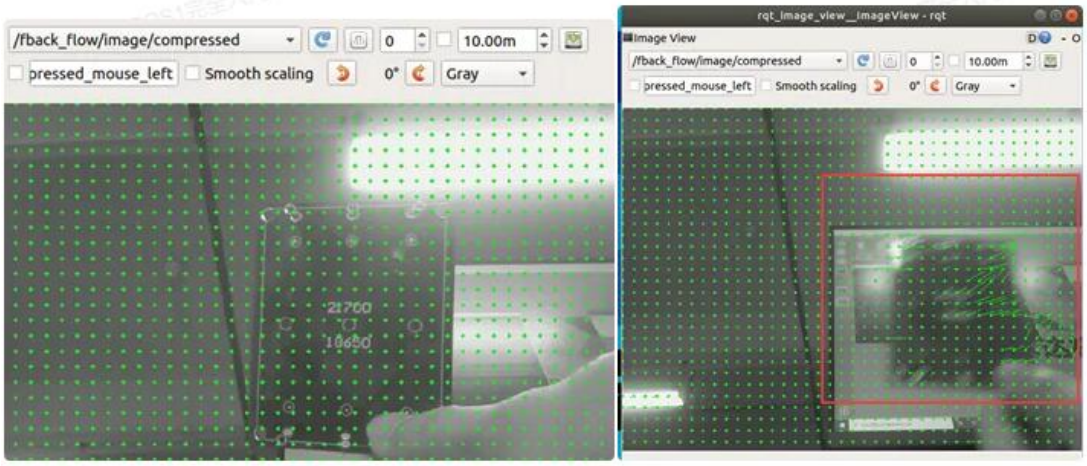
Starting object tracking on PC: `roslaunch robot_vision camshift.launch debug_view:=true`



Use a PCB board as a demonstration. After selecting the board with the left mouse button, move the board to see that it is being tracked. This is the continuous tracking achieved by utilizing the detected feature points.

Fback_flow dense optical flow

Robot end starts camera `roslaunch robot_vision robot_camera.launch`
Start the `roslaunch robot_vision fback_flow.launch` on the robot end
Start the `rqt_image_view` tool on the PC /fback_flow/image/compressed



The so-called optical flow refers to the flow direction of the position changes of objects in the two frames of the image. It is used to track the trend of object movement, and at this point, a pair of small green dots can be seen on the screen. When the screen is stationary, the green dots are evenly distributed. When the handheld PCB board moves, the direction of object movement can be identified.
There are several other processes related to motion analysis that you can try and run on your own.

Lk_flow sparse optical flow

Robot end starts camera `roslaunch robot_vision robot_camera.launch`
PC side `roslaunch robot_vision lk_flow.launch debug_view:=true`

Phase_corr phase method optical flow

Robot end starts camera `roslaunch robot_vision robot_camera.launch`
Start `roslaunch robot_vision phase_corr.launch debug_view:=true` on PC

Simple optical flow detection:

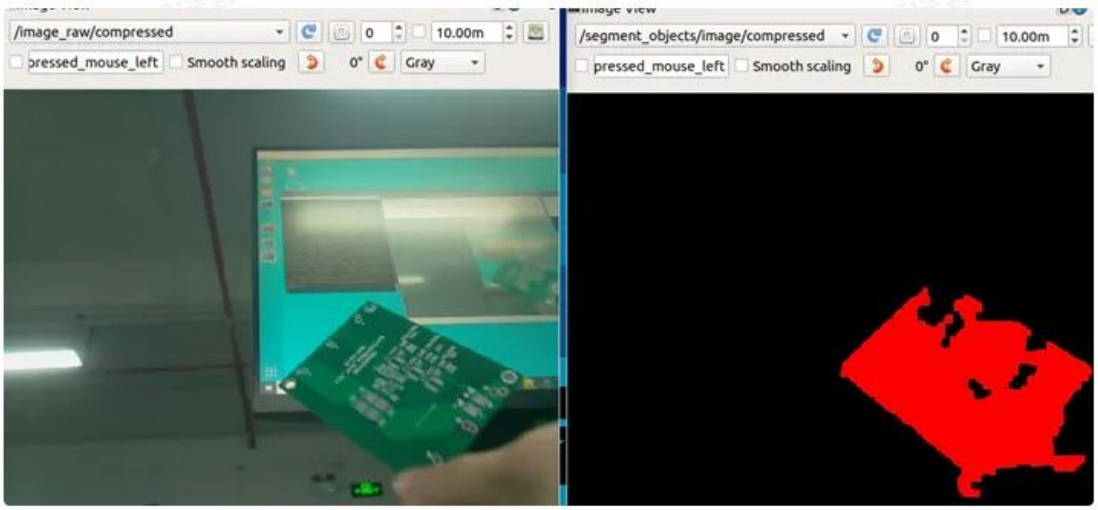
Robot end starts camera `roslaunch robot_vision robot_camera.launch`
Start `roslaunch robot_vision simple_flow.launch debug_view:=true` on PC

OpenCV Object Segmentation Class

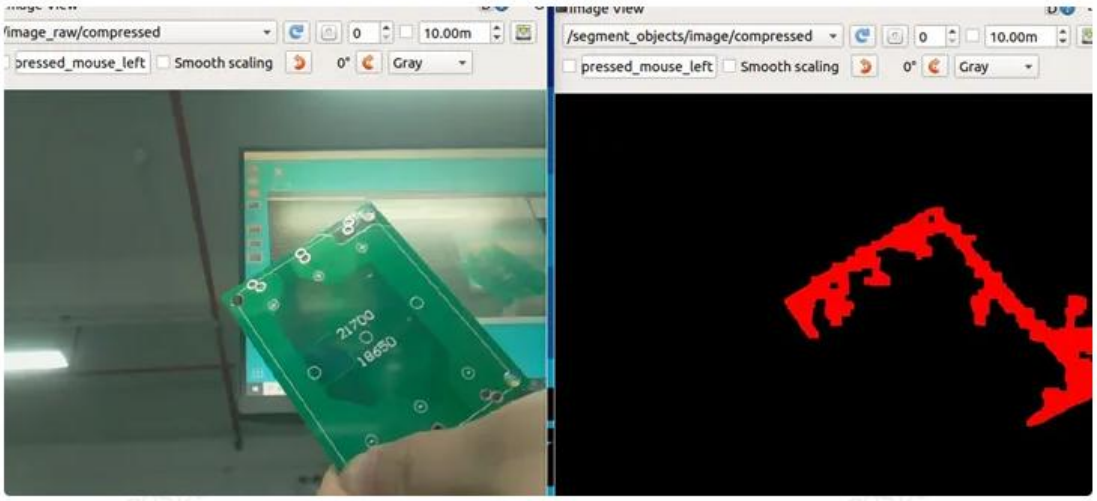
This section introduces the operation of object segmentation class demo in OpenCV

Object segmentation segment_objects

Robot end starts camera `roslaunch robot_vision robot_camera.launch`
Robot Start Object Segmentation `roslaunch robot_vision segment_objects.launch`
On the PC side, open two `rqt_image_view` tools on two terminals and subscribe to /image_raw/compressed and /segment_objects/image/compressed.
The image on the right was originally completely black when the PCB board suddenly appeared under the camera. It will be displayed on the black background on the right side.



When we are interested in a certain area of an image and it undergoes significant changes, we can extract the area of interest through object segmentation methods.



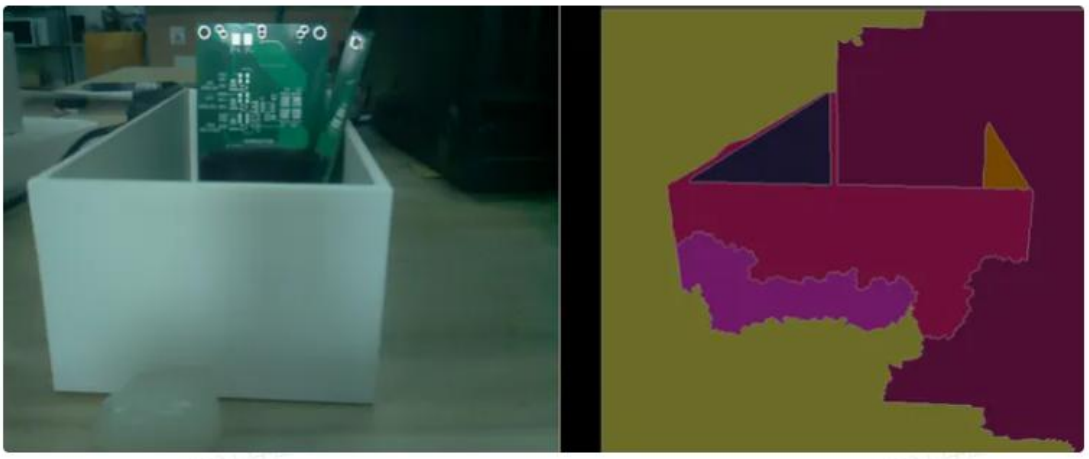
This method has a drawback that when the change stops, the display will gradually turn black until the background is completely black, so it cannot choose the object it wants to extract.

Watershed_segmentation

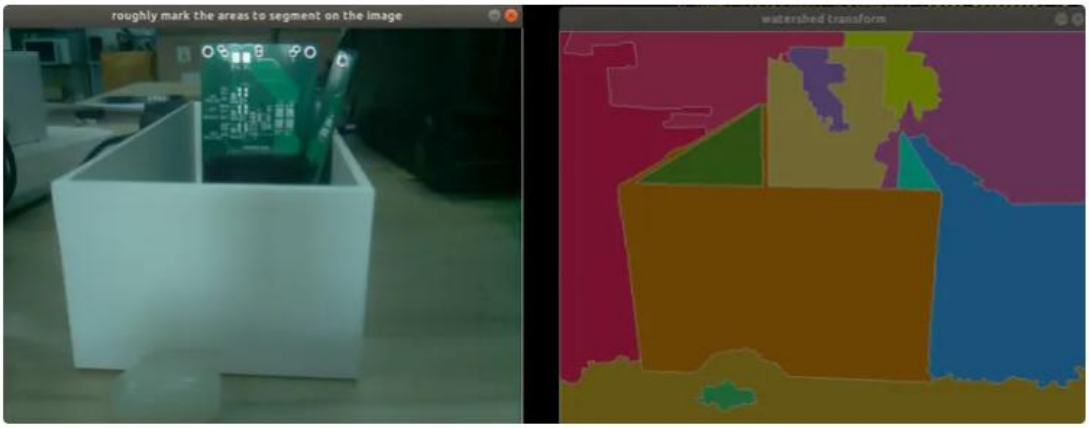
`roslaunch robot_vision robot_camera.launch` on the robot end
Start `roslaunch robot_vision watershed_segmentation.launch debug_view:=true`

on the PC.
At this point, the normal image captured by the camera appears:

You can see that the image obtained by the normal camera is now on the left. At this point, the right mouse button will draw a line on the white small box area in the left image, and the selected small area will appear in the right image display.



As we continue to improve, drawing through each part of the image will result in a complete segmentation map, with clear boundaries for each area.

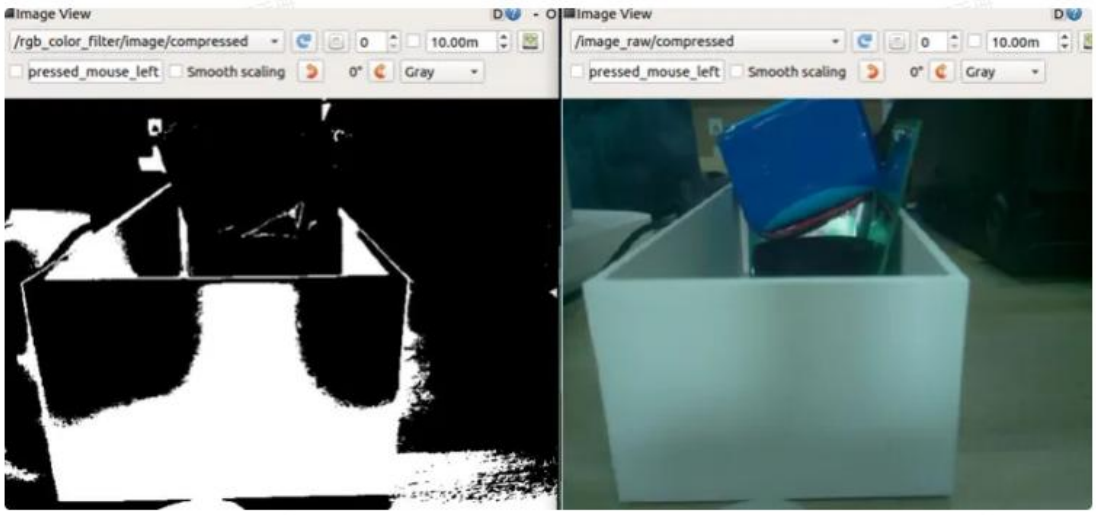


OpenCV - Color Filter Class

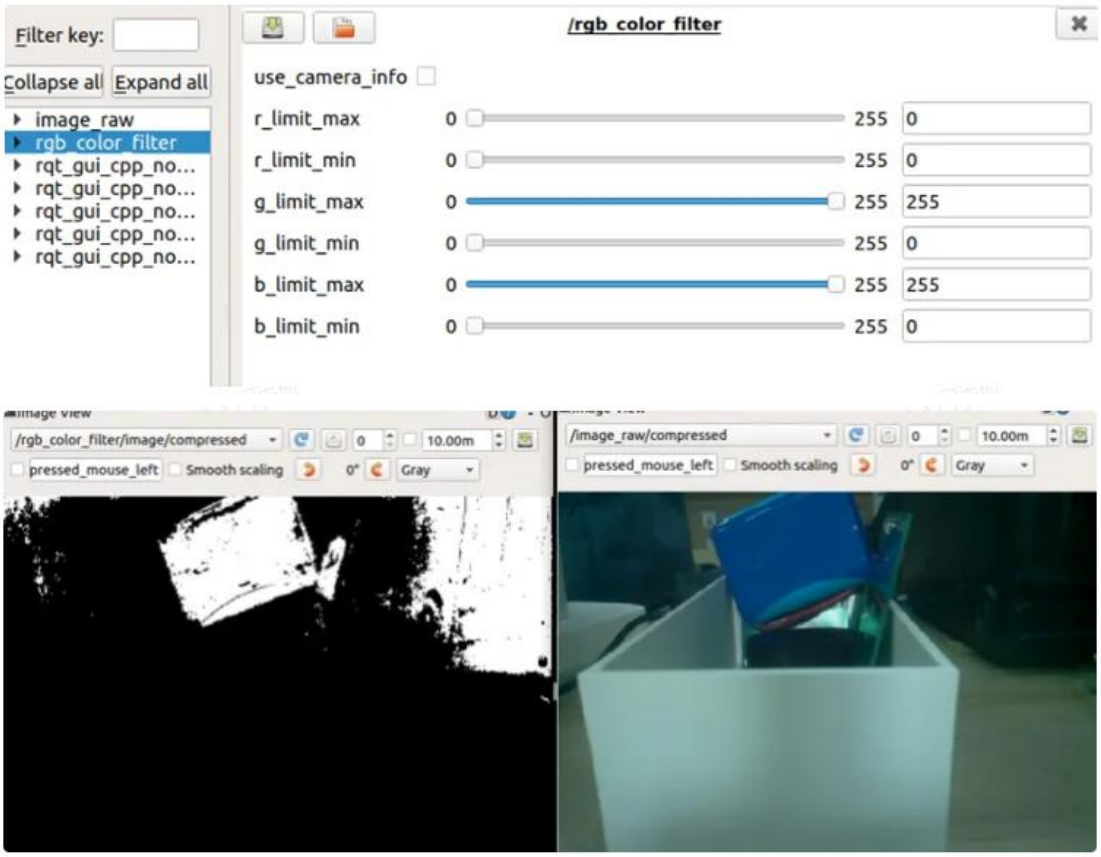
This section introduces the operation of the color filter class demo in OpenCV

RGB color space filtering RGB color_filter

Robot end starts camera `roslaunch robot_vision robot_camera.launch`
Start RGB color filter on the robot end.`roslaunch robot_vision rgb_color_filter.launch`
On the PC side, open two `rqt_image_view` tools on two terminals and subscribe to /image_raw/compressed and /rgb_color_filter/image/compressed.
Here are the filtered and original images, respectively



The so-called color filter refers to analyzing the colors in the original image on the right, for example, we make a parameter adjustment to the green color of the battery.
Start the dynamic parameter tuning tool `rosrun rqt_reconfigure rqt_reconfigure` on the PC side
Here, we can display only the blue battery by modifying the parameters for red, green, and blue.

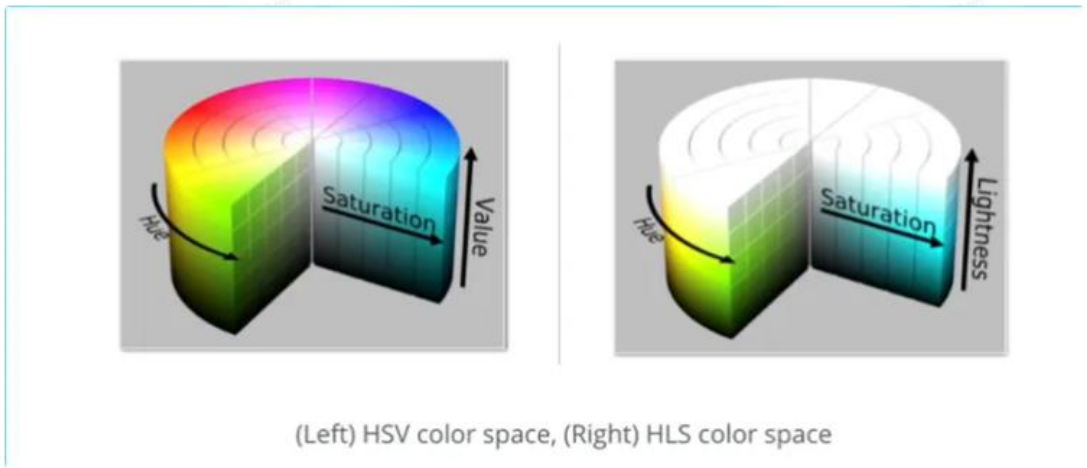


Careful friends here will notice that when we adjust the display of blue, we do not simply turn off red and green, leaving only blue. Moreover, the output image is not only displayed in blue, but also in the black areas on the edges.

This is because the colors in nature are all mixed together through different ratios of red, green, and blue, such as (red)+(green)=(yellow), etc. The blue we see is not pure blue. If only blue is left, all other colors will be turned off and nothing will be displayed. As a result, two other color description methods, HSV and HLS, have been derived

HSV: Hue hue, Saturation saturation, Value brightness

HLS: Hue hue, Saturation saturation, Lightness brightness



(Left) HSV color space, (Right) HLS color space

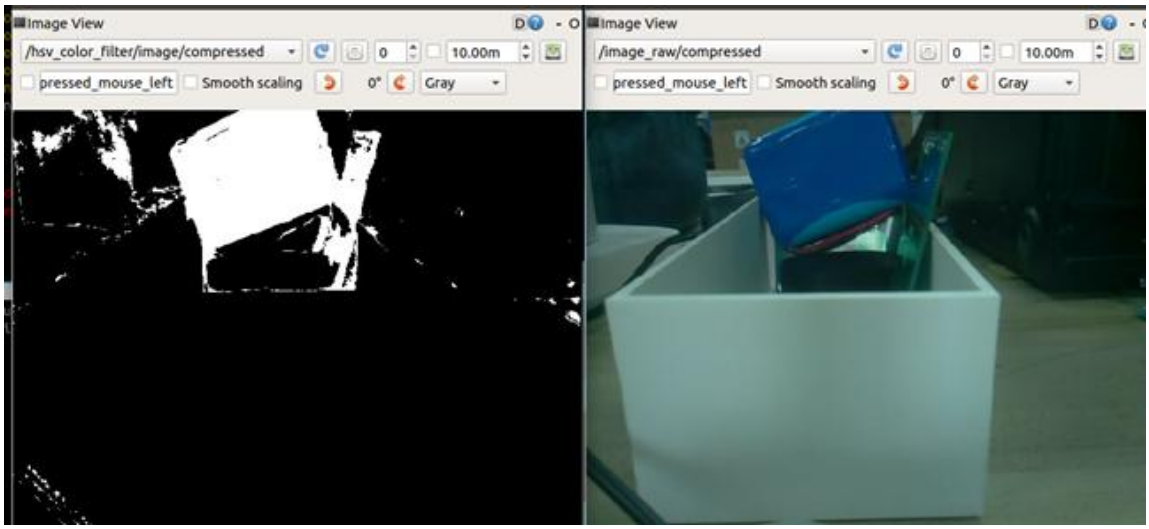
Hsv color space filtering hsv_color_filter

Robot end starts camera `roslaunch robot_vision robot_camera.launch`

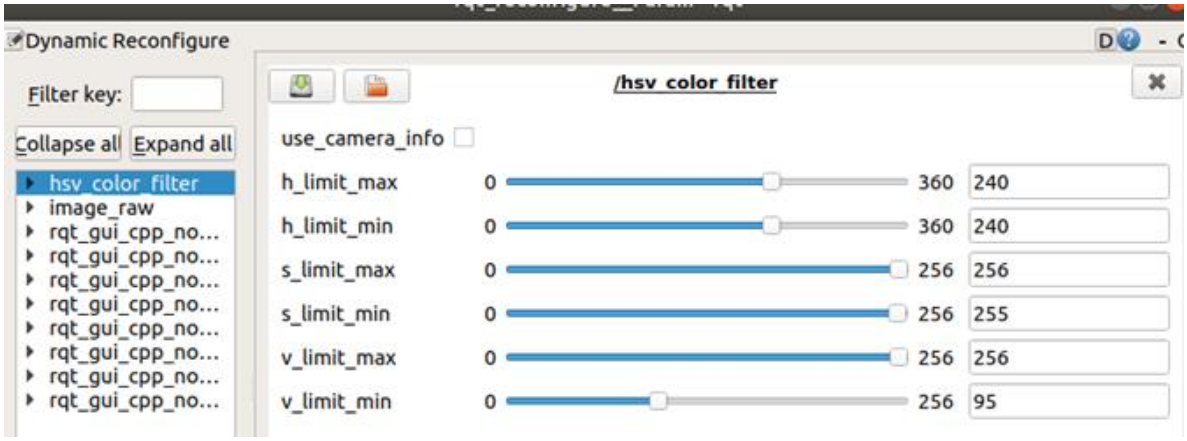
Start HSV color filter on the robot end. `roslaunch robot_vision hsv_color_filter.launch`

On the PC side, open two rqt_image_view tools on two terminals and subscribe to /image_raw/compressed and/ /hsv_color_filter/image/compressed, respectively

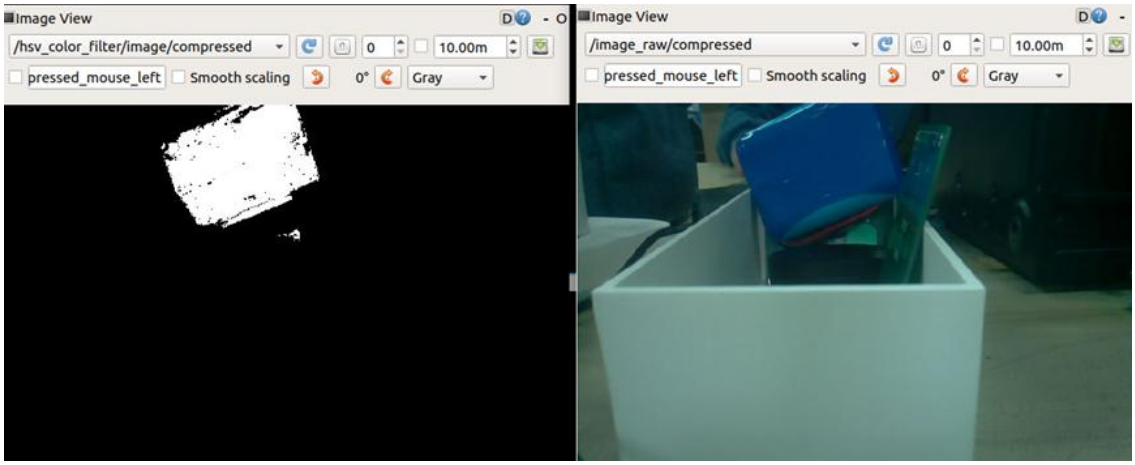
This is the screen displayed when the image tool was first opened, and you can see that in addition to blue, some other colors are also displayed.



At this point, if we want to only display blue, according to the HSV color space. The lower the saturation, the closer it is to white light, while the lower the brightness, the darker it is and the closer it is to black light. According to the order of red, yellow, green, blue, and purple, the lower the hue, the closer it is to red light. So the blue we want to display here belongs to light blue, which is relatively bright, that is, about medium brightness, and it is not close to white light, which means it is a color with extremely high saturation and hue. So, we can directly filter these three parameters. Choose a high hue of 240, with extremely high saturation and a moderate brightness. This way, only the blue parameters will be displayed and adjusted.

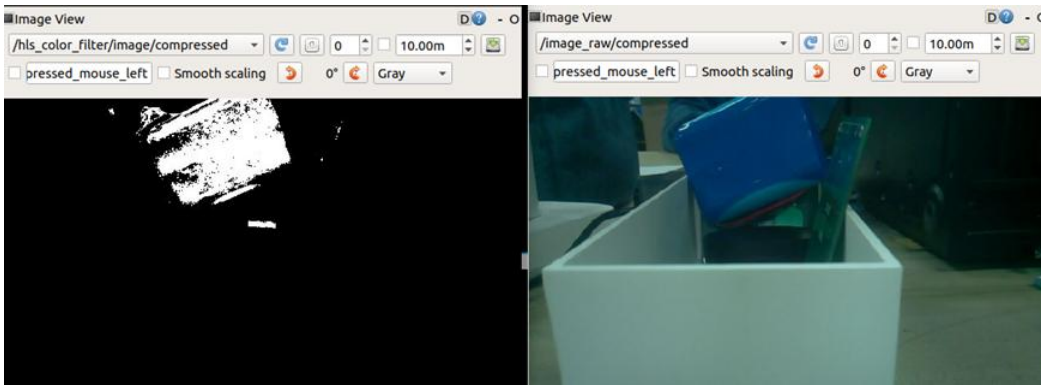


After adjustment, the blue color can be filtered out relatively stably.



Hls color space filtering hls_color_filter

Robot end starts camera `roslaunch robot_vision robot_camera.launch`
Start HLS color filter on the robot end `roslaunch robot_vision hls_color_filter.launch`
On the PC side, open two `rqt_image_view` tools on two terminals and subscribe to /image_raw/compressed and /hls_color_filter/image/compressed, respectively



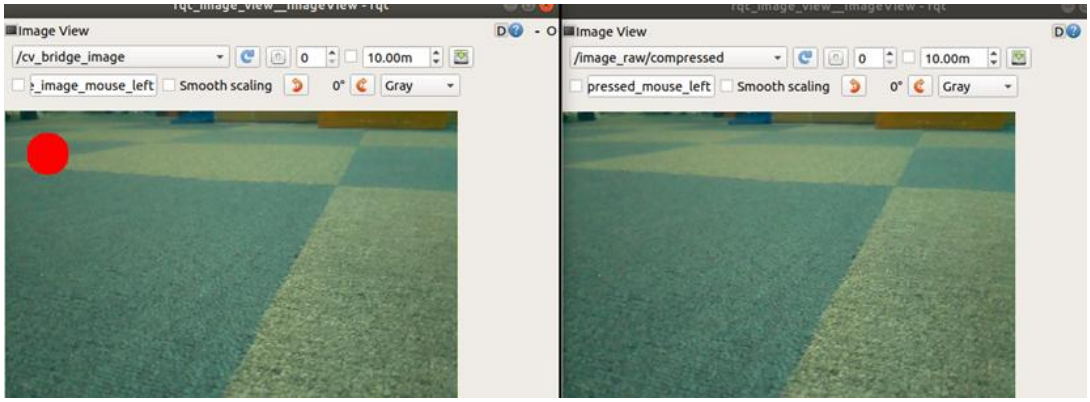
The dynamic tuning of HLS color space filters is similar to the HSV tuning process, and will not be demonstrated here.

OpenCV - Simple Image Processing Class

This section introduces the application of simple image processing classes in OpenCV

Converting Image Format and ROS Topic Format in OpenCV

Robot end starts camera `roslaunch robot_vision robot_camera.launch`
Start `rosrun robot_vision cv_bridge_test.py` on the PC side
On the PC side, open two `rqt_image_view` tools on two terminals and subscribe /image_raw/compressed and /cv_bridge_image, respectively



It can be seen that the topic on the left has an additional dot compared to the right. It was subscribed to by the cv_bridge_test node, which then converted the image_raw topic into OpenCV data and drew a dot on the image. Then convert it to the topic of ROS and publish the results.

Using the rqt_graph tool on the PC side allows you to see the relationship between image nodes and topics

The relationship between

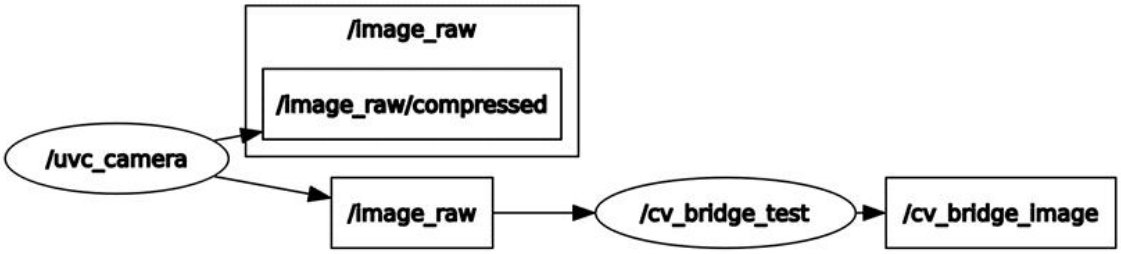


Image overlay:

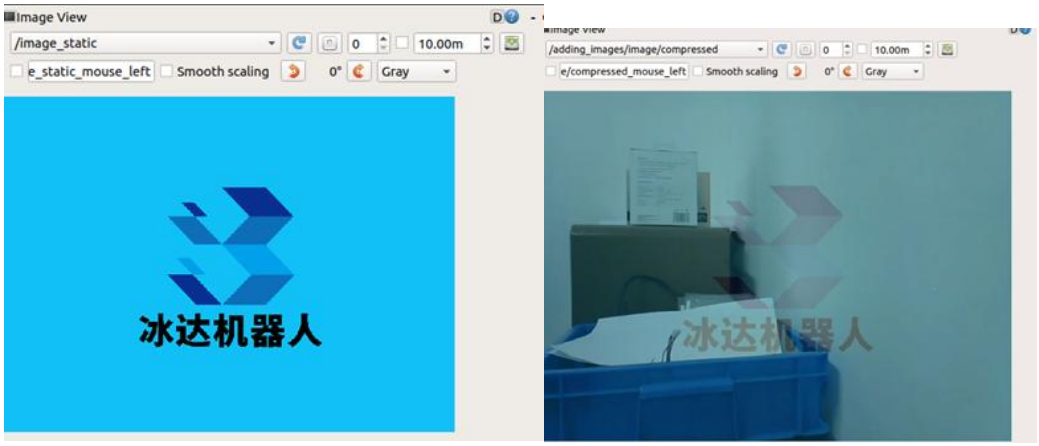
Robot end starts camera roslaunch robot_vision robot_camera.launch

Robot end starts virtual camera roslaunch robot_vision fake_camera.launch

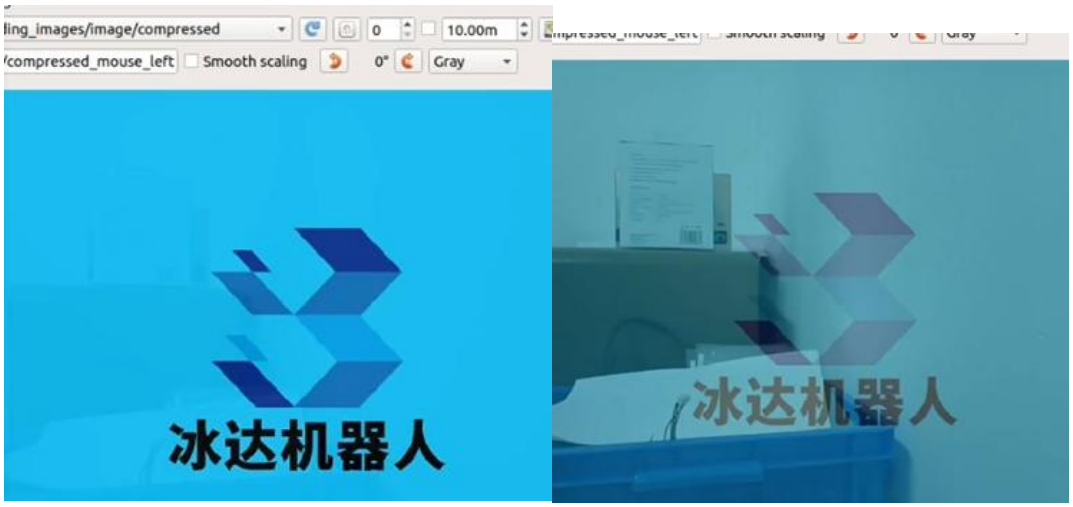
PC Start Image Overlay roslaunch robot_vision adding_images.launch

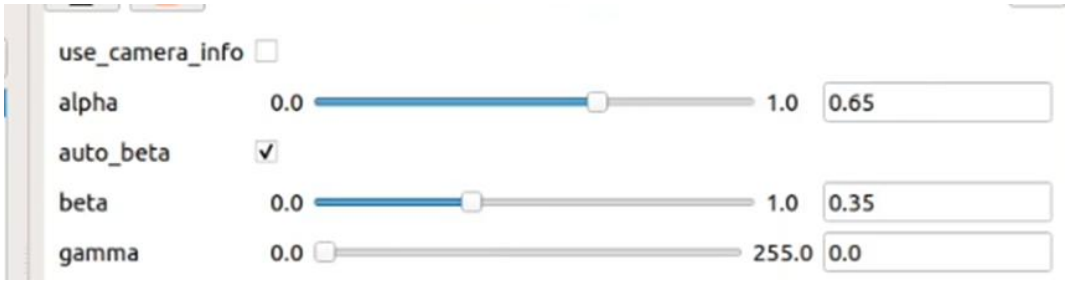
On the PC side, open two rqt_image_view tools using two terminals, subscribe to /image_static and/ /adding_images/image/compressed, respectively

The left side shows static images published by virtual cameras, and the right side shows the effect of overlaying real camera images and static images



Start the dynamic parameter tuning tool `rosrun rqt_reconfigure rqt_reconfigure` on the PC to modify transparency





By using dynamic tuning tools, the transparency can be changed to achieve the above image effects

Visual Example - Robot Vision Trajectory

This section demonstrates a visual trajectory example of robot vision

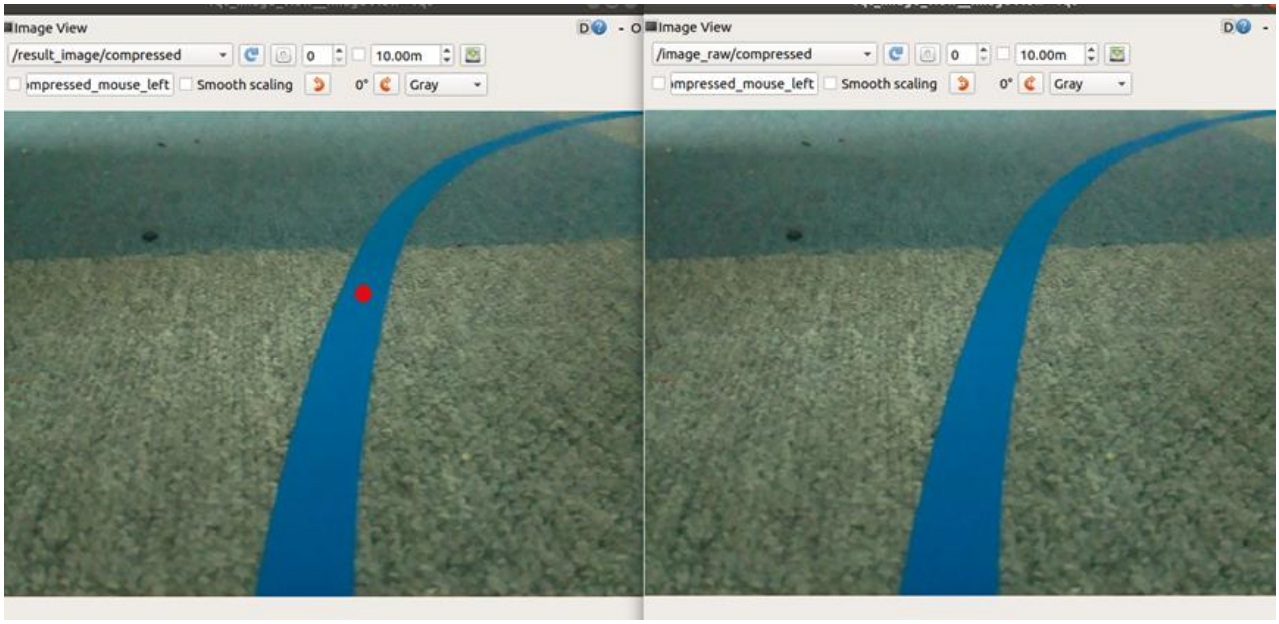
Preparation work: Set up an area as a circulation area, as shown in the following figure. Use blue tape to stick a route on the ground (the maximum turning radius of the robot is 0.5m, do not stick it too twisted), adjust the robot camera to tilt towards the ground by about 30 degrees, and place the robot on the route.

Robot end starts camera `roslaunch robot_vision robot_camera.launch`

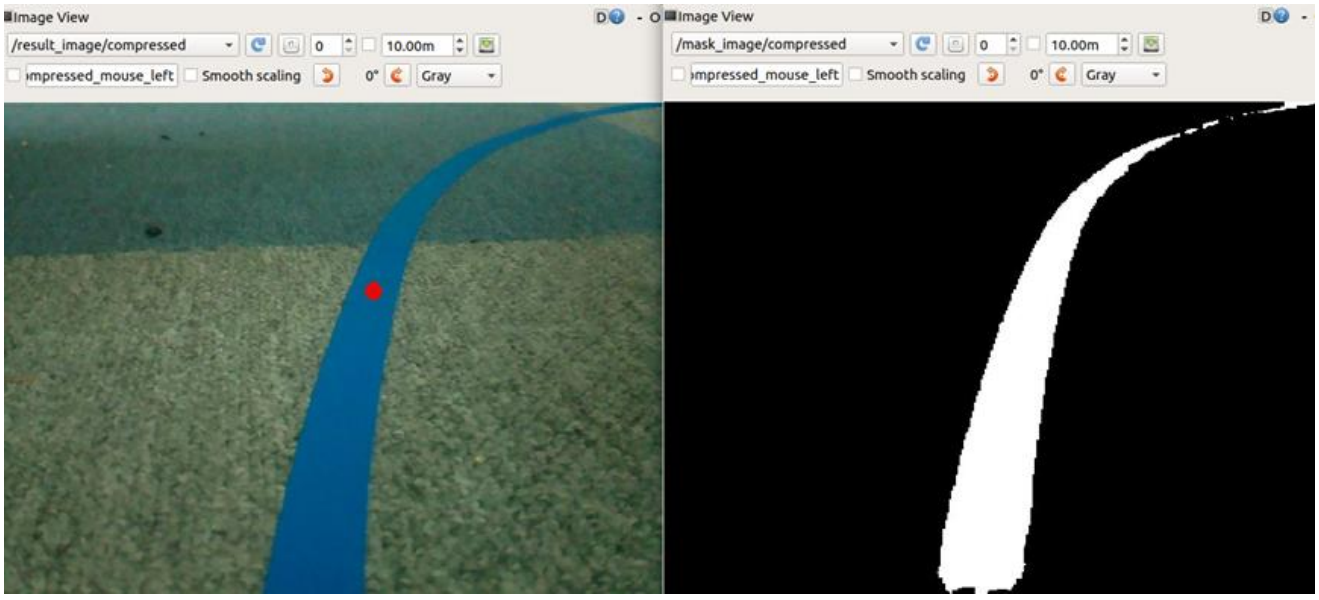
Robot end starts visual trajectory `roslaunch robot_vision line_follow.launch`

On the PC side, open two `rqt_image_view` tools on two terminals and subscribe to /image_raw/compressed and /result_image/compressed, respectively .

The image on the left is the result of following the line, the red dots indicate the robot's positioning at the center of the line, and the image on the right is a normal camera image.



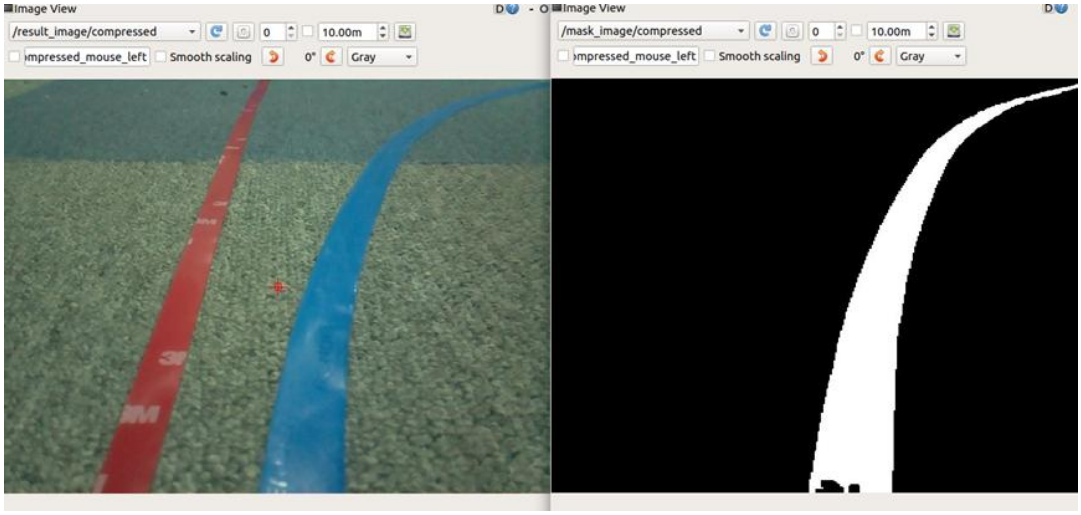
Change the window on the right to subscribe /mask_image/compressed to display the position of the line found by the robot



Robot end startup chassis `roslaunch base_control base_control.launch`
The robot will automatically start walking along the line to the end.

Visual Example - Adjusting the Line Color

This section introduces the adjustment of line colors. In the previous section, we demonstrated a visual line example of robot vision, where blue lines are used by default in the routine. Next, we will introduce how to adjust if you want to use lines of other colors.
First, place a line of other colors next to the blue line, using red as an example
Robot end starts camera `roslaunch robot_vision robot_camera.launch`
The robot starts the routing node and passes in the test_mode parameter:
`roslaunch robot_vision line_follow.launch test_mode:=true`
On the PC side, open two rqt_image_view tools on two terminals and subscribe to /mask_image/compressed and /result_image/compressed respectively
You can see that there is a red center in the left image, which is the center of the entire image. The image on the right is the blue line found by the robot.



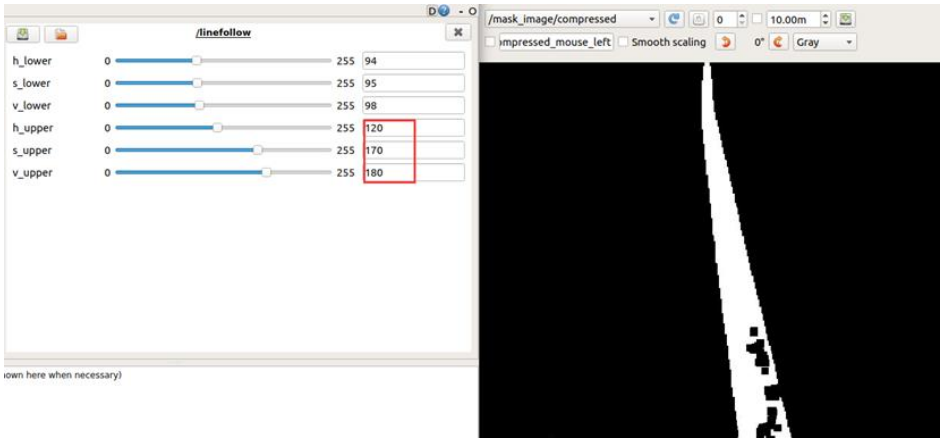
When starting the trajectory, we passed in the test_mode parameter, which will continuously output the HSV color space value at the red centroid in the left image. For example, what is currently displayed is the HSV value on the ground.

```
[INFO] [1616583845.658537]: Point HSV Value is [ 65  49 125]
[INFO] [1616583845.701818]: Point HSV Value is [ 67  54 127]
[INFO] [1616583845.740481]: Point HSV Value is [ 72  51 124]
```

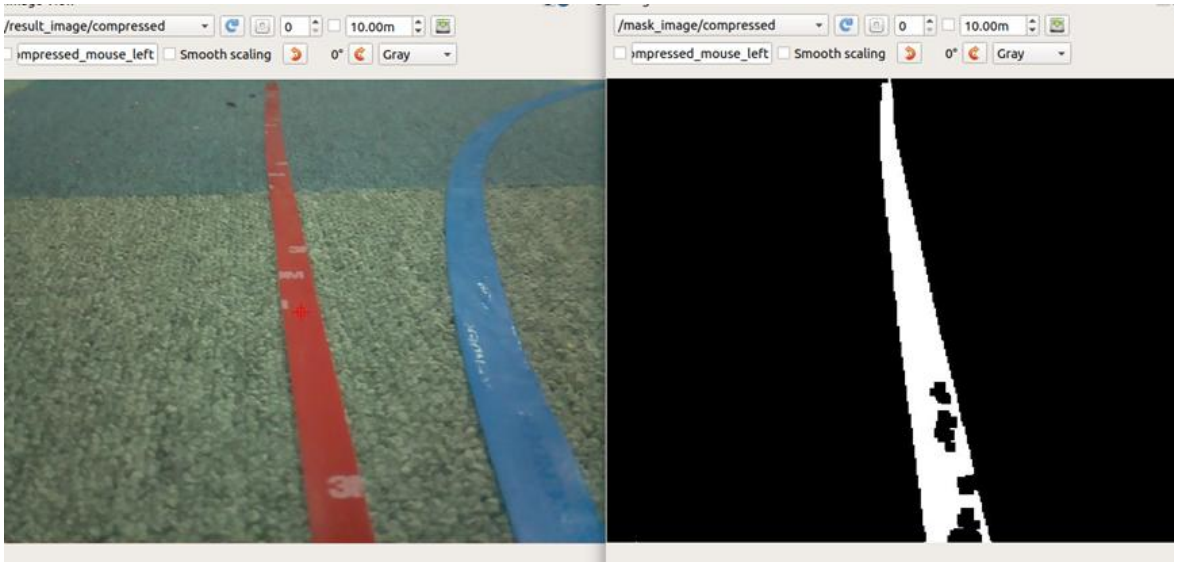
Move the position of the robot so that the center of gravity is aligned with the red line. At this point, the information has become the HSV value of the red line.

```
[INFO] [1616584625.897093]: Point HSV Value is [118 165 176]
[INFO] [1616584625.938777]: Point HSV Value is [117 167 176]
[INFO] [1616584625.980453]: Point HSV Value is [118 167 177]
[INFO] [1616584626.020085]: Point HSV Value is [117 172 179]
[INFO] [1616584626.062261]: Point HSV Value is [118 166 180]
[INFO] [1616584626.107527]: Point HSV Value is [118 168 178]
```

Start the dynamic parameter tuning tool `rosrun rqt_reconfigure rqt_reconfigure` on the PC side

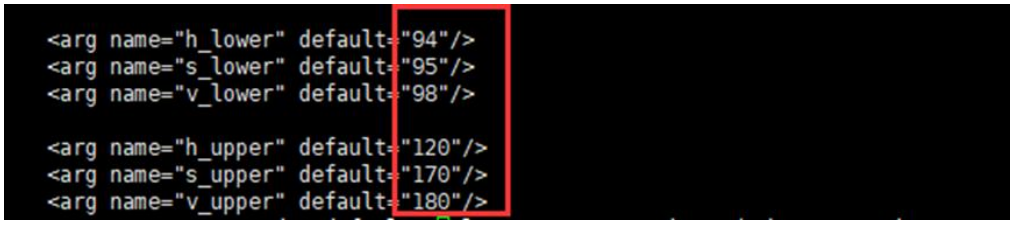


The upper limit of HSV parameters must be strictly limited to the detected parameter range, because the red line I used here also has a white log above it, so the lower limit should be appropriately relaxed to improve the image acquisition effect. You can see that the image output from the /mask_image/compressed topic only shows the position of the red line.



Because the result of dynamic parameter tuning is one-time and only valid for this application, if you want it to continue to take effect, you need to directly modify the launch file on the robot side.

```
roscd robot_vision/launch
vim line_follow.launch
```



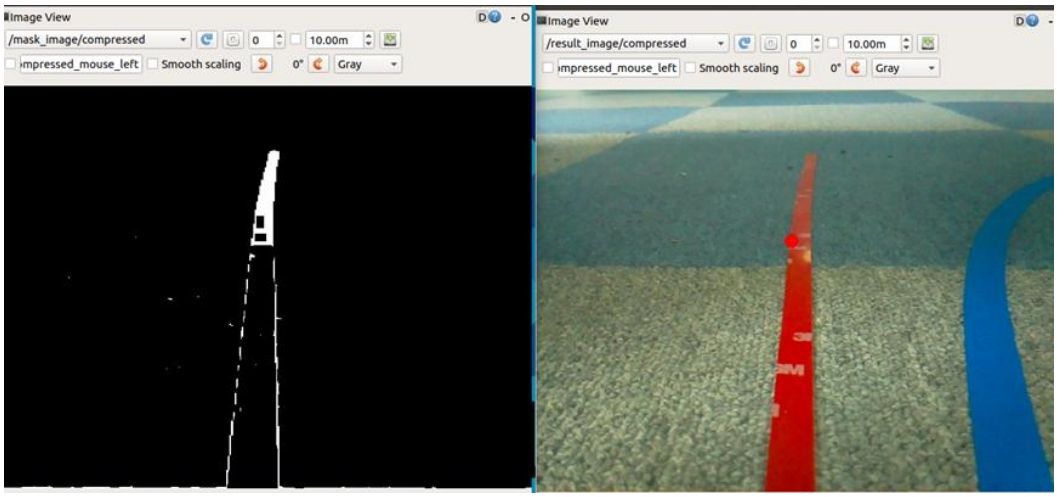
```
<arg name="h_lower" default="94"/>
<arg name="s_lower" default="95"/>
<arg name="v_lower" default="98"/>

<arg name="h_upper" default="120"/>
<arg name="s_upper" default="170"/>
<arg name="v_upper" default="180"/>
```

Change the HSV value in the line_follow.launch file directly to the value of the red line

Robot end starts camera `roslaunch robot_vision robot_camera.launch`

Robot end starts visual trajectory `roslaunch robot_vision line_follow.launch`

On the PC side, open two `rqt_image_view` tools on two terminals and subscribe to /mask_image/compressed and /result_image/compresse, respectively



After rewiring here, only the red line will be displayed. Due to changes in lighting, the display effect has been affected. If you encounter any issues while operating it yourself, you can modify the parameters and adjust them again.

3. Radar mapping and navigation

Activate LiDAR and data viewing

This section introduces the startup of LiDAR and the viewing of radar data

Start the laser radar on the robot end by launching the `roslaunch robot_navigation lidar.launch`

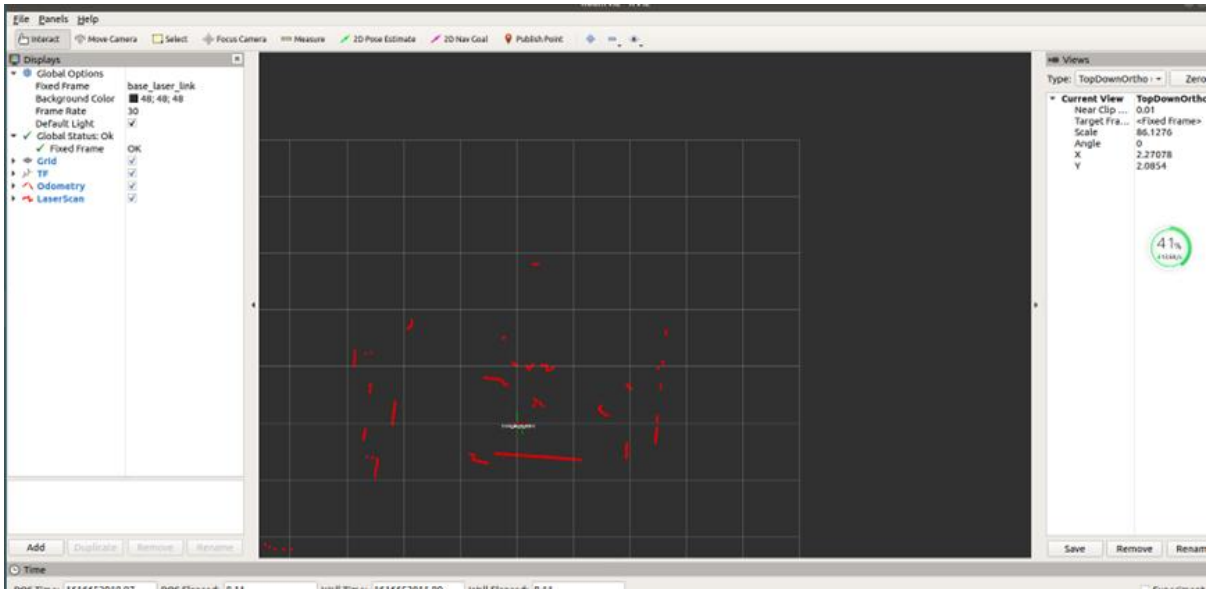
```
bingda@robot:~$ rostopic list
/odom
/rosout
/rosout_agg
/scan
/tf
bingda@robot:~$
```

At this point, when viewing the topic list, you will find an additional scan topic, which is the one released by Radar. Seeing this topic indicates that the radar has started up normally.

By viewing scan topics, you can view radar data

f, inf, 2.759999990463257, inf, 2.757999897003174, inf, 2.7569642066955566, inf, 2.759164333343506, inf, 2.75900006294
2505, inf, 2.759824752807617, inf, inf, inf, inf, inf, inf, inf, inf, inf, 6.037838935852051, inf, 6.031000137329102,
inf, 6.050000190734863, inf, 6.078999996185303, inf, 6.114429950714111, inf, inf, 6.138034820556641, inf, 6.1710000038
14697, inf, 6.1020002365112305, inf, 5.933000087738037, inf, 5.763999938964844, inf, 5.743000030517578, inf, 5.7328095
43609619, inf, inf, inf, inf, inf, inf, inf, inf, inf, inf, inf, inf, inf, inf, inf, inf, inf, inf, inf, inf, inf, inf
, inf, inf, inf, inf, inf, inf, inf, inf, inf, inf, inf, inf, inf, inf, inf, inf, inf, inf, inf, inf, inf, inf, i
nf, inf, inf, inf, inf, inf, inf, inf, inf, inf, inf, inf, inf, inf, inf, inf, inf, inf, inf, inf, inf, inf, inf,
inf, inf, inf, inf, inf, inf, inf, inf, inf, inf, inf, inf, inf, inf, inf, inf, inf, inf, inf, inf, inf, inf, in
f, inf, inf, inf, inf, inf, inf, inf, inf, inf, inf, inf, inf, 0.15800000727176666, inf, 0.15600000321865082, inf
, inf, inf, inf, inf, inf, inf, inf, inf, 0.48170164227485657, inf, inf, 0.47999998927116394, inf, 0.4773703
5155296326, inf, 0.47540542483329773, inf, 0.4732257127761841, inf, 0.47175121307373047, inf, 0.4702267348766327, inf,
0.46912601590156555, inf, 0.4674595594406128, inf, inf, inf, inf, inf, 3.4059998989105225, inf, 3.4159998893737793, i
nf, inf, inf, 3.413959264755249, inf, 3.4668402671813965, inf, 3.5372488498687744, inf, inf, 3.5889999866485596, inf,
3.6429998874664307, inf, 3.5980000495910645, inf, 3.593397855758667, inf, 3.613375663757324, inf, 3.6138312816619873,
inf, 3.647494077682495, inf, inf, inf, 3.7284302711486816, inf, 3.7160000801086426, inf, 3.7294065952301025, inf, 3.77
86104679107666, inf, 3.7939999103546143, inf, 3.816999912261963, inf, 3.5339999198913574, inf, 3.4290356636047363, inf
, inf, inf, inf, inf, inf, inf, inf, inf, inf, inf, inf, inf, inf, inf, inf, inf, inf, inf, inf, inf, inf, inf, i
nf, inf, inf, inf, inf, inf, inf, inf, inf, inf, inf, inf, inf, inf, inf, inf, inf, inf, inf, inf, inf, inf, 5.38
7139797210693, inf, 5.370999813079834, inf, 5.248000144958496, inf, 5.162191867828369, inf, 5.117181301116943, inf, 5.
051000118255615, inf, 4.974999904632568, inf, 4.853000164031982, inf, inf, 4.745999813079834, inf, 4.690000057220459,
inf, 4.626636028289795, inf, 4.551224708557129, inf, 4.493000030517578, inf, 4.429018974304199, inf, 4.362114429473877
, inf, 4.302530288696289, inf, 4.252521991729736, inf, 4.196000099182129, inf, 4.135000228881836, inf, 4.0909218788146
97, inf, 4.04374885559082, inf, inf, 4.001796245574951, inf, 3.9555840492248535, inf, 3.9117767810821533, inf, 3.86500
0009536743, inf, 3.8420000076293945, inf, 3.809999942779541, inf, 3.7639999389648438, inf, 3.7239999771118164, inf, 3.
698667049407959, inf, 3.676708698272705, inf, 3.652229070663452, inf, 3.6220250129699707, inf, 3.5959999561309814, inf
, 3.571000099182129, inf, 3.553845167160034, inf, 3.537242650957178, inf, inf, 3.5220000743865967, inf, 3.50367593765

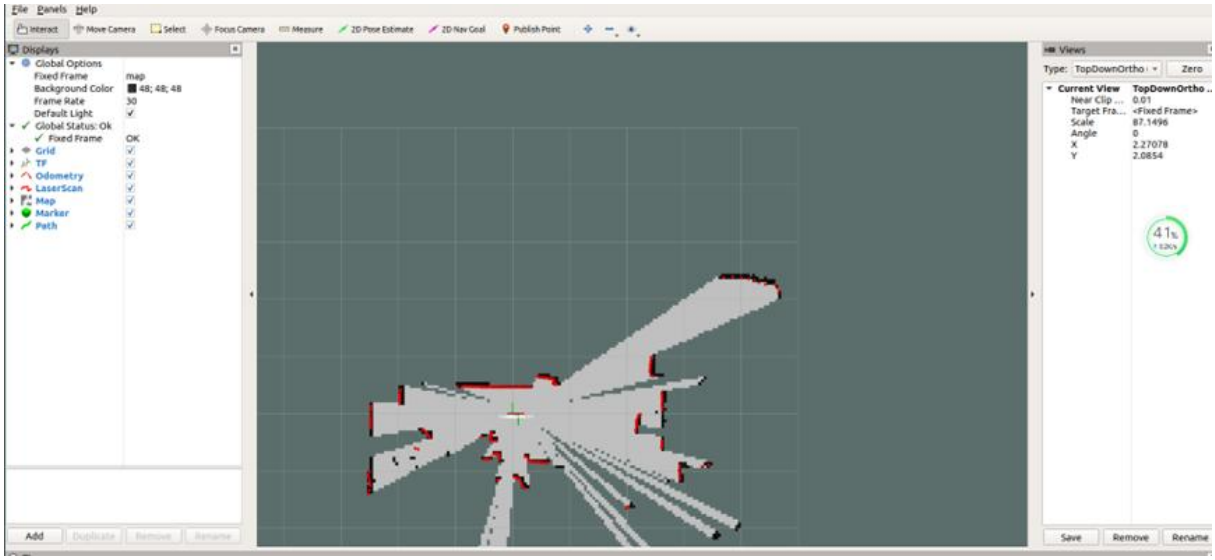
Run the rviz tool on the PC, `roslaunch robot_navigation lidar_rviz.launch`



Here you can see the radar data, which is scanned by a laser around the surface. Draw obstacles in the form of points based on their distance. (Radar is a single line scanning area that only covers the plane at the height of the radar transmission window, and objects at or below this height cannot be scanned.)

Robot running laser SLAM

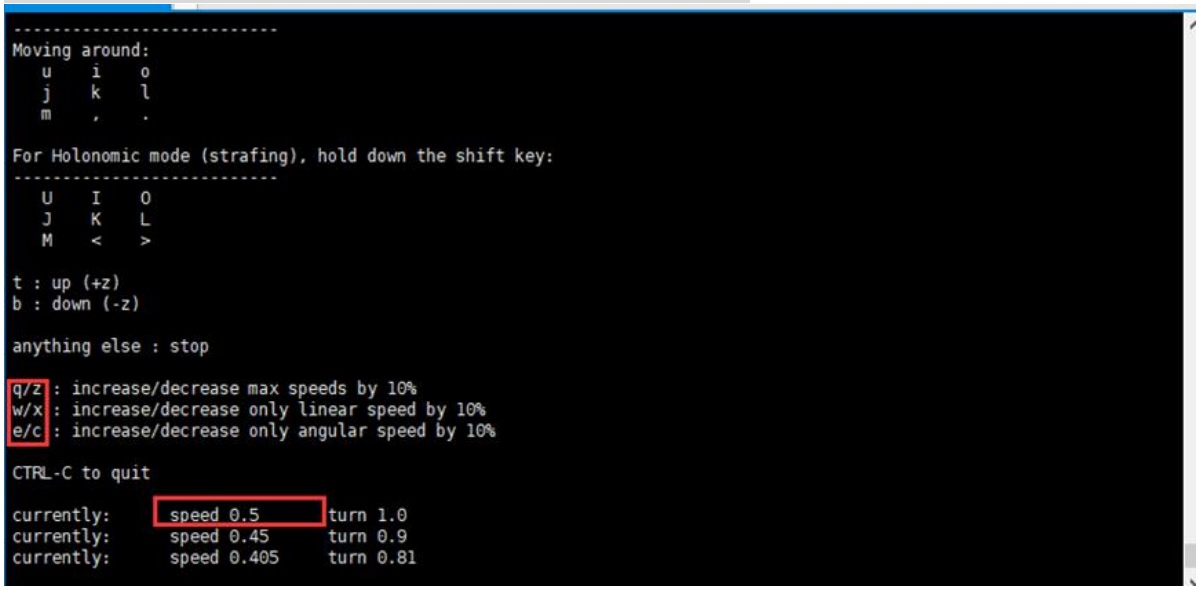
This section demonstrates the SLAM mapping function of LiDAR
Execute the slam launch file on the robot side
`roslaunch robot_navigation robot_slam_laser.launch`
Execute rviz graphical monitoring program on PC (load rviz configuration file)
`roslaunch robot_navigation slam_rviz.launch`



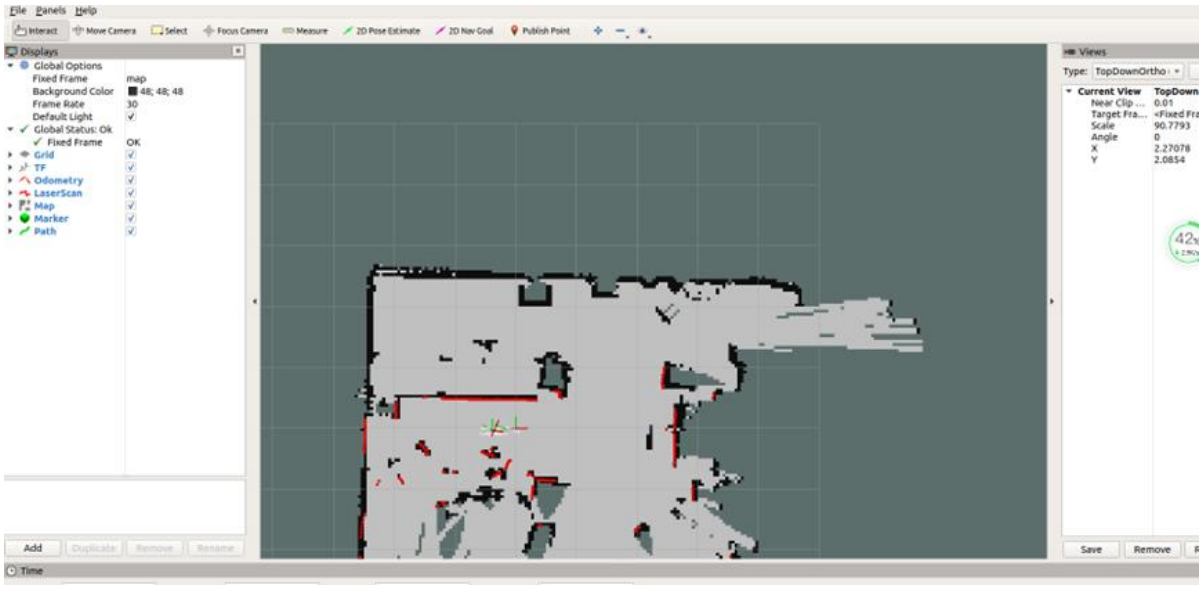
The radar SLAM mapping has started normally, and a preliminary map centered on the robot can be seen under the rviz tool. Next, we will have the robot move its position to refine the entire map.

Start a keyboard control program on the PC or robot end

```
rosrun teleop_twist_keyboard teleop_twist_keyboard.py
```

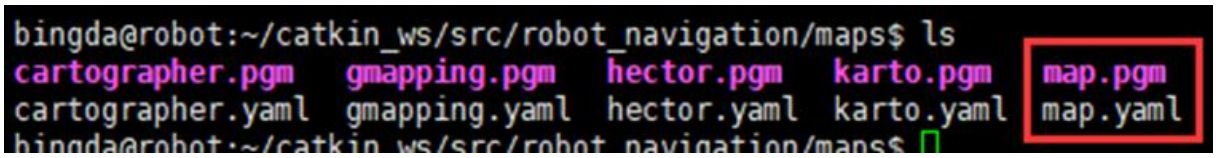


Here, we follow the prompts to press the z key to reduce the robot's movement speed (it is recommended to lower the remote control speed to improve the mapping effect, and the speed should not exceed 0.25m/s. Note that the mouse cursor needs to be kept in the shell window where the keyboard control program is running, otherwise the relevant buttons will be invalid)



The black border in the map represents the walls in the actual terrain, which are displayed in the form of boundaries in the map. When the boundaries around the map are completely closed or when you want to save the map, open a new control terminal on the robot end (if you execute the following command on the PC end, the robot end will not save the map file, and there will be navigation problems in the next chapter), enter the location where you want to save the map, and then save the map. Here, take the maps under the robot navigation function package as an example (if you are not familiar with this aspect, it is recommended to follow the command line below, because the navigation program later also uses this path and file name)

```
roscd robot_navigation/maps
rosrun map_server map_saver -f map
```



After saving, there will be two files in the maps folder: map.pgm and map.yaml. At this point, the mapping process is complete.

Running LiDAR navigation and obstacle avoidance

This section introduces the routine of using LiDAR for navigation and obstacle avoidance

Run the launch file for navigation on the robot side

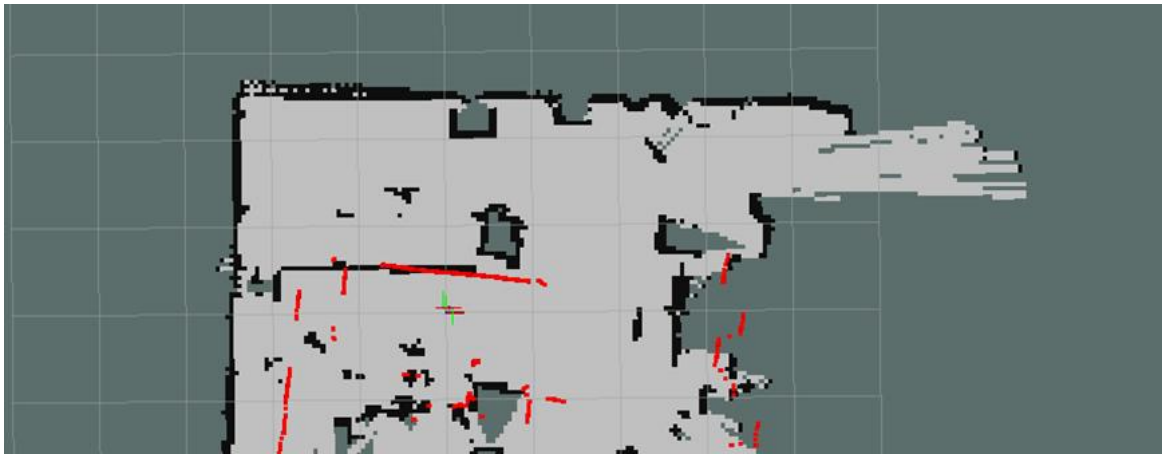
```
roslaunch robot_navigation robot_navigation.launch
```

(It should be noted that the robot navigation uses the default map file saved in the default path after creating the map in the previous chapter. If the map name and path saved in the previous chapter are customized, they cannot be run directly. It is recommended to change to the default path and file name.)

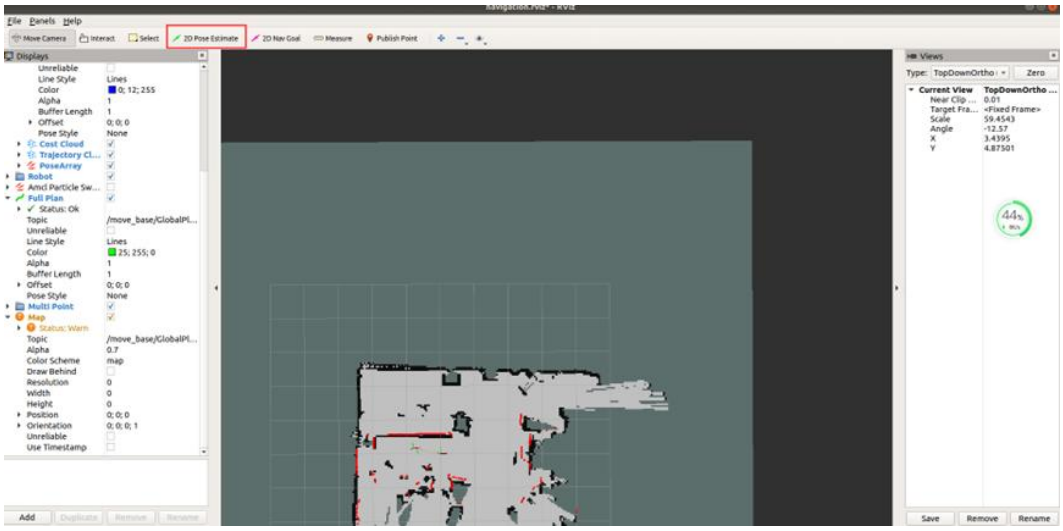
Execute rviz graphical monitoring program on PC

```
roslaunch robot_navigation navigation_rviz.launch
```

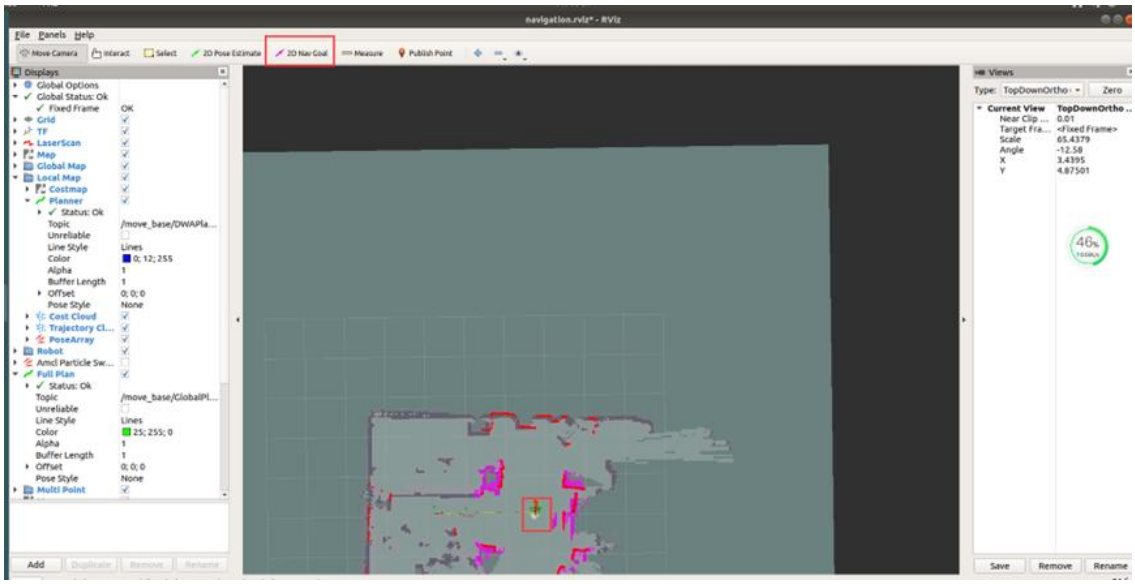
After starting the PC, the rviz will display as shown in the picture. The current actual position of the robot may not match the position on the map, and the robot position needs to be manually set.



Click on the "2D Pose Estimate" button on the rviz in the red box in the diagram, then left click on a point on the map and pull the mouse to set the direction. After releasing the mouse, the point just clicked and the selected direction are the current position of the robot, and the corresponding radar data will also be updated, as shown in the following figure

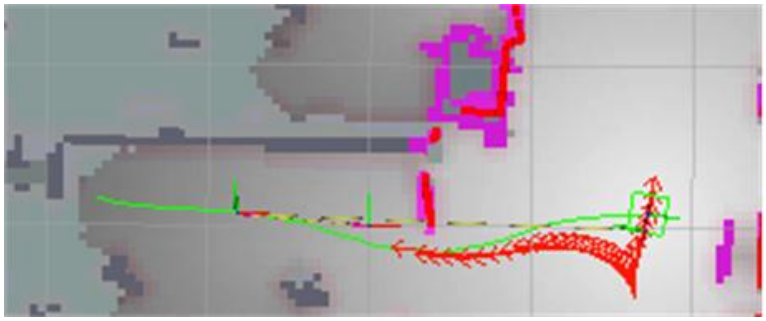


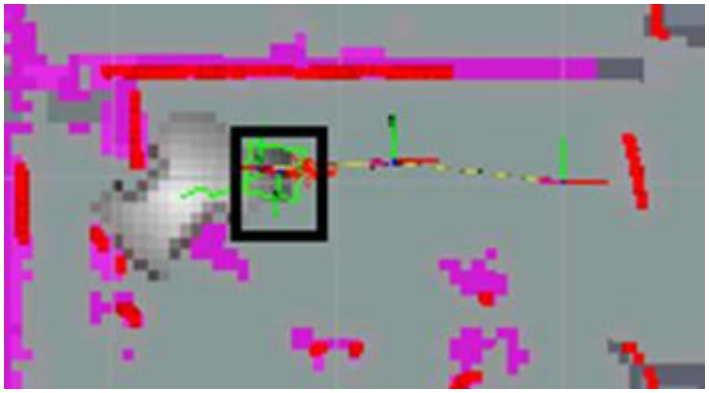
After clicking the "**2D Nav Goal**" button on rviz, use the mouse to left click on a point on the map to set it as the navigation target point. When the left button is not released, pull the mouse to specify the direction of the target point. After releasing it, the robot starts executing navigation tasks, and there is also relevant log output in the terminal where the robot runs the launch file.



At this point, the robot has reached the designated target point. We place an obstacle (black boxed area) on its way back and let the robot navigate back to its previous position.

At this point, observing the path planning of the robot, it can be found that it has planned a path to avoid obstacles and successfully reached the target point





Switching of laser SLAM mapping algorithm

This section introduces the switching of SLAM mapping algorithm

There are many algorithms for SLAM mapping, and in section 4.3.2, we demonstrated one of them. Let's start it again:

`roslaunch robot_navigation robot_slam_laser.launch` on the robot end

At this point, we look at the nodes that have already been started, which include a chassis control node base_control, a coordinate conversion node for the LiDAR chassis base_footprint_to_laser, and a LiDAR node sc_mini. Among them, gmapping is the node of the robot's mapping algorithm.

```
bingda@robot:~$ rosnode list
/base_control
/base_footprint_to_laser
/gmapping
/rosout
/sc_mini
bingda@robot:~$
```

Now let's try replacing the mapping algorithm gmapping with another algorithm

Launching robot navigation robot slam laser on the robot end. `roslaunch robot_navigation robot_slam_laser.launch slam_methods:=hector`

The following errors may occur when using Hector, so there is no need to pay attention to them.

```
[ERROR] [1616728794.286010372]: Transform failed during publishing of map_o
dom transform: Lookup would require extrapolation into the future.  Request
ed time 1616728793.286439720 but the latest data is at time 1616728792.7798
29602, when looking up transform from frame [base_footprint] to frame [odom
]
```

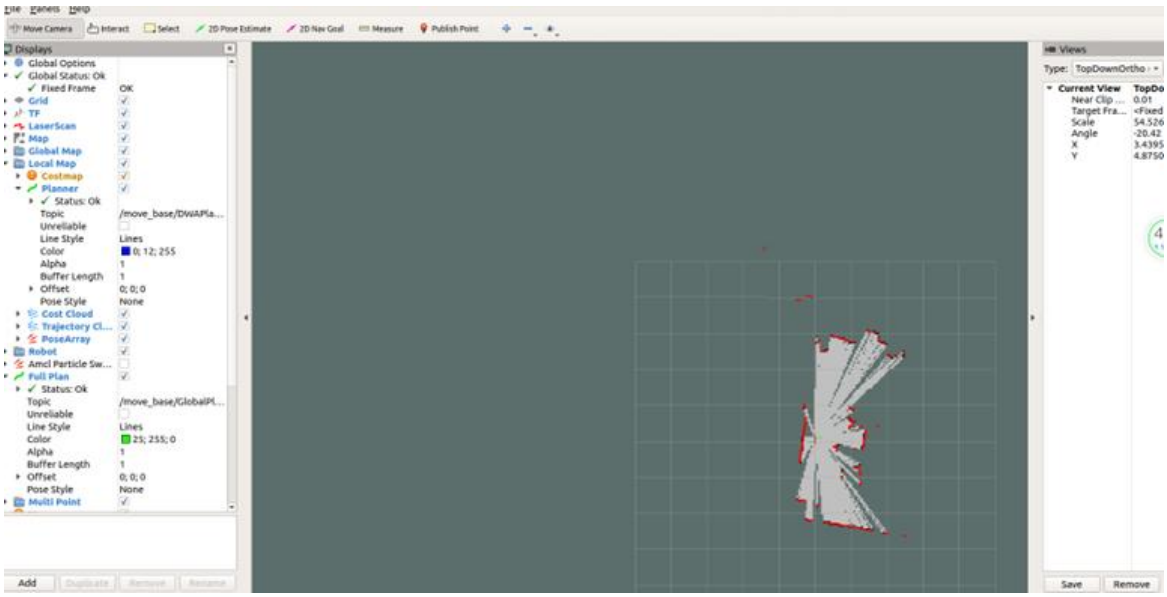
This is because we use a launch file to start the robot and mapping algorithm. However, the launch file does not guarantee the starting order of each node. Hector starts quickly, and the chassis and radar have not yet started, so there will be a lack of odom information. After the chassis starts, an error message will stop outputting.

The gmapping algorithm has been replaced with the Hector algorithm

```
bingda@robot:~$ rosnode list
/base_control
/base_footprint_to_laser
/hector_mapping
/rosout
/sc_mini
bingda@robot:~$
```

Execute rviz graphical monitoring program on PC

```
roslaunch robot_navigation navigation_rviz.launch
```



We have implemented four common mapping algorithms in the robot, namely cartographer, hector, karto, and gmapping. After starting the file, simply input the names of the four mapping algorithms mentioned above (the default state is gmapping). As follows:

```
roslaunch robot_navigation robot_slam_laser.launch slam_methods:=cartographer
roslaunch robot_navigation robot_slam_laser.launch slam_methods:=hector
roslaunch robot_navigation robot_slam_laser.launch slam_methods:=karto
```

Switching of Local Path Planning Algorithm

This section introduces the switching of local path planning algorithms

In section 4.3.3, we introduced that the LiDAR navigation of robots is achieved by running the navigation launch file on the robot end

```
roslaunch robot_navigation robot_navigation.launch
```

The local path planning in this launch defaults to using the TEB path planner

The so-called local path planning is relative to global path planning, which is the complete path planning between the robot's starting point and target point. Due to the differences in the kinematic characteristics of robots themselves, they are not paths that robots can immediately execute. For example, robots with Ackermann structure are unable to turn in place. At this point, a local path planner can be used to dynamically adjust the global path planning.

Our robot has implemented the use of two mainstream path planners, DWA and TEB. The TEB algorithm considers that the minimum turning radius of the robot is more suitable for Ackermann structured robots, while the DWA algorithm can achieve in situ turning and is more suitable for differential steering vehicle models. The default usage is TEB path planner. If you need to switch to DWA path planner, simply pass in the planner parameter as dwa

```
roslaunch robot_navigation robot_navigation.launch planner:=dwa
```

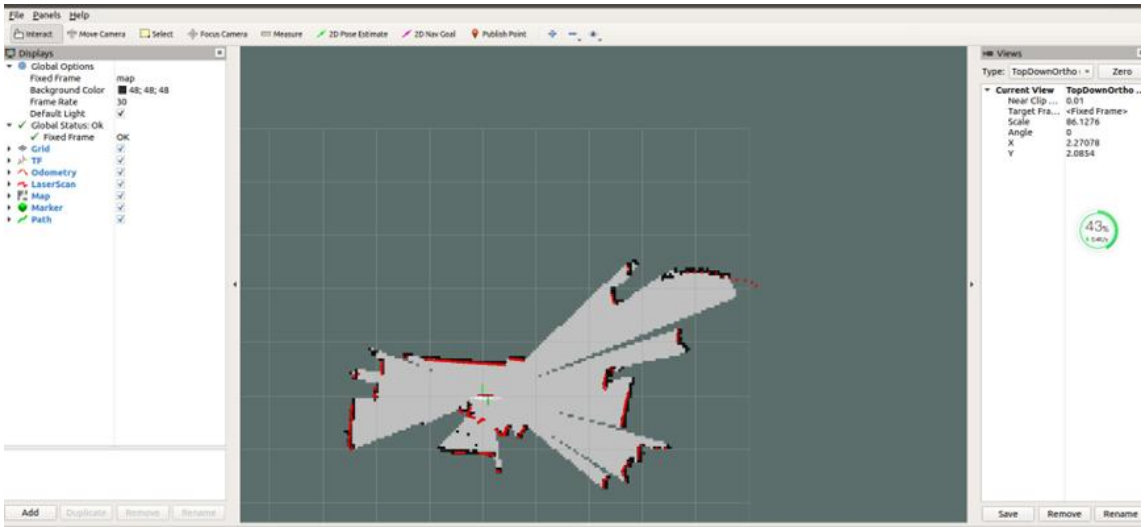
Navigation and mapping of scenes without maps

This section demonstrates the application of robots in navigation and mapping in unmapped scenarios

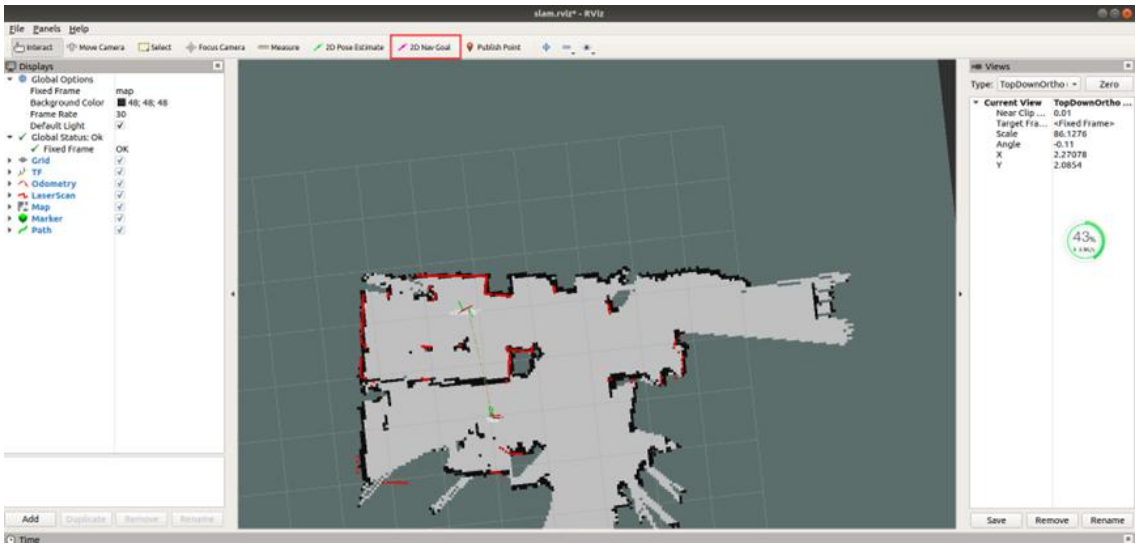
Execute the launch file of SLAM on the robot side and pass in the planner parameter as team

```
roslaunch robot_navigation robot_slam_laser.launch planner:=teb
```

Run `roslaunch robot_navigation slam_rviz.launch` on the PC side



Referring to the radar navigation method in section 4.3.3, click the "**2D Nav Goal**" button on rviz to set a target point for the robot. At this point, the robot will continuously improve the map during the navigation process.



The method of saving the map after navigation and mapping is also to save it on the robot end

```
roscd robot_navigation/maps
rosrun map_server map_saver -f map
```

The Use of Stage Robot Simulator

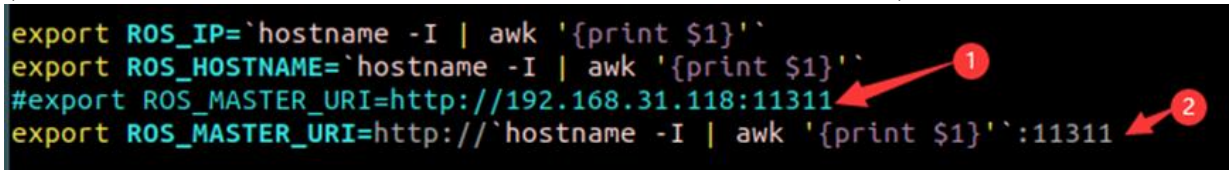
This section demonstrates the use of a stage robot simulator

In section 4.1.6, the distributed communication between robots and PC was explained. The master runs on the robot and the PC communicates with the robot through the master running on the robot. The simulation experiments in this chapter are conducted entirely on PC and do not rely on physical robots. So it is necessary to temporarily dismantle the distributed communication and let the master run on the PC.

1. Modify the ROS-MASTER_URI value in the PC~/. bashrc file, and modify the environment variables in the. bashrc file as follows

```
#export ROS_MASTER_URI=http://192.168.31.118:11311
export ROS_MASTER_URI=http://`hostname -I | awk '{print $1}'`:11311
```

(i.e. comment out the line labeled 1 and uncomment the line labeled 2)



(To be modified when conducting physical robot experiments)

2. Modify the BASE-TYPE environment variable in. bashrc to the type of robot you want to simulate (this environment variable will affect the robot model used in the simulation, using NanoCar as an example)

Export BASE-TYPE=NanoCar

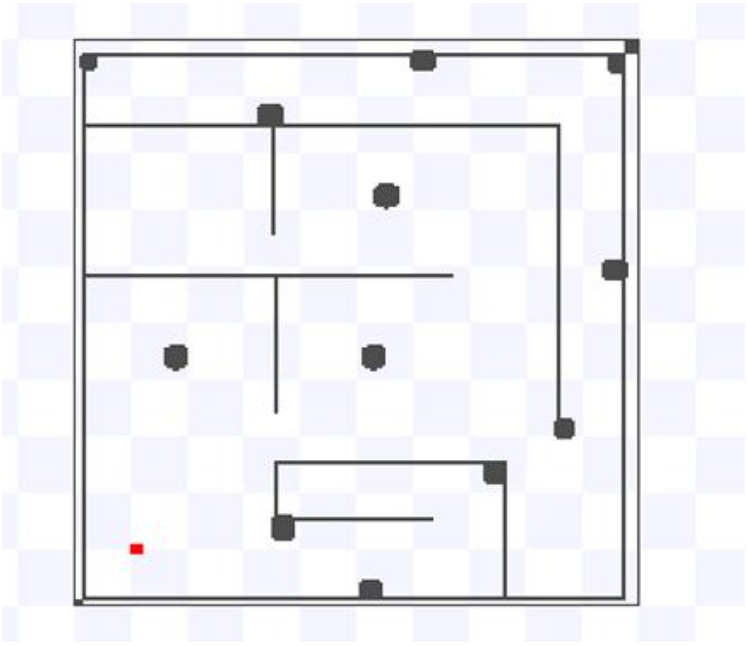
After completing the modifications, save and exit, and source to make it effective

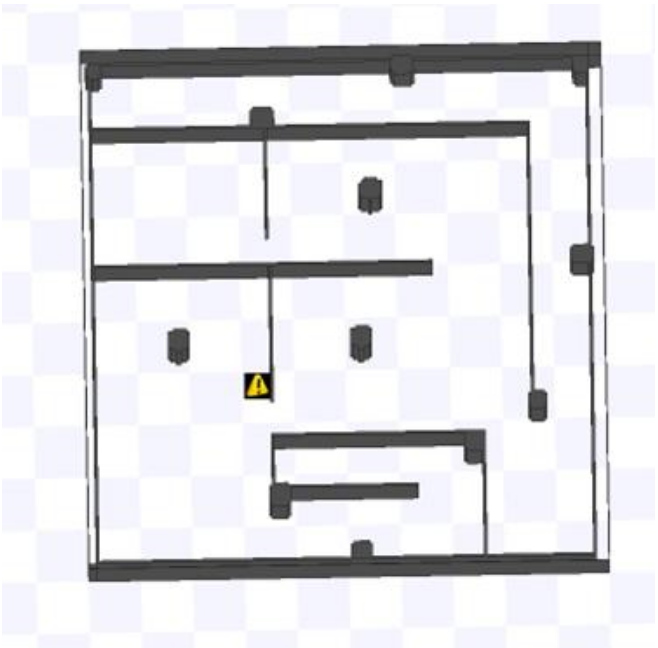
Source~/. bashrc

3. Next, start the simulation of the robot

```
roslaunch robot_simulation simulation_one_robot.launch
```

After startup, a stage window will pop up, and the red small square in the figure represents the simplified model of the robot, while the black one represents the environment model where the robot is located.





At this point, start another keyboard remote control node

```
rosrun teleop_twist_keyboard teleop_twist_keyboard.py
```

Robots can move around in the environment through keyboard remote control

Stage can simulate the collision effect of robots. If you control the robot to collide with a wall, a yellow collision mark will appear at the collision position (as shown in the above picture).

Simulation mapping navigation in stage

This section will demonstrate the LiDAR mapping and navigation of simulating robots in the stage simulator

The use of simulation function can avoid frequent testing in both physical robots and real environments during mapping and navigation algorithm development, which consumes a lot of time. Simulation can focus on algorithm development and parameter optimization. When good results are achieved in simulation, the code can be tested on real robots. According to our testing, the performance of robots in our simulation environment is almost consistent with that of physical robots.

In Chapter previous, we have introduced the application of LiDAR mapping on physical robots. This chapter uses simulation to achieve this, and the startup launch file used is the same. The difference is that here, configuration parameters need to be passed in through the launch file to enable simulation

```
roslaunch robot_navigation robot_slam_laser.launch simulation:=true
```

Simulation:=true Use simulation, set to false to use physical robots, default parameter to false

```
process[stageros-2]: started with pid [7345]
ERROR: cannot launch node of type [gmapping/slam_gmapping]: gmapping
ROS path [0]=/opt/ros/melodic/share/ros
ROS path [1]=/home/bingda/catkin_ws/src
ROS path [2]=/opt/ros/melodic/share
```

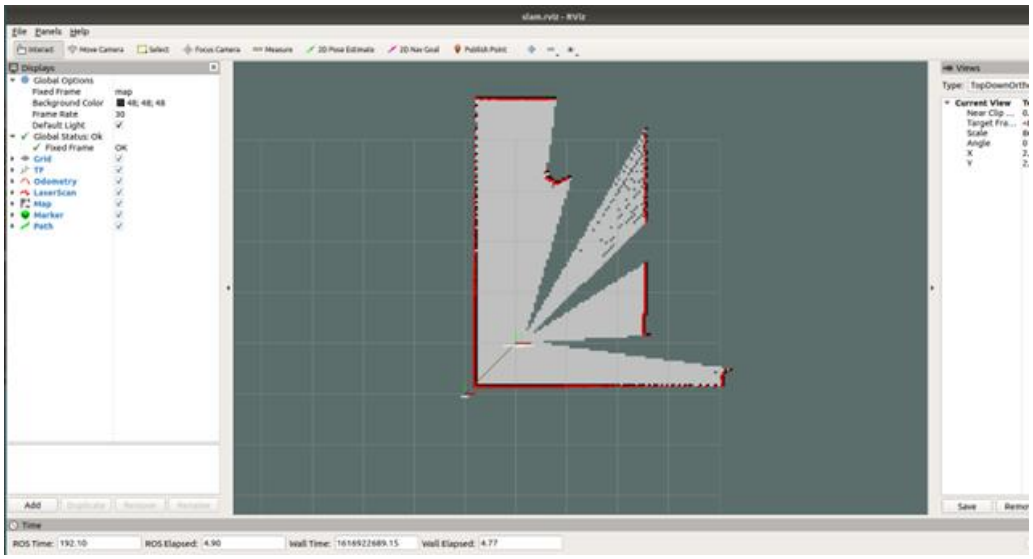
If the dependency relationship of the feature pack used was not resolved before, similar error prompts may appear after running, which is a common error we encounter. The reason is that the package is missing. Simply install it manually, and the installation format is sudo apt install ros-melodic-package name. For example:

```
sudo apt install ros-melodic-gmapping
```

Restart simulation mapping after installation is complete

```
roslaunch robot_navigation robot_slam_laser.launch simulation:=true
```

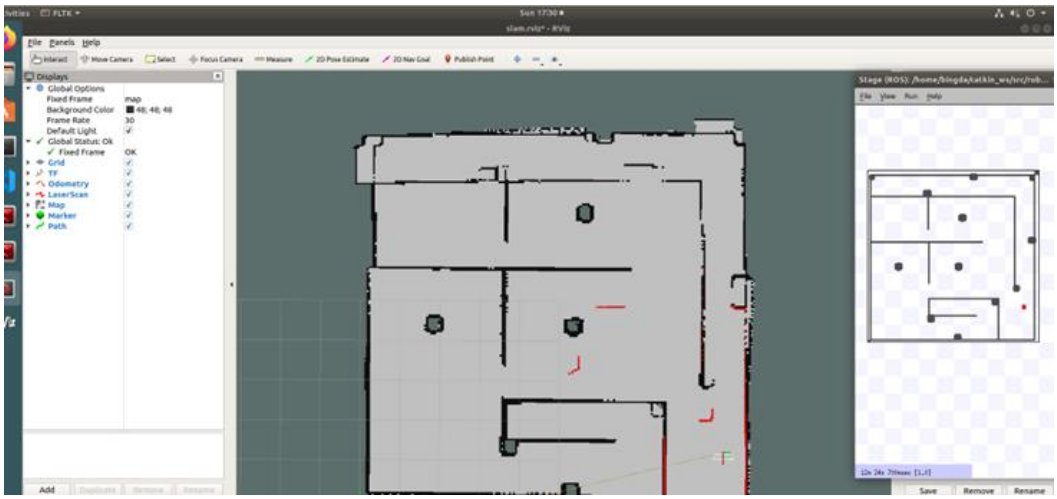
Start a rviz node,`roslaunch robot_navigation slam_rviz.launch`



At this point, start another keyboard remote control node

```
rosrun teleop_twist_keyboard teleop_twist_keyboard.py
```

After the startup is completed, the same method as in section 4.3.2 can be used to control the robot to traverse the environment through the keyboard to achieve mapping



Directly save the map in the current user's home directory and run `rosrun map_server map_saver -f stage_map`

When running simulation mapping, other parameters such as open_rviz being true can be passed in, which will also enable rviz

`roslaunch robot_navigation robot_slam_laser.launch simulation:=true open_rviz:=true`

A similar method is used to implement robot navigation in a simulation environment

`roslaunch robot_navigation robot_navigation.launch simulation:=true open_rviz:=true`

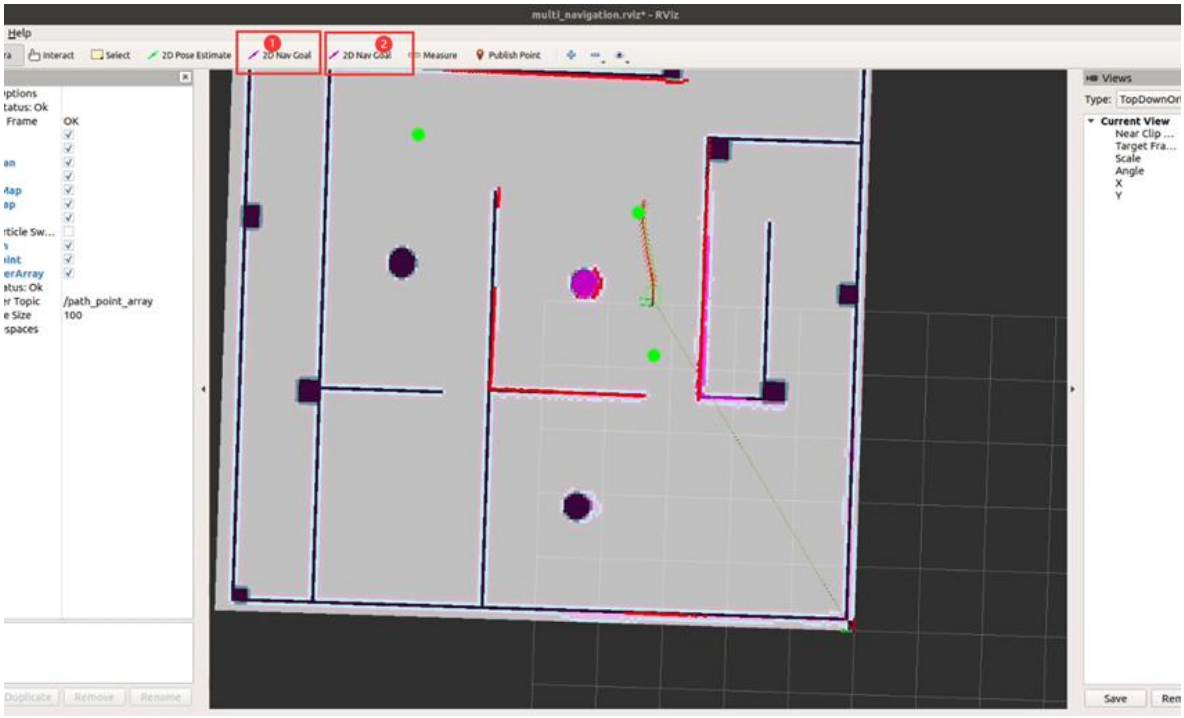
The operation after startup is exactly the same as the physical robot operation introduced in Chapter 4.3.3, except that in Chapter 3.5, the robot runs in a real environment, while in this chapter, the robot runs in a simulation environment

It should be noted that when conducting simulation experiments on a PC, the chassis type BASE-TYPE in the environment variable needs to be set to NanoCar. When setting the chassis type BASE-TYPE to NanoCar, the cmd_angle_instead_level parameter in the robot.navigation/param/NanoCar/teb_local_planner_params.yam configuration file needs to be changed from false to true. This is because the cmd_level topic accepted by the "Car" model in the stage simulator does not give the robot angular velocity, but refers to the steering angle of the steering structure. The simulation of other models does not need to be modified.

Robot multi target point navigation

In section 4.3.3, we have introduced the navigation of robots, which differs from multi-target point navigation in that. Ordinary navigation can only publish one target point. Before reaching the target point, if a new target point is published, the original target point will be discarded and the new target point will be directly directed. Multi target point navigation, on the other hand, follows the order in which the target points are published.

Still using stage simulation implementation here
roslaunch robot_navigation robot_navigation.launch simulation:=true If running on a physical robot, there is no need to pass in the simulation:=true parameter
Launch the multi-point navigation tool roslaunch robot_navigation multi_points_navigation.launch
Execute the rviz graphical monitoring program roslaunch robot_navigation multi_navigation.launch



As can be seen, compared to the rviz interface in the robot navigation chapter, there is an additional "2D Nav Goal" button. The topics triggered by these two buttons are also different. The left button has the same function as the navigation chapter. If a robot is given a target point, and a new target point is specified before the robot reaches the target point, the robot will abandon the previous target point and go to the new target point. The right button is the protagonist of this chapter, which can continuously assign multiple target points to the robot, and the robot will go to each target point in sequence.

Robot multi-point fully automatic cruise control

Launch navigation roslaunch robot_navigation robot_navigation.launch simulation:=true
If running on a physical robot, there is no need to pass in the simulation:=true parameter
Execute the rviz graphical monitoring program roslaunch robot_navigation navigation_rviz.launch
Then open a new terminal subscription/movebase_simple/goal topic on the PC end
rostopic echo /move_base_simple/goal

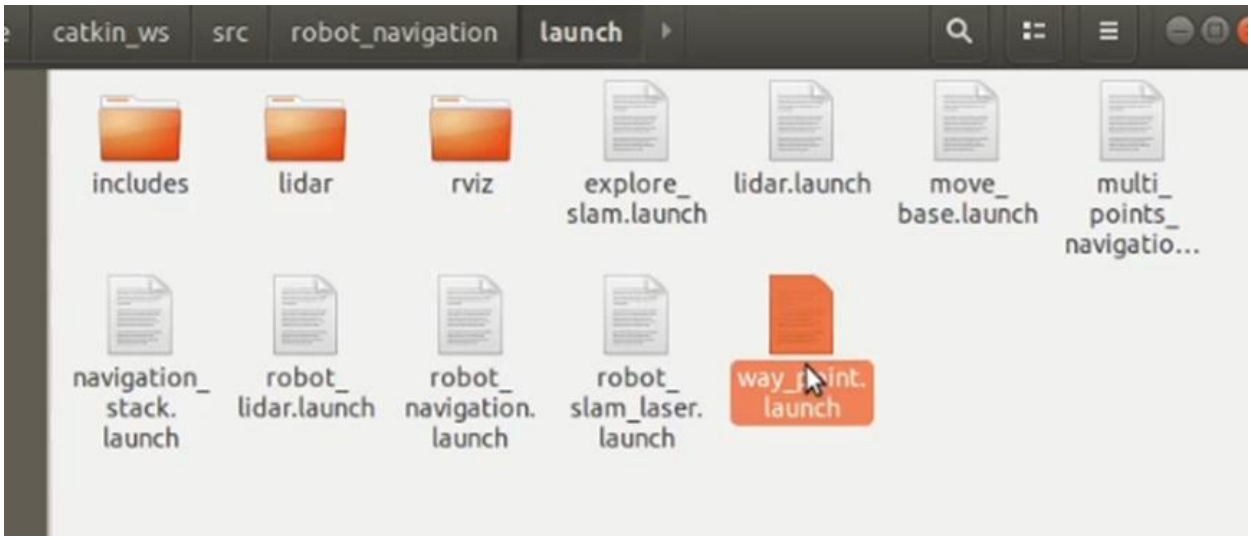
Then, in rviz, use the "**2D Nav Goal**" button to let the robot navigate to the first point you want to cruise. At this point, subscribing to the /move_base_simple/goal topic will output a topic

```
bingda@ubuntu:~$ rostopic echo /move_base_simple/goal
WARNING: no messages received and simulated time is active.
Is /clock being published?
header:
  seq: 0
  stamp:
    secs: 90
    nsecs: 700000000
  frame_id: "map"
pose:
  position:
    x: 2.02374196053
    y: 3.00534415245
    z: 0.0
  orientation:
    x: 0.0
    y: 0.0
    z: -0.00750593405335
    w: 0.99997183008
---
```

Record the three values at the red box

Then navigate to the second point you want to navigate to, and so on. Record all the points you want to navigate to

Open the way_point.launch file under the robot.navigation feature package to make modifications



Fill the values just recorded into this array one by one, and note that each column is a group. Here, I have set 3 target points (there is no upper limit to the number of cruise points, but at least 2 points are required). Save and exit after modification is completed

```
<launch>
    <!-- For Simulation -->
    <arg name="sim_mode" default="false" />
    <param name="/use_sim_time" value="$(arg sim_mode)"/>
    <arg name="loopTimes"       default="0" />
    <!-- move base -->
    <node pkg="robot_navigation" type="way_point.py" respawn="false" nam
output="screen">
        <!-- params for move_base -->
        <param name="goalListX" value="[2.0  4.0  1.0]" />
        <param name="goalListY" value="[3.0  3.0  2.0]" />
        <param name="goalListZ" value="[0.0  0.0  0.0]" />
        <param name="loopTimes" value="$(arg loopTimes)"/>
        <param name="map_frame" value="map" />
    </node>

</launch>
```

roslaunch robot_navigation way_point.launch loopTimes:=2

Here, our loopTimes parameter is the number of cruises, which defaults to 0. If set to 2, the robot will cruise back and forth at the cruise point twice before stopping. If set to 0, it will continue to cruise.

```
process[way_point-1]: started with pid [12942]
[INFO] [1614845781.393945]: Multi Goals Executing...
[INFO] [1614845782.409938]: Current Goal ID is: 0
[INFO] [1614845792.333478]: Current Goal ID is: 1
[INFO] [1614845803.333010]: Current Goal ID is: 2
[INFO] [1614845829.334205]: Current Goal ID is: 0
[INFO] [1614845839.833188]: Current Goal ID is: 1
[INFO] [1614845850.832701]: Current Goal ID is: 2
[INFO] [1614845876.832952]: Loop: 2 Times Finshed
```

The robot will output the target ID of the next cruising point every time it reaches a cruising point, and after the number of cruises, it will output Loop: 2 Times Finshed information.

4. Depth camera visual SLAM routine running

Test camera functionality

Step 1: Start the camera on the robot end

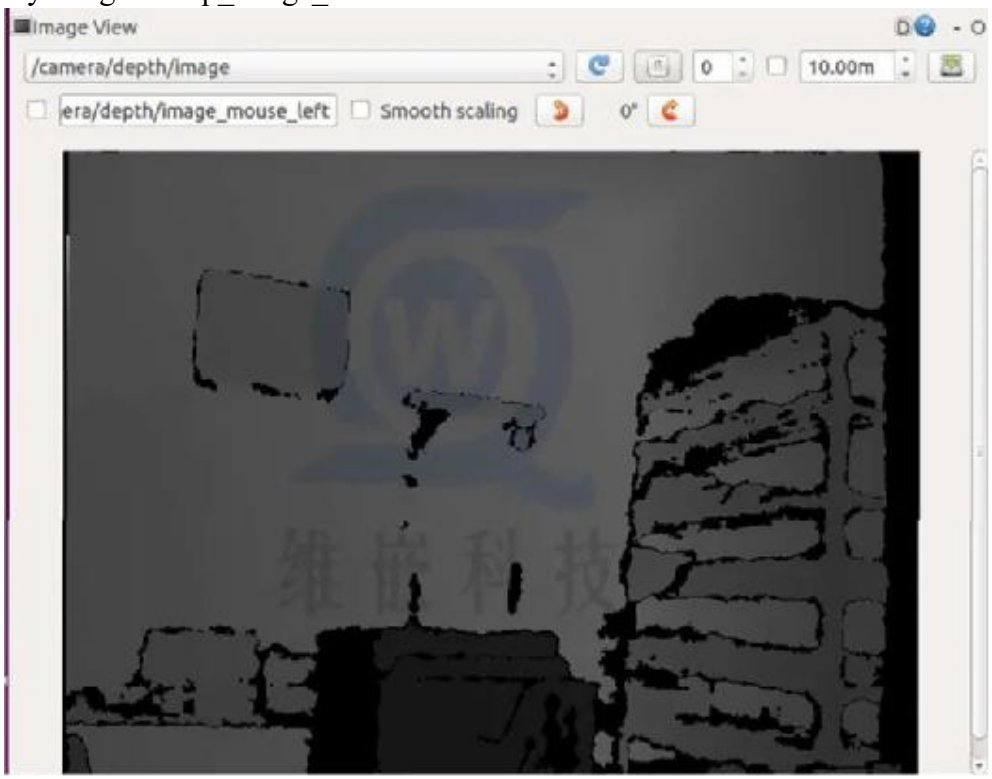
roslaunch robot_vslam camera.launch

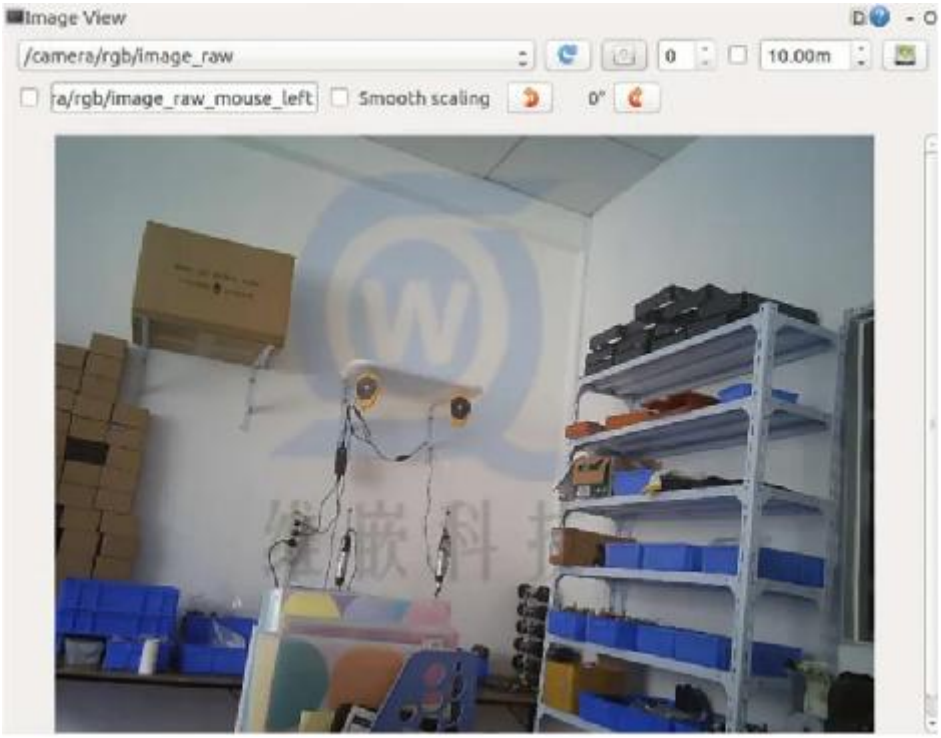
Step2: View images on

PC (needs to be executed on devices with displays, distributed communication between PC and robot has been configured normally)

rqt_image_view or rviz

can select the corresponding image topic in rqt_image_view or rviz to view related images. The following are some commonly used topics:
depth images /camera/depth/image
color images /camera/rgb/image_raw/compressed
infrared images /camera/ir/image (note: ir images can only be displayed through rviz, and in rqt_image_view, they will be all black, which is related to the image format and not a device issue)
Display images in rqt_image_view





Displaying images in Rviz

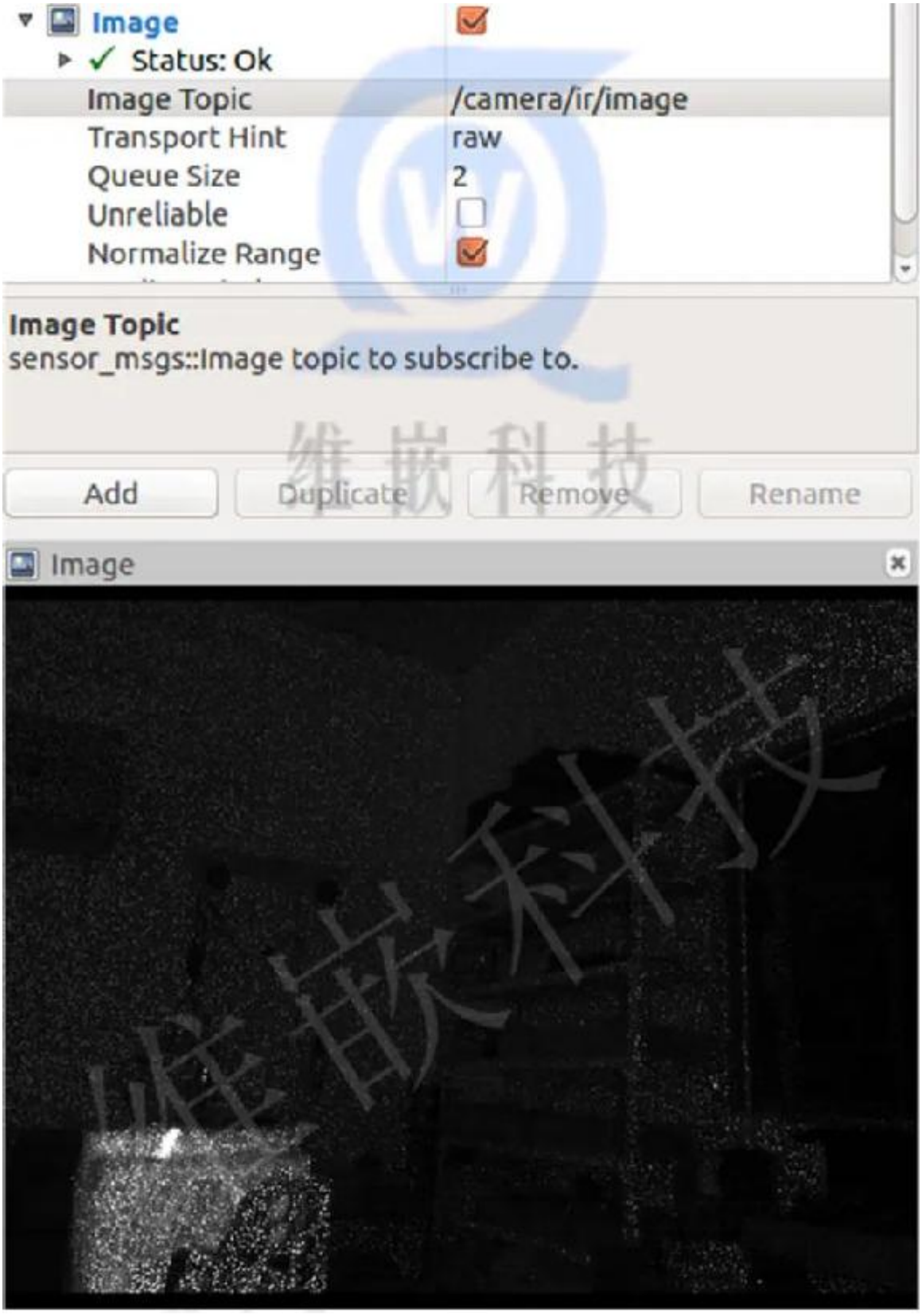


Image
Status: Ok
Image Topic
/camera/ir/image
Transport Hint
raw
Queue Size
2
Unreliable
Normalize Range
Image Topic
sensor_msgs::Image topic to subscribe to.
Add
Duplicate
Remove
Rename
Image

RGBD camera point cloud display

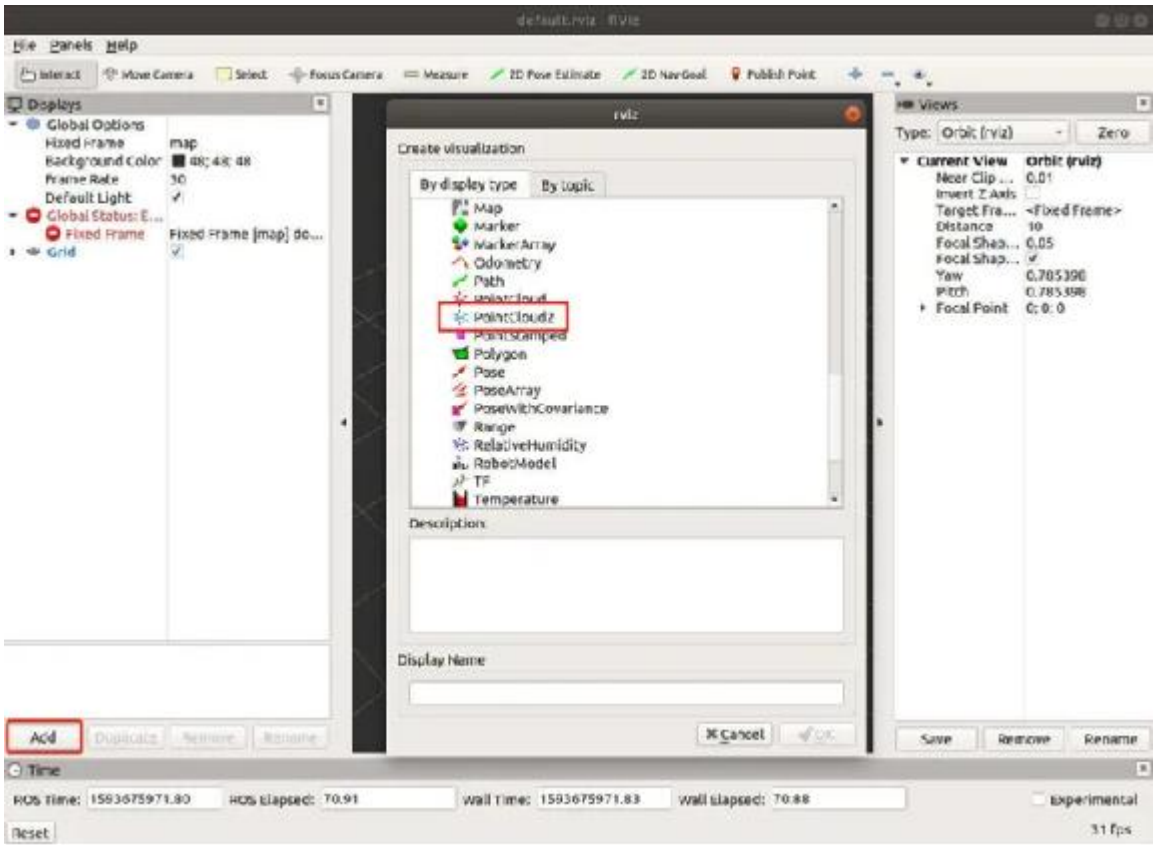
Step 1: Start the camera on the robot end

```
roslaunch robot_vslam camera.launch
```

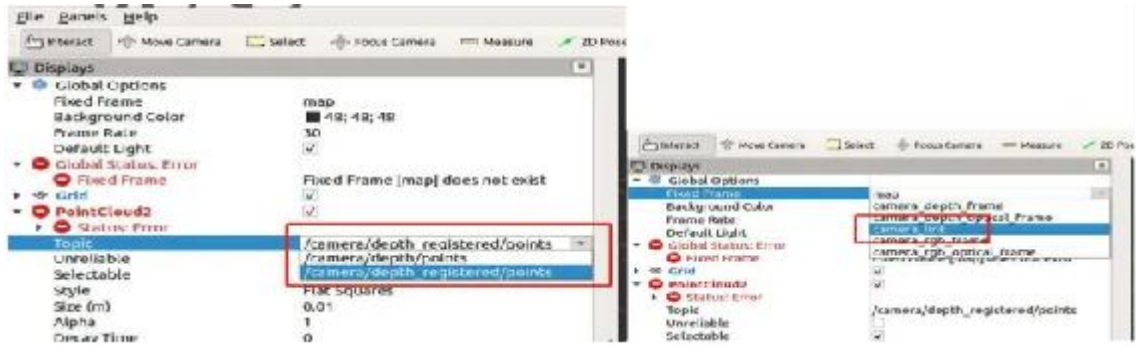
Step 2: Start rviz on PC to display point cloud

```
rviz
```

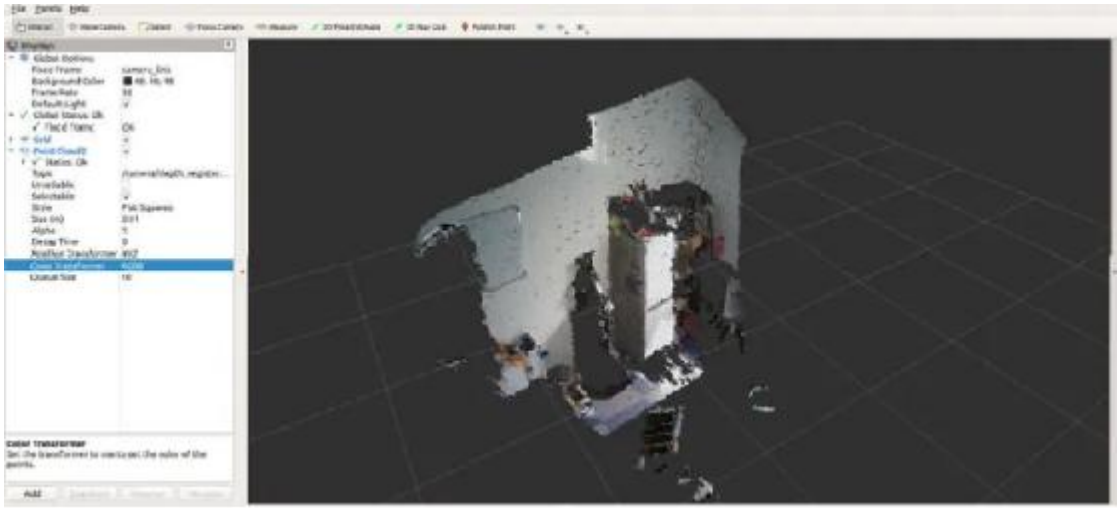
In order to help everyone understand the use of various plugins in rviz, we do not use pre configured configuration files. We will take everyone to build a rviz display environment hand in hand. First, we will add a point cloud display plugin, click "Add", and double-click "PointCloud2" in the pop-up interface



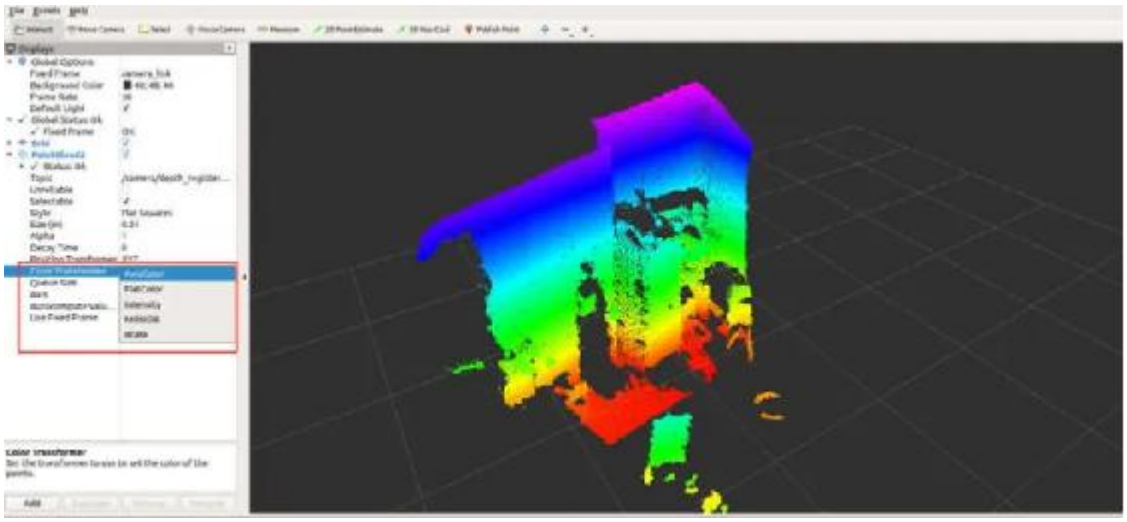
Expand the PointCloud2 selection card on the left, select Topic, and then select Fixed Frame as camera_link



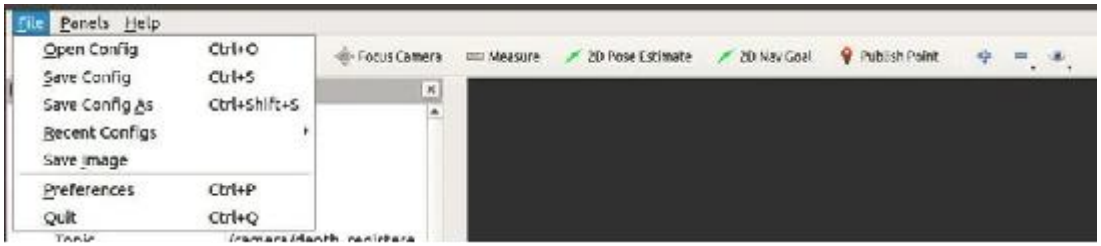
The point cloud map will be displayed in the back interface



Selecting different color modes in Color Transformer can color point clouds according to different rules



Finally, by using the "File" tab in the upper left corner Save Config As, you can save the current rviz configuration to a specified location. The next time you open rviz, you can directly open the configuration file through Open Config to avoid manually configuring the interface again.



Convert depth camera images to radar scanning data

The depth map of a depth camera can be converted into radar scanning data. Here, we use the depthimage_to_laserscan feature package to

first start the depth camera on the **robot side** (to achieve some tf transformations, we use the launch file in the robot_vslam package)

```
roslaunch robot_vslam camera.launch
```

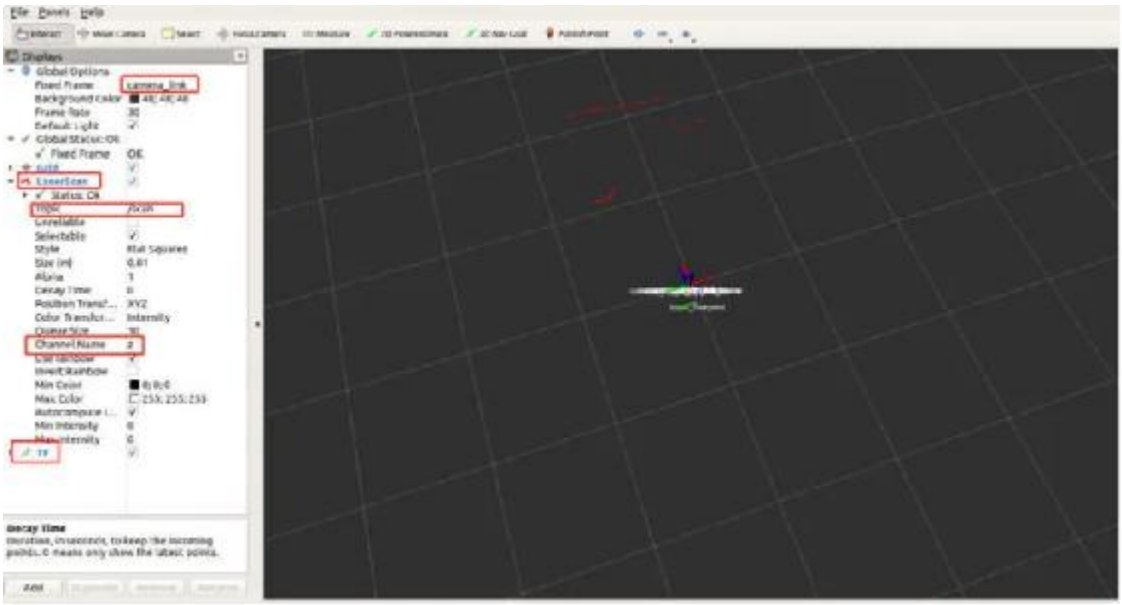
Then start the conversion from depth map to radar data on the **robot** or **PC end**

```
roslaunch robot_vslam depth_to_laser.launch
```

Open a rviz display interface on the PC

```
rviz
```

Configure the interface to display radar data normally



It can be noted that compared to the TF coordinate position of the robot, the radar data is not 360 degrees like the commonly used LiDAR data. This is because the field of view angle (FOV) of the depth camera (i.e., the horizontal FOV of our camera is 58.4 degrees, and the vertical FOV is 45.5 degrees) is limited. Therefore, the angle range of the converted radar data is consistent with the horizontal FOV of the camera. Simply put, it can only see things within a certain angle range in front of the camera.

Using a depth camera to achieve LiDAR mapping

We have already introduced the conversion of depth images into radar data. Here, we will try using a depth camera to achieve the LiDAR mapping function in the basic tutorial section.

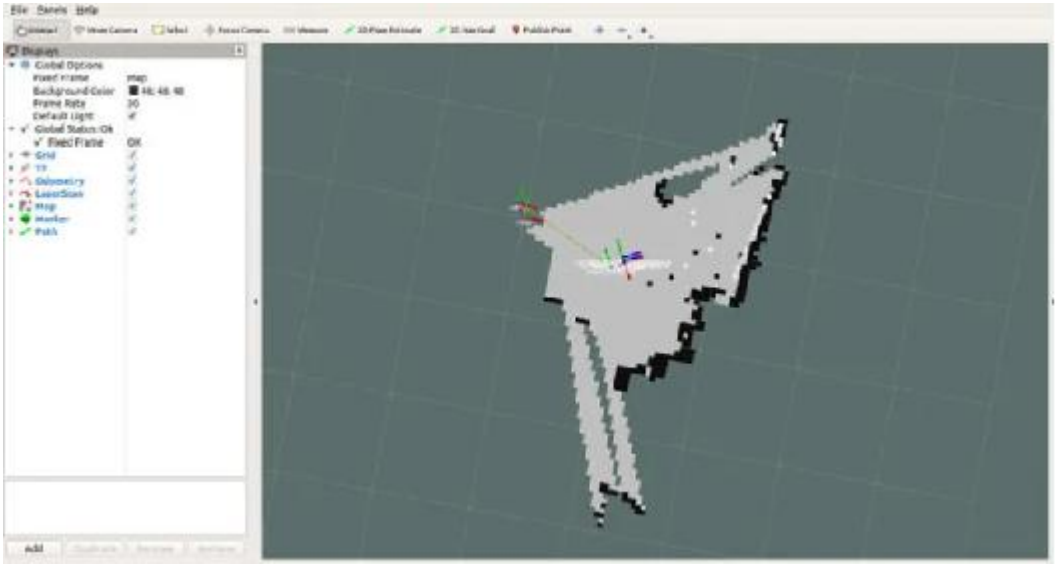
Robot end:

```
roslaunch base_control base_control.launch
roslaunch robot_vslam camera.launch
roslaunch robot_vslam depth_to_laser.launch
roslaunch robot_navigation gmapping.launch
```

PC end:

```
roslaunch robot_navigation slam_rviz.launch
rosrun teleop_twist_keyboard teleop_twist_keyboard.py
```

In the rviz interface, it can be seen that using only a depth camera can also achieve the effect of LiDAR mapping. The operation method and map saving method during the mapping process are the same as the LiDAR mapping method in the basic tutorial, so we will not repeat the introduction here.
However, according to our testing, using a depth camera to convert radar data for mapping is not very effective. This is mainly because the angle of the radar data converted through a depth camera is less than 60 degrees, and there are too few feature points that can be matched for the mapping algorithm, making it difficult to achieve good results. Here, we still use the parameters used for LiDAR mapping by default. Interested friends can make some adjustments to the parameters in the launch file of algorithms such as gmapping, which may improve the effect to a certain extent.



Using a depth camera to achieve LiDAR navigation

Robot end:

```
roslaunch base_control base_control.launch
roslaunch robot_vslam camera.launch
roslaunch robot_vslam depth_to_laser.launch
roslaunch robot_navigation navigation_stack.launch move_foard_only:=true
```

For models with Ackermann steering structure, it is necessary to pass in the planner parameters, namely:

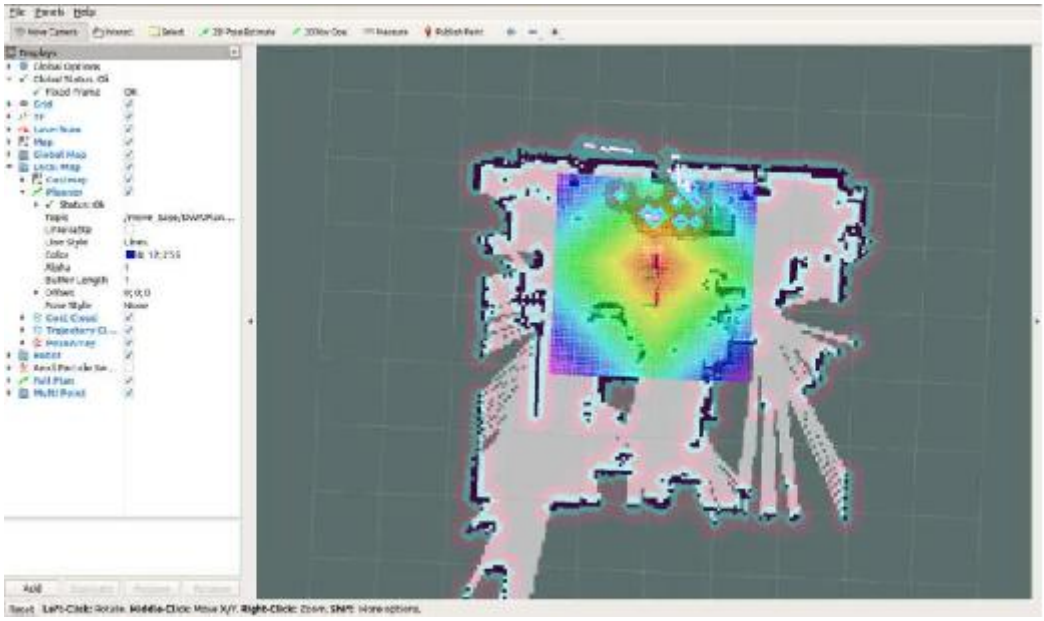
```
roslaunch robot_navigation navigation_stack.launch move_foard_only:=true
planner:=teb
```

PC end:

```
roslaunch robot_navigation navigation_rviz.launch
```

Please note that in the launcn file of the navigation stack, we have passed in the move_foard_only:=true parameter. This is because the depth camera only has a forward field of view, so there is a possibility of collision if the car moves backwards. Therefore, we only allow the car to move forward.
In the rviz interface, we can see that the interface is consistent with the LiDAR

navigation interface, and the operation method is also the same as the navigation in the basic tutorial section.
However, due to our limitations on the backward movement of the car and the limited angle of radar data, it is easy for the car to get stuck and unable to move due to the inability to plan the path.



Run visual SLAM mapping (rtabmap algorithm rgbd camera)

Open three terminals on the robot end and run them separately:

```
roslaunch base_control base_control.launch
roslaunch robot_vslam camera.launch
roslaunch robot_vslam rtabmap_rgbd.launch
```

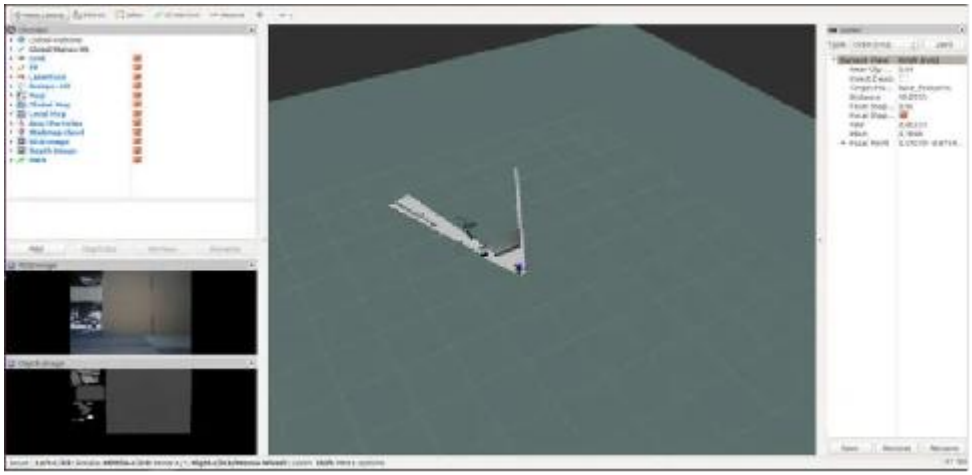
After the startup of rtabmap_rgbd.launch, the camera node and rtabmap node will respectively output the following information (because the output information here is more and valuable for reference, we did not write all the startup files as one launch file and output it in the same terminal)

```
 96506995 VERBOSE    [FPS] IR: 0.00 Depth: 0.00
[ INFO] [1590907767.496138251]: camera calibration URL: file:///home/nanorobot/catkin_ws/src/robot_vslam
/config/astrapro/depth_320.yaml
 97041083 VERBOSE    [FPS] IR: 0.00 Depth: 31.83
 98015290 VERBOSE    [FPS] IR: 0.00 Depth: 30.50
 99016732 VERBOSE    [FPS] IR: 0.00 Depth: 30.28
100045600 VERBOSE    [FPS] IR: 0.00 Depth: 29.89
101018839 VERBOSE    [FPS] IR: 0.00 Depth: 29.92
102019813 VERBOSE    [FPS] IR: 0.00 Depth: 30.16
103000399 VERBOSE    [FPS] IR: 0.00 Depth: 30.11
```

```
[ INFO] [1590907767.189637727]:
/rtabmap/rtabmap subscribed to (approx sync):
   /odom,
   /camera/rgb/image_rect_color,
   /camera/depth_registered/hw_registered/image_rect,
   /camera/rgb/camera_info,
   /scan
[ INFO] [1590907767.218522827]: rtabmap 0.19.3 started...
[ INFO] [1590907767.632839668]: rtabmap (1): Rate=1.00s, Limit=0.000s, RTAB-Map=0.0748s, Maps update=0.0
085s pub=0.0002s [local map=1, WM=1]
[ INFO] [1590907768.628034644]: rtabmap (2): Rate=1.00s, Limit=0.000s, RTAB-Map=0.0528s, Maps update=0.0
001s pub=0.0000s [local map=1, WM=1]
```

Open the rviz terminal on the PC end

roslaunch robot_vslam rtabmap_rviz.launch allows you to view the established flat and three-dimensional maps in the rviz terminal



We can also use the graphical interface that comes with rtabmap, rtabmapviz, to observe the mapping process on the **PC side**

roslaunch robot_vslam rtabmapviz.launch

It should be noted that rtabmapviz occupies a huge bandwidth (over 100Mb). If the local area network conditions are poor, it is not recommended to use it. If necessary, use it and turn it off in a timely manner to ensure smooth network connectivity.

Finally, start a keyboard control node on the PC or robot end to remotely control the robot to move around and establish a map

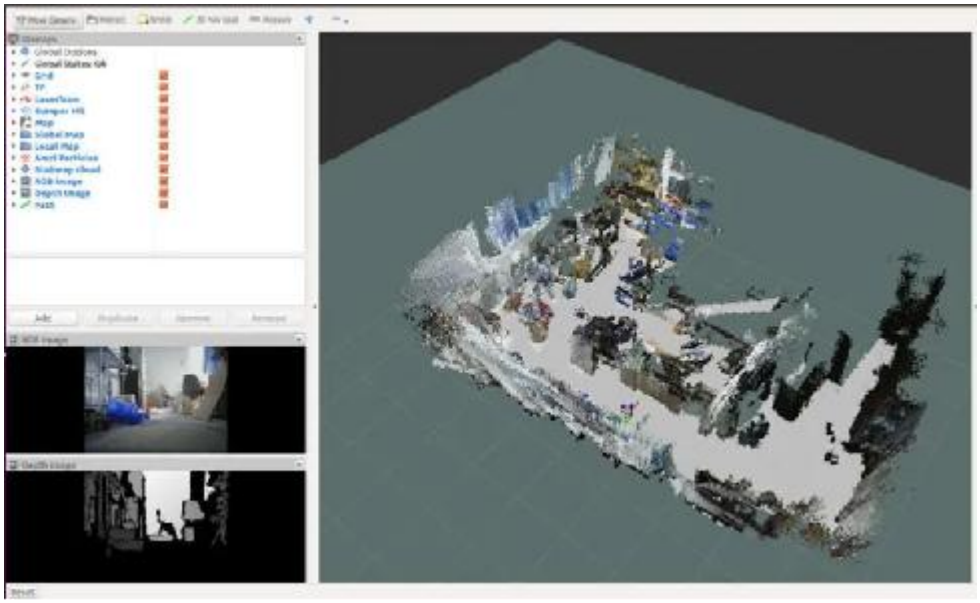
rosrun teleop_twist_keyboard teleop_twist_keyboard.py

Our suggestion is to ensure that the robot's linear motion speed does not exceed 0.2m/s and rotation speed does not exceed 0.4rad/s to ensure the quality of mapping. The

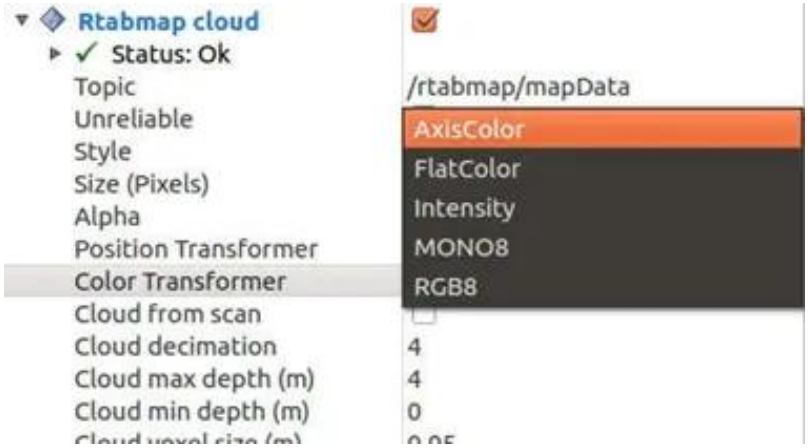
mapping process is basically consistent with that of LiDAR mapping, and the remote control robot completes the mapping by traversing the environment once. The map here is saved by default in the robot side /home/nanorobot/.ros/ rtabmap.db file. As this file contains information about images and feature points during the mapping process, the file will be relatively large.

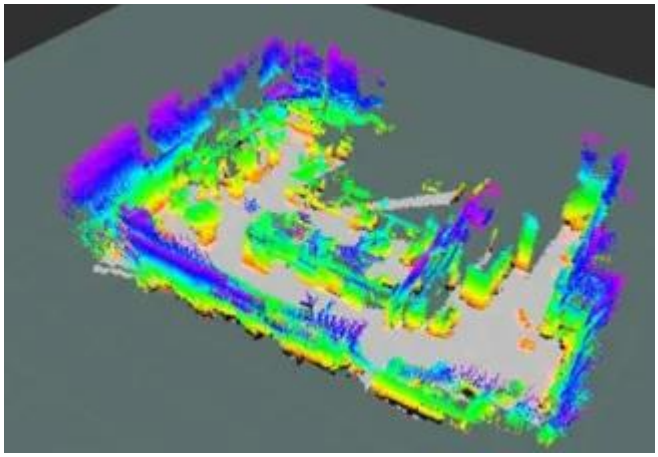
Note: Starting the roslaunch robot_vslam rtabmap_rgbd.launch will overwrite the rtabmap.db file. If you need to keep the results of the last mapping, be sure to backup the file.

After completing the mapping, the map in rviz is shown below



Here we can also modify the color of the point cloud display to achieve other effects, such as





After completing the mapping, close the open nodes by pressing Ctrl+c. In the next chapter, we will use the map created here to achieve robot navigation

Run visual SLAM navigation (rtabmap algorithm rgbd camera)

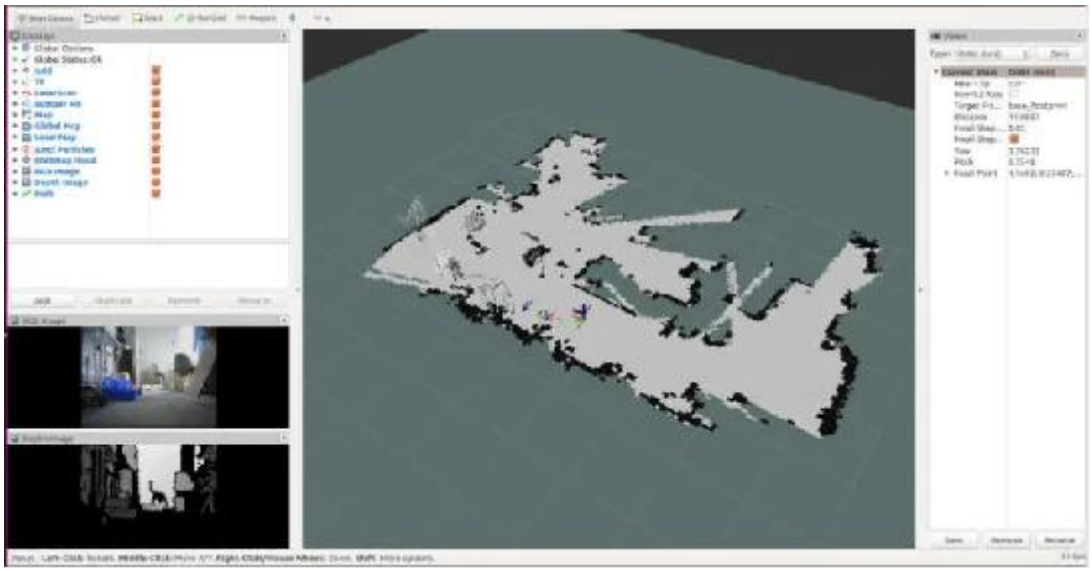
Open four terminals on the robot end and run them separately:

```
roslaunch base_control base_control.launch
roslaunch robot_vslam camera.launch
roslaunch robot_vslam rtabmap_rgbd.launch localization:=true
roslaunch robot_vslam move_base.launch planner:=dwa move_forward_only:=true
```

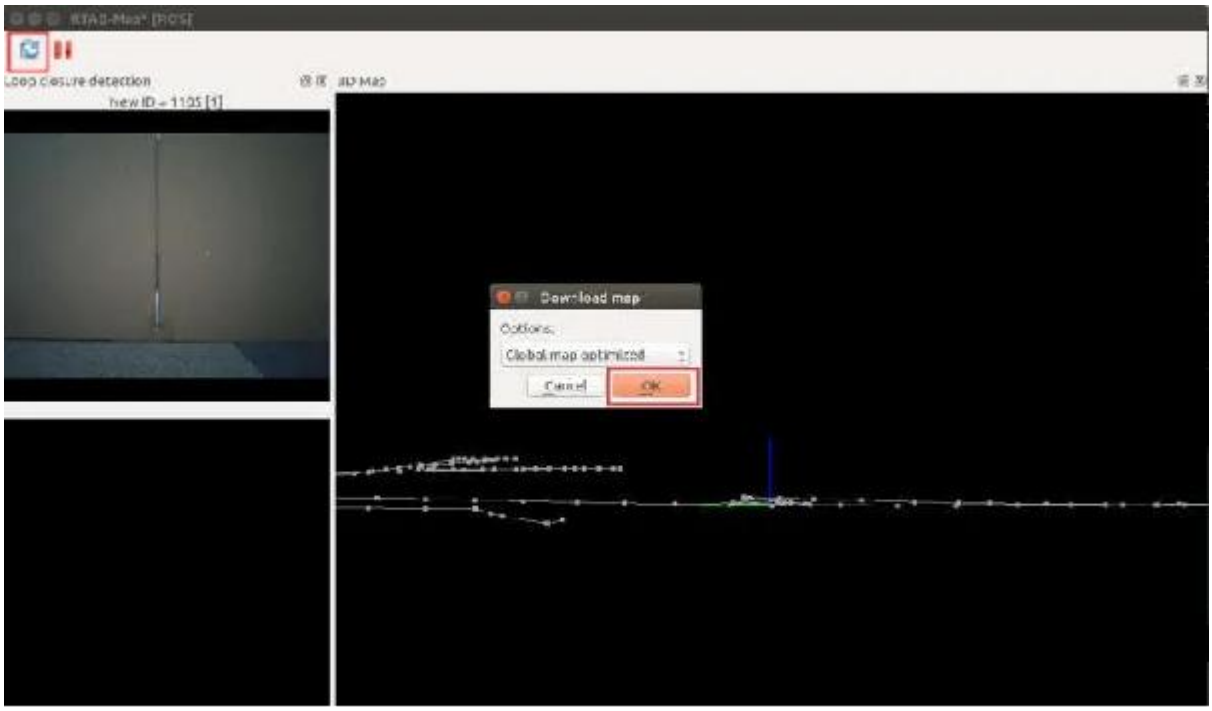
The planner parameters for movebase can be selected from dwa and teb, which have been introduced in the LiDAR navigation section. The moveforward_only parameter is designed to only allow the robot to move forward (the visual sensor only has forward perception).

After starting, check for errors at each node. If there are no errors, navigation and positioning operations can be started. On the PC side, open the rviz terminal and the rtabmapviz visualization tool

```
roslaunch robot_vslam rtabmap_rviz.launch
```



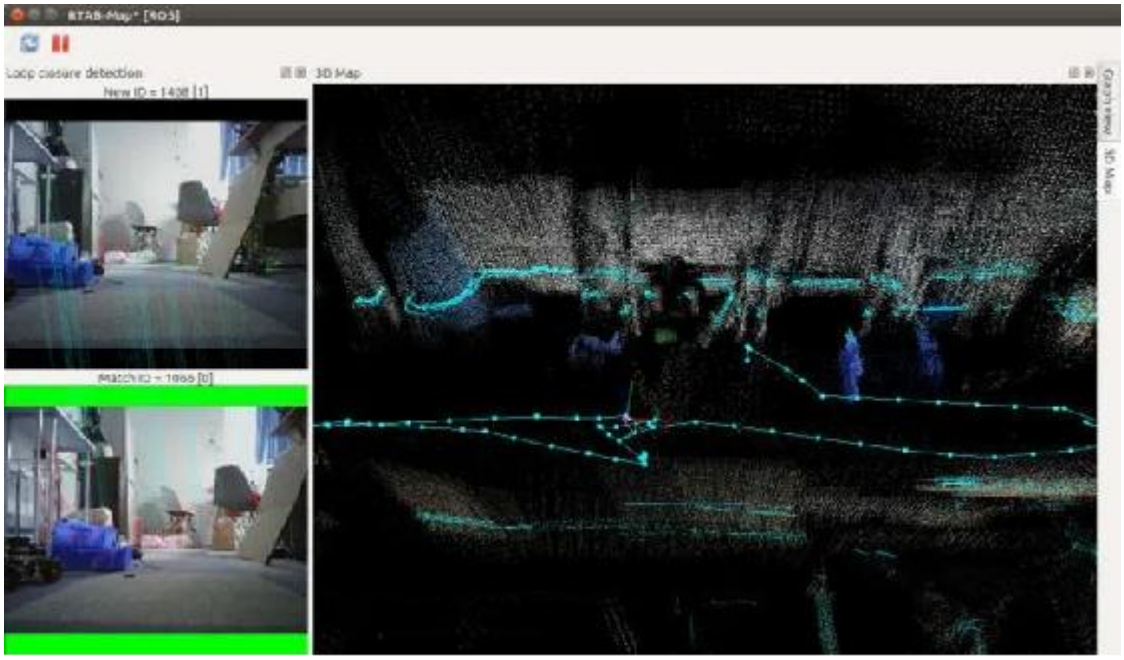
```
roslaunch robot_vslam rtabmapviz.launch
```



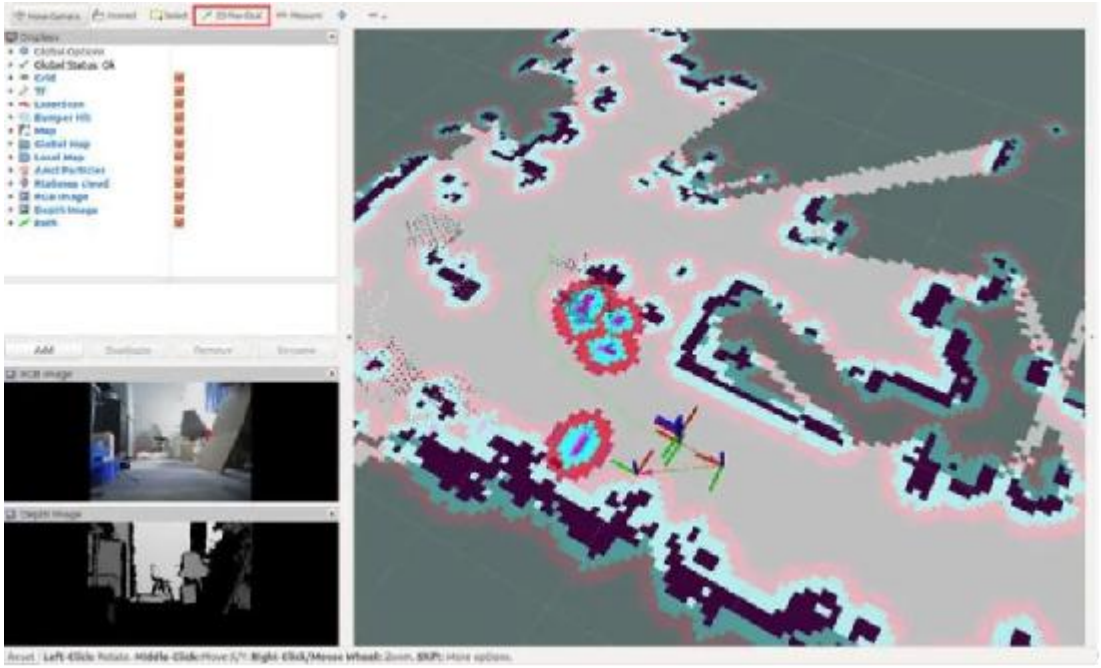
Clicking on the button in the red box in rtabmapviz to synchronize map database information may take some time, depending on your local area network conditions. If no image appears in the bottom left position and both sides of the image are black, the robot has not yet completed the initial position determination. If shown in the following figure, the robot has completed positioning and can start navigation without the need to perform the following steps (usually, the robot can directly complete positioning by starting from a position with rich textures and has already reached the last time. The purpose of this document is to illustrate that the initial positioning process is intentionally started from an area with almost no texture, making it impossible for the robot to complete the initial positioning). Start a keyboard control node
on the PC or robot end:

```
rosrun teleop_twist_keyboard teleop_twist_keyboard.py
```

Slowly move the robot until an image appears in the bottom left corner and both sides are green



Next, you can specify the target point in the map for the robot to navigate through.

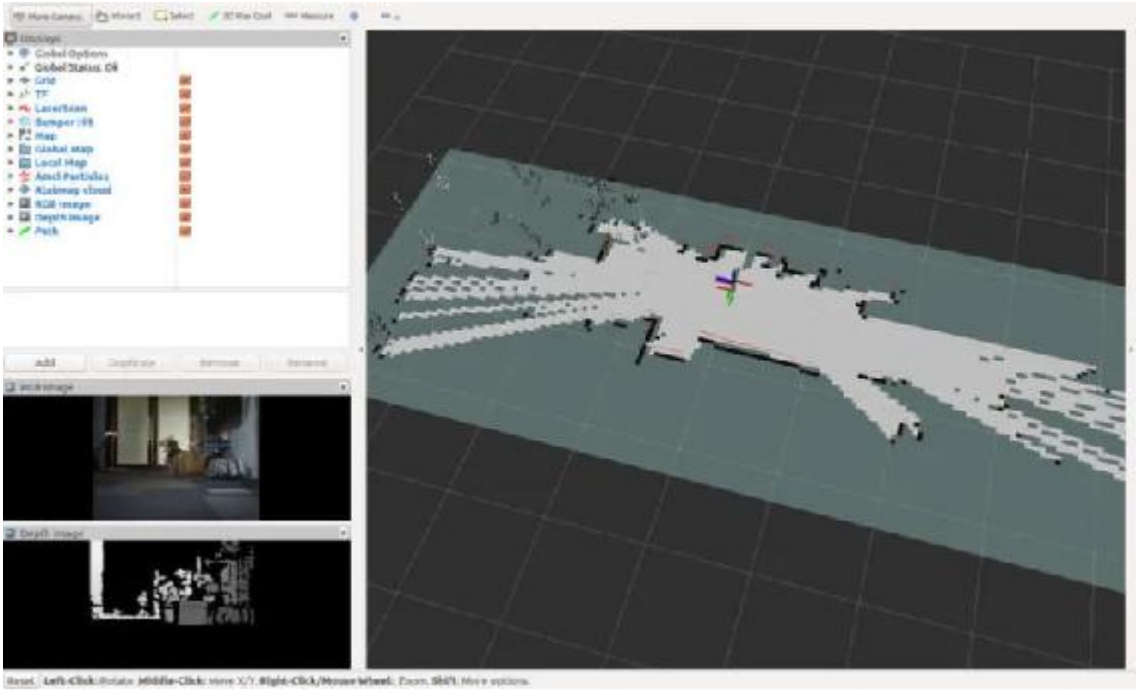


Attention: The field of view angle of the rgbd camera is small, and there is a large blind spot in depth information, which may cause collisions during navigation. This can be improved by adjusting and increasing the inflation_radius parameter in robot_vslam/param/NanoRobot/costmap_common_params.yaml , but it is difficult to achieve the same effect as lidar navigation or rgbd camera+lidar.

Run visual SLAM mapping (RGB camera+LiDAR)

The operation process here is completely consistent with Chapter 3, except for some differences in the startup instructions. Therefore, only the startup instructions are

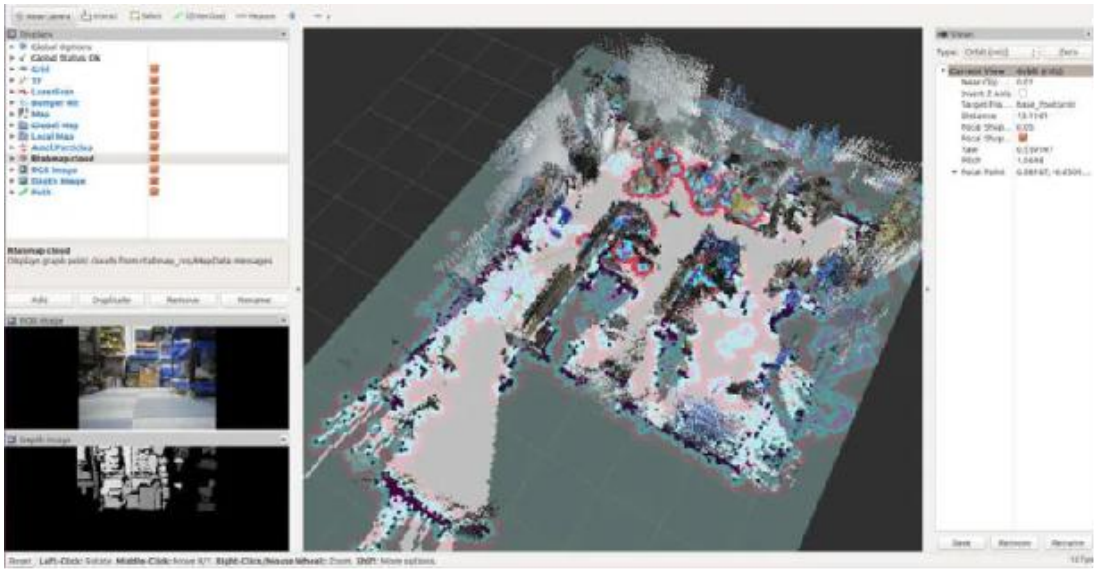
listed here, and the operation process is no longer described.
Open three terminals on the robot end and run them separately:
roslaunch robot_navigation robot_lidar.launch
roslaunch robot_vslam camera.launch
roslaunch robot_vslam rtabmap_rgbd_lidar.launch
Open the rviz terminal on the PC side
roslaunch robot_vslam rtabmap_rviz.launch
Start a keyboard control node on the PC or robot end to remotely control the robot to move around and establish a map
rosrun teleop_twist_keyboard teleop_twist_keyboard.py
Our suggestion is to ensure that the robot's linear motion speed does not exceed 0.2m/s and rotation speed does not exceed 0.4rad/s to ensure the quality of mapping. The
mapping process is basically consistent with that of LiDAR mapping, and the remote control robot completes the mapping by traversing the environment once. It can be seen here that due to the addition of 360 degree LiDAR data, the mapping performs better in 2D mapping.



Run visual SLAM navigation (RGB camera+LiDAR)

Open four terminals on the robot end and run them separately:
roslaunch robot_navigation robot_lidar.launch
roslaunch robot_vslam camera.launch
roslaunch robot_vslam rtabmap_rgbd_lidar.launch localization:=true
roslaunch robot_vslam move_base.launch planner:=dwa move_forward_only:=false
Due to the addition of a 360 degree LiDAR, the robot has omnidirectional perception ability. Therefore, the moveforward-only parameter can be set to false, allowing the

robot to move backwards on the
PC end and open the rviz terminal and rtabmapviz visualization tool
roslaunch robot_vslam rtabmap_rviz.launch
roslaunch robot_vslam rtabmapviz.launch



Due to the addition of LiDAR, robots will have significant improvements in obstacle avoidance at close range. Related reference materials:
http://wiki.ros.org/rtabmap_ros/Tutorials