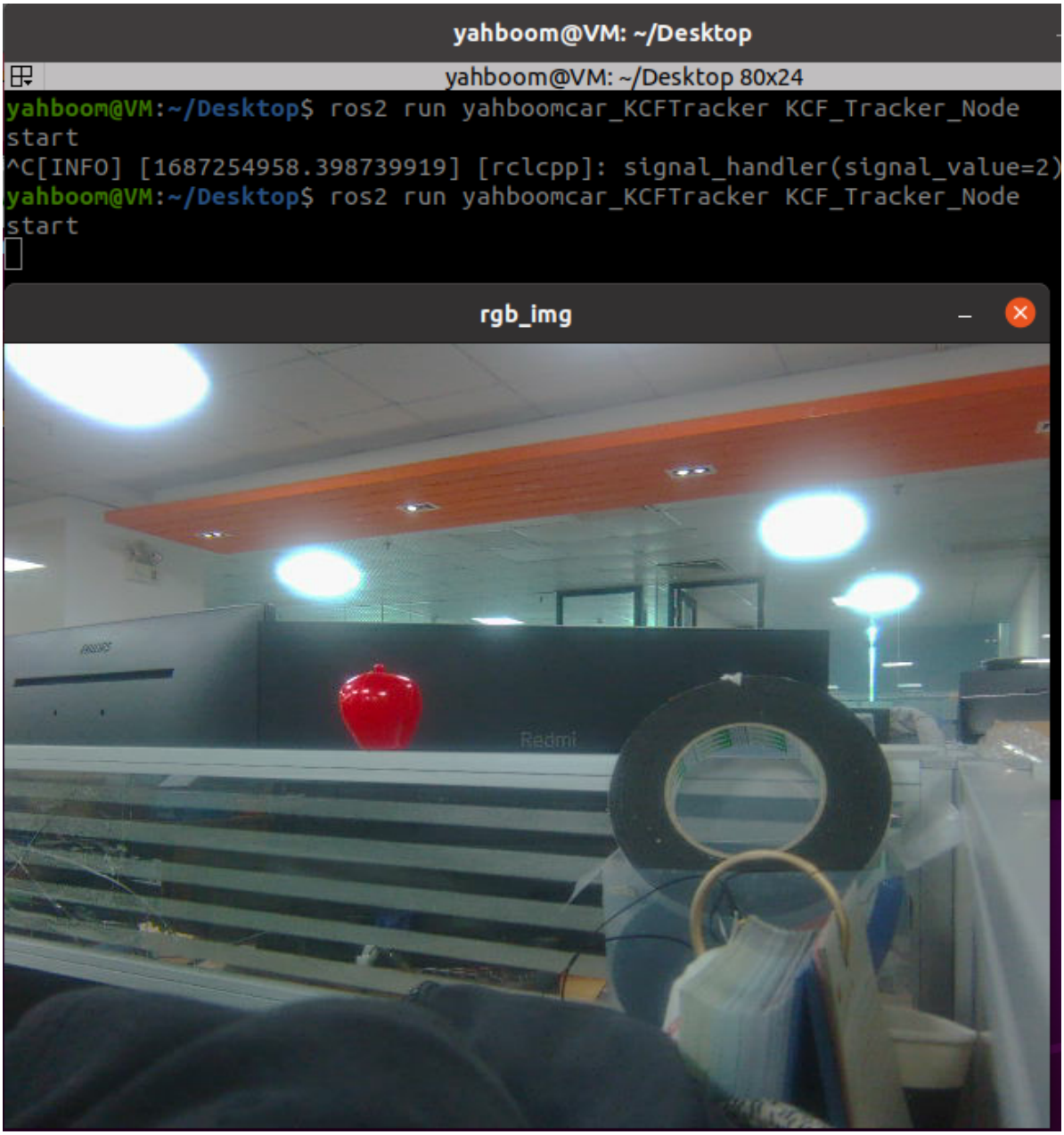
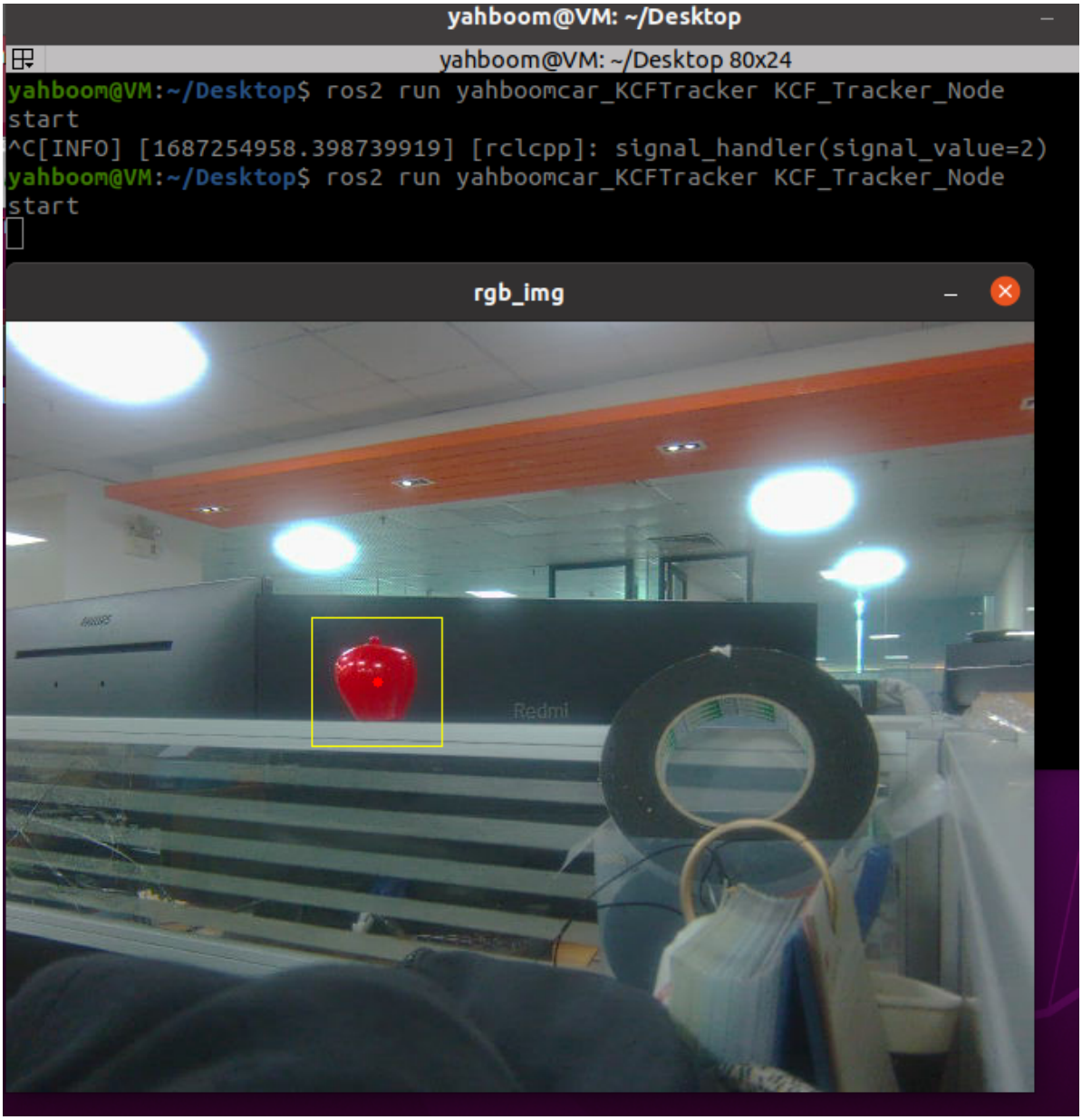
4.KCF object tracking

1.Start program

```
ros2 launch orbbec_camera orbbec_camera.launch.py
ros2 run yahboomcar_KCFTracker KCF_Tracker_Node
```



After the program successfully starts, enter the image shown in the above figure. Select the object to be tracked with the mouse, and release it to box it out, as shown in the following figure.



Then, press the spacebar to start tracking objects.
The terminal will print out the center coordinates and distance of the tracked object.

Similarly, without a robot chassis driver, it is not possible to visually observe the phenomenon of object tracking, but it can be achieved through terminal information and/cmd_ The changes in vel topic data indirectly reflect object tracking.
To view speed topic data, enter the following command:,

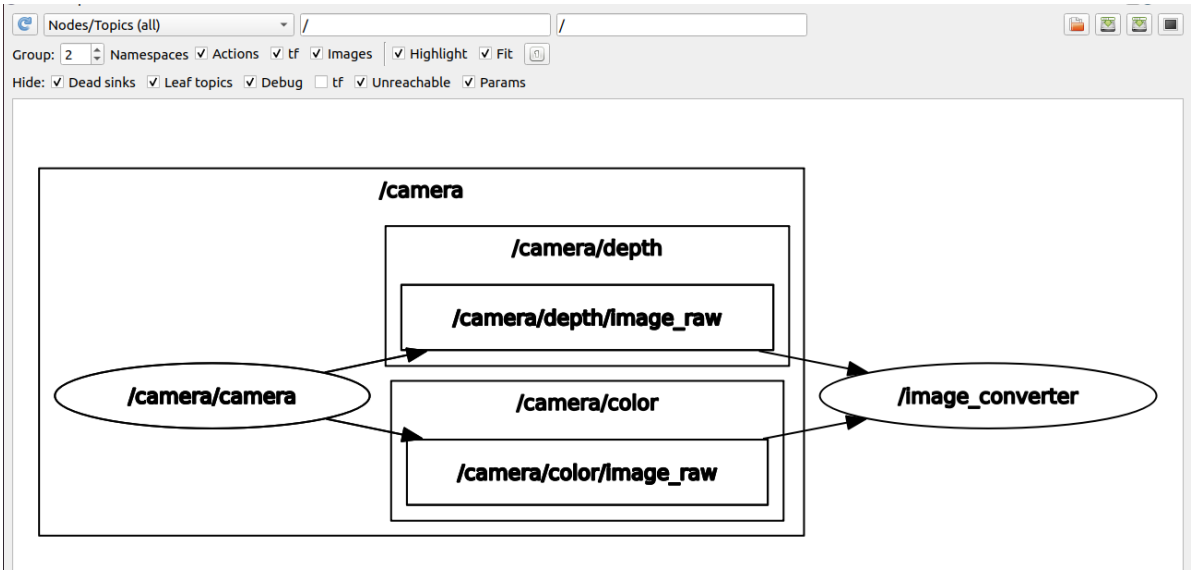
```
ros2 topic echo /cmd_vel
```

```
yahboom@VM:~/Desktop$ ros2 topic echo /cmd_vel
linear:
  x: -1.215250015258789
  y: 0.0
  z: 0.0
angular:
  x: 0.0
  y: 0.0
  z: 0.4450000524520874
```

Moving the tracked object also changes the velocity data here.

View communication between nodes, terminal input,

```
ros2 run rqt_graph rqt_graph
```



2. Code

Code path

```
~/orbbec_ws/src/yahboomcar_KCFTracker/src/KCF_Tracker.cpp
```

The principle of functional implementation is similar to color tracking, which calculates linear velocity and angular velocity based on the center coordinate of the target and the depth information fed by the depth camera, and then releases them to the chassis. Part of the code is as follows:,

```
//这部分是选择物体后，得出中心坐标，用于计算角速度
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
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
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)
```