Object Tracking

6.1、Program Function Description

After the program is started. Use the mouse to select the object to be tracked, press the [Space] key, and the car will enter the tracking mode. The car will keep a distance of 1 meter from the tracked object, and always ensure that the tracked object remains in the center of the screen.

In addition, the [L1] key of the handle can lock/open the motion control of the car. When the motion control is turned on, the function will be locked; when the motion control is locked, the function can be turned on.

6.2、Code Reference Path

After SSH connects to the car, the location of the function source code is located at,

```
#launch file
/home/sunrise/yahboomcar_ws/src/yahboomcar_kcftracker/launch/kcftracker_launch.xml
```

The node or launch file to be started is explained as follows,

- KCF_Tracker_Node: mainly completes image processing and calculates the center coordinates of the tracked object. The code path is,

```
/home/sunrise/yahboomcar_ws/src/yahboomcar_kcftracker/src/KCF_Tracker_main.cpp
```

- astro_pro_plus.launch.xml: Start Astra camera
- yahboomcar_bringup_launch.py: Start chassis

6.3, Program startup

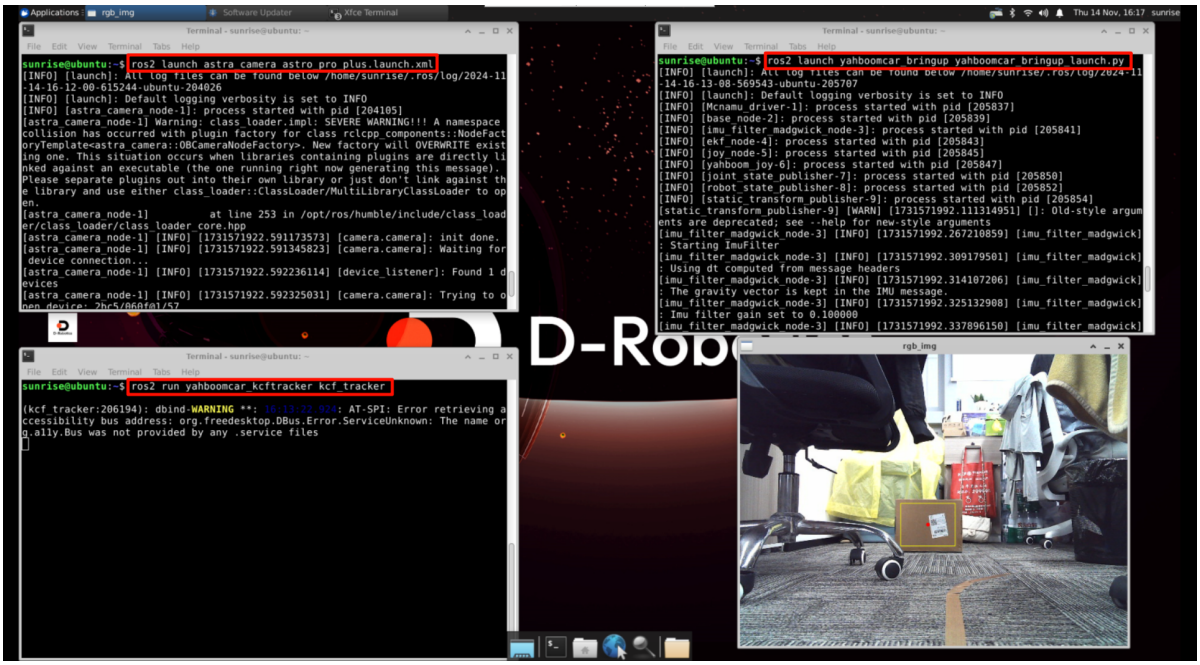
6.3.1, Program startup and web page display

After VNC connects to the car, enter in the terminal,

```
#Start the depth camera (choose one according to the model)
#astraproplus camera
ros2 launch astra_camera astro_pro_plus.launch.xml

#Start chassis data reception
ros2 launch yahboomcar_bringup yahboomcar_bringup_launch.py

#Start object tracking node
ros2 run yahboomcar_kcftracker kcf_tracker
```



Use the mouse to frame the object to be tracked, and release it to select the target. Then press the [Spacebar] to start tracking, and move the object slowly. The car will follow and keep a distance of 1 meter from the object.

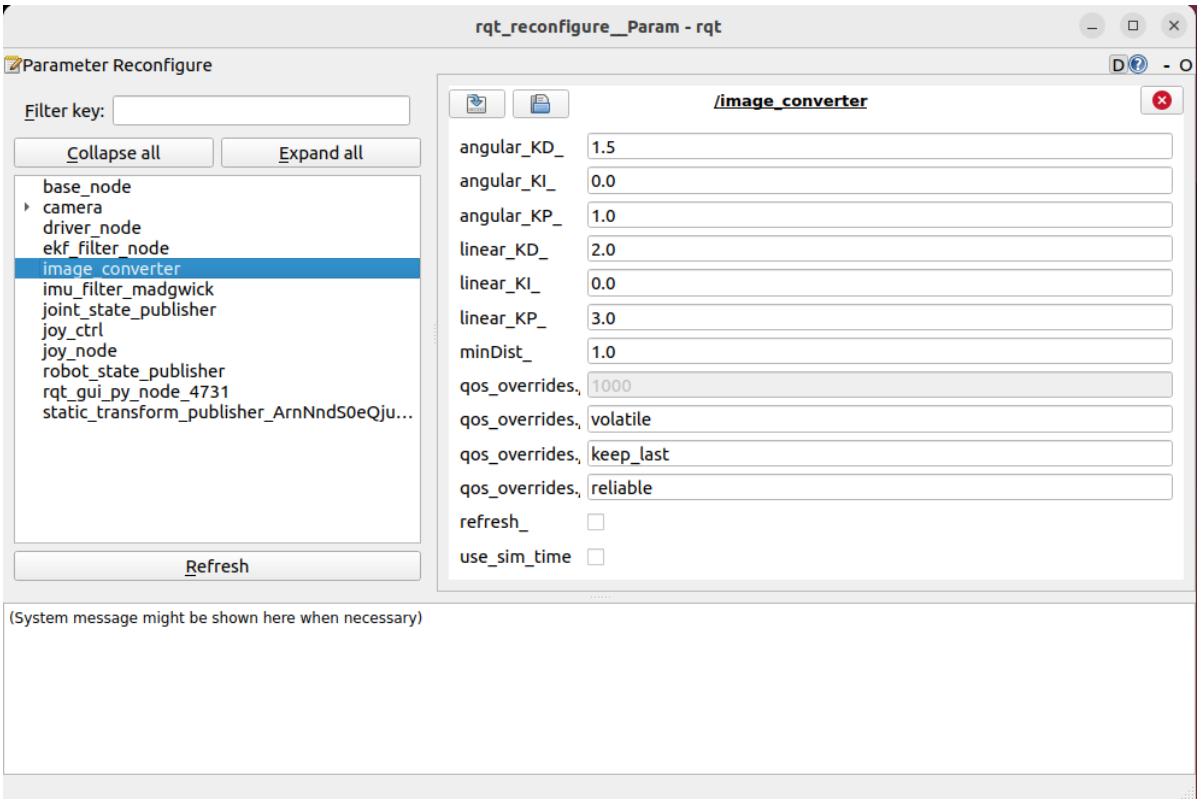
Press the [q] key to stop tracking and exit the program; press the [r] keyboard to clear the selected target and reselect it;

Move the object slowly, and the car will follow and keep a distance of 1 meter from the object.

6.3.2, Dynamic parameter adjustment

You can also use the dynamic parameter adjuster to debug the parameters, open a terminal input, virtual machine terminal input,

```
ros2 run rqt_reconfigure rqt_reconfigure
```



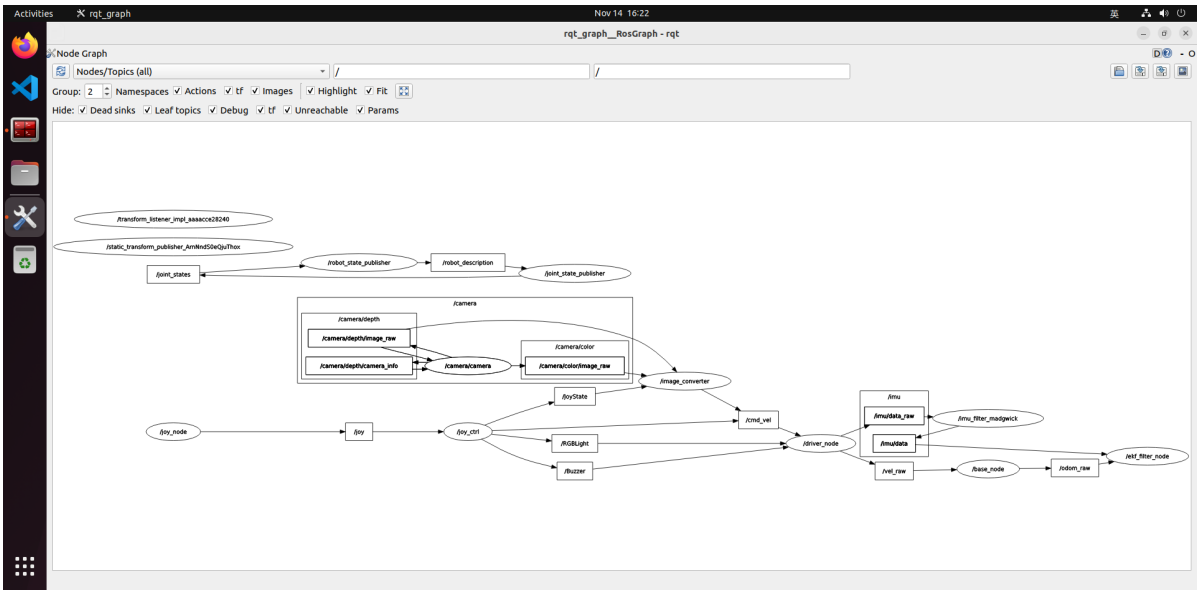
The adjustable parameters include the car's linear speed, angular velocity PID and tracking distance.

After modifying the parameters, press the Enter key or click the blank space in the GUI to write the parameter value.

6.3.3, View the node topic communication graph

Open the virtual machine terminal and use the following command to view the topic communication between nodes,

```
ros2 run rqt_graph rqt_graph
```



6.4, Core source code analysis

The principle of function implementation is similar to color tracking. It calculates the linear velocity and angular velocity based on the center coordinates of the target and the depth information fed by the depth camera, and then publishes it to the chassis. Some codes in KCF_Tracker.cpp are as follows,

```
//After selecting the object, the center coordinates are obtained for calculating the angular velocity
if (bBeginKCF) {
result = tracker.update(rgbimage);
rectangle(rgbimage, result, Scalar(0, 255, 255), 1, 8);
circle(rgbimage, Point(result.x + result.width / 2, result.y + result.height / 2), 3, Scalar(0, 0, 255),-1); } else rectangle(rgbimage, selectRect, Scalar(255, 0, 0), 2, 8, 0); //Calculate the values of center_x and distance for calculating speed int center_x = (int)(result.x + result.width / 2); int num_depth_points = 5; for (int i = 0; i < 5; i++) { if (dist_val[i] > 40 && dist_val[i] < 8000) distance += dist_val[i]; else num_depth_points--; } if (num_depth_points == 0) distance = this->minDist; else distance /= num_depth_points;

//Calculate linear speed and angular speed
linear_speed = -linear_PID->compute(this->minDist, distance);
rotation_speed = angular_PID->compute(320, center_x) / 1000.0;
```