4. KCF object tracking

1. Program function description

After the program starts, select the object to be tracked by the mouse and press the space bar, the cart enters the tracking mode. The cart will keep a distance of 1 meter from the object to be tracked, and always make sure the object to be tracked stays in the center of the screen. Press R to enter selection mode and Q to exit the program. After the joystick program is started, you can also pause/resume tracking by using the R2 key on the joystick.

2. Code reference path

After entering the docker container, the location of the source code for this feature is located at.

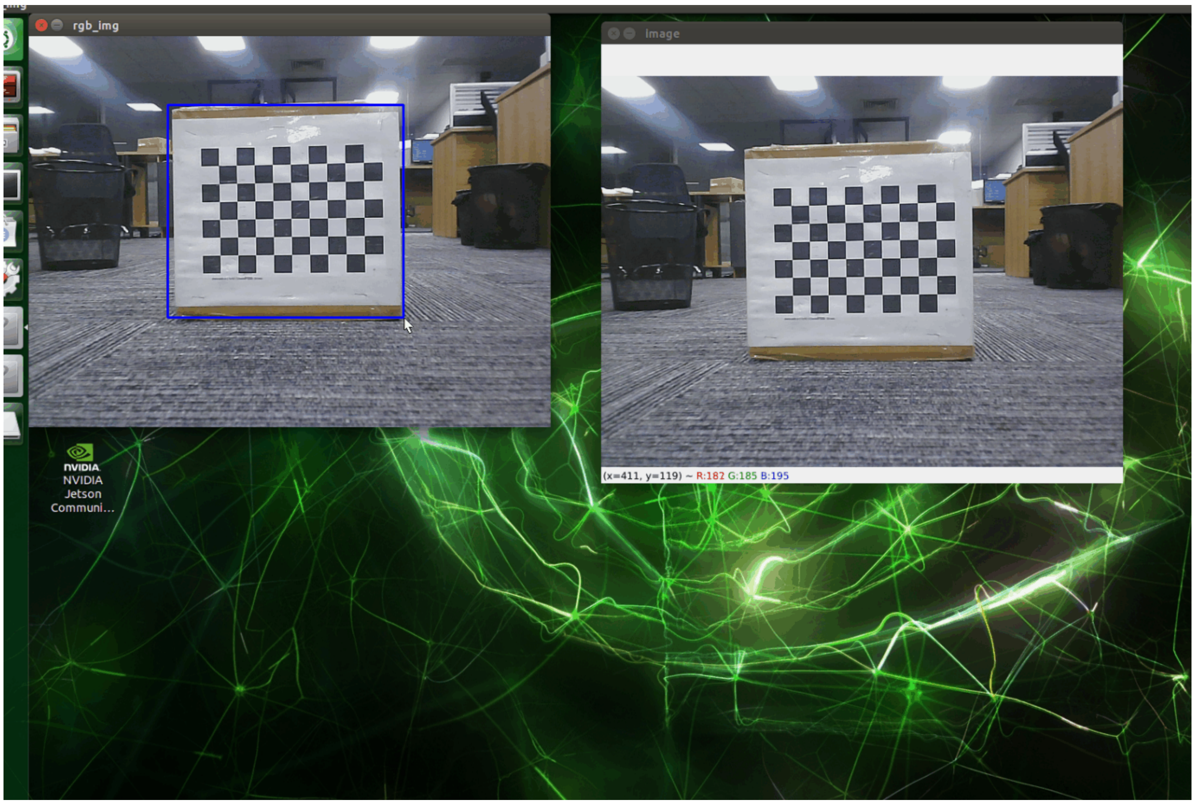
```
/root/yahboomcar_ros2_ws/yahboomcar_ws/src/yahboomcar_KCFTracker/src/KCF_Tracker
.cpp
```

3. Program startup

3.1 Startup Commands

After entering the docker container, depending on the actual model, terminal input, the

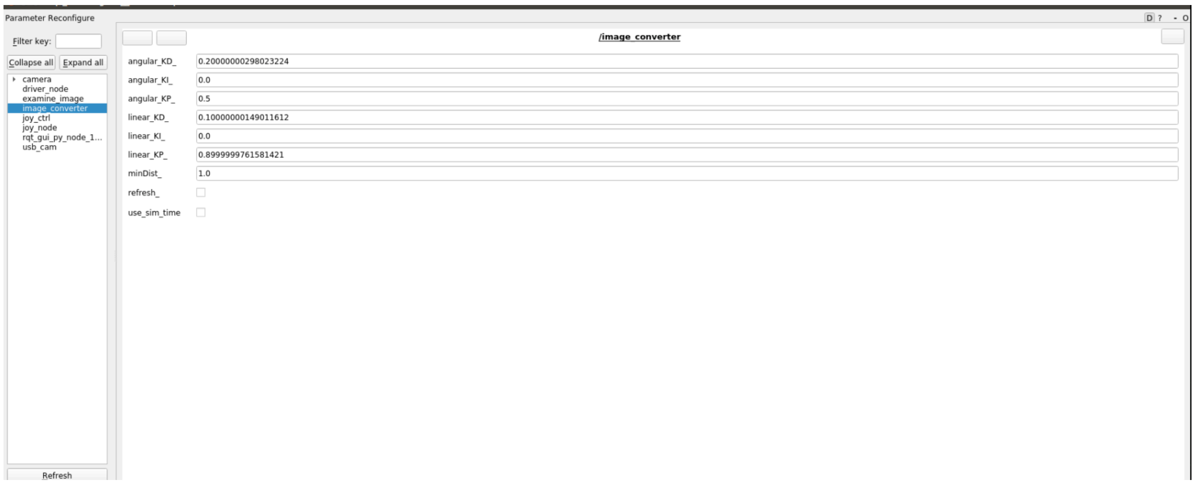
```
#启动小车底盘 #Start the trolley chassis
ros2 run yahboomcar_bringup Ackman_driver_R2
#启动深度相机,以astraproplus为例
#Start the depth camera, for example astraproplus
ros2 launch astra_camera astro_pro_plus.launch.xml
#启动手柄节点 #Start the handle node
ros2 run yahboomcar_ctrl yahboom_joy_R2
ros2 run joy joy_node
#启动KCF追踪程序 #Start the KCF tracking program
ros2 run yahboomcar_KCFTracker KCF_Tracker_Node
```



Use the mouse to frame the object to be tracked and release it to select the target. Then press spacebar to start tracking, move the object slowly, the cart will follow and keep 1m distance from the object.

You can also use the Dynamic Parameter Adjuster to debug the parameters by typing into the Docker terminal.

```
ros2 run rqt_reconfigure rqt_reconfigure
```



/image_converter

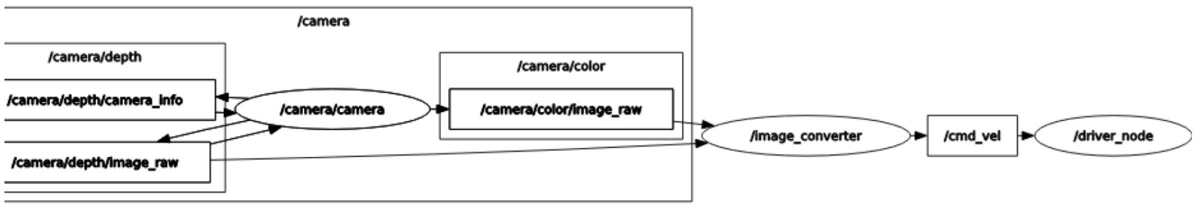
angular_KD_	0.20000000298023224
angular_KI_	0.0
angular_KP_	0.5
linear_KD_	0.10000000149011612
linear_KI_	0.0
linear_KP_	0.8999999761581421
minDist_	1.0
refresh_	☐
use_sim_time	☐

The adjustable parameters are the PID of linear velocity and angular velocity of the trolley and the distance of tracking, after modifying the parameters, click "refresh" to refresh the data.

3.2 Viewing Node Topic Communication Map

The following commands can be used to view the topic communication between nodes.

```
ros2 run rqt_graph rqt_graph
```



4. Core code

The principle of function implementation is similar to color tracking, both are based on the center coordinates of the target and the depth information fed by the depth camera to calculate the linear velocity and angular velocity, and then released to the chassis, part of the code into the following.

```
//这部分是选择物体后，得出中心坐标，用于计算角速度.
// This is the part where the object is selected and the center coordinates are
derived, which are used to calculate 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);

//这部分是计算出center_x，distance的值，用于计算速度
//This part is to calculate the values of center_x, distance for calculating
velocity
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] > 0.4 && dist_val[i] < 10.0) distance += dist_val[i];
    else num_depth_points--;
}
distance /= num_depth_points;

//计算线速度和角速度
//Calculate linear and angular velocities
if (num_depth_points != 0) {
    std::cout<<"minDist: "<<minDist<<std::endl;
    if (abs(distance - this->minDist) < 0.1) linear_speed = 0;
    else linear_speed = -linear_PID->compute(this->minDist, distance);//-
linear_PID->compute(minDist, distance)
}
rotation_speed = angular_PID->compute(320 / 100.0, center_x /
100.0);//angular_PID->compute(320 / 100.0, center_x / 100.0)
```