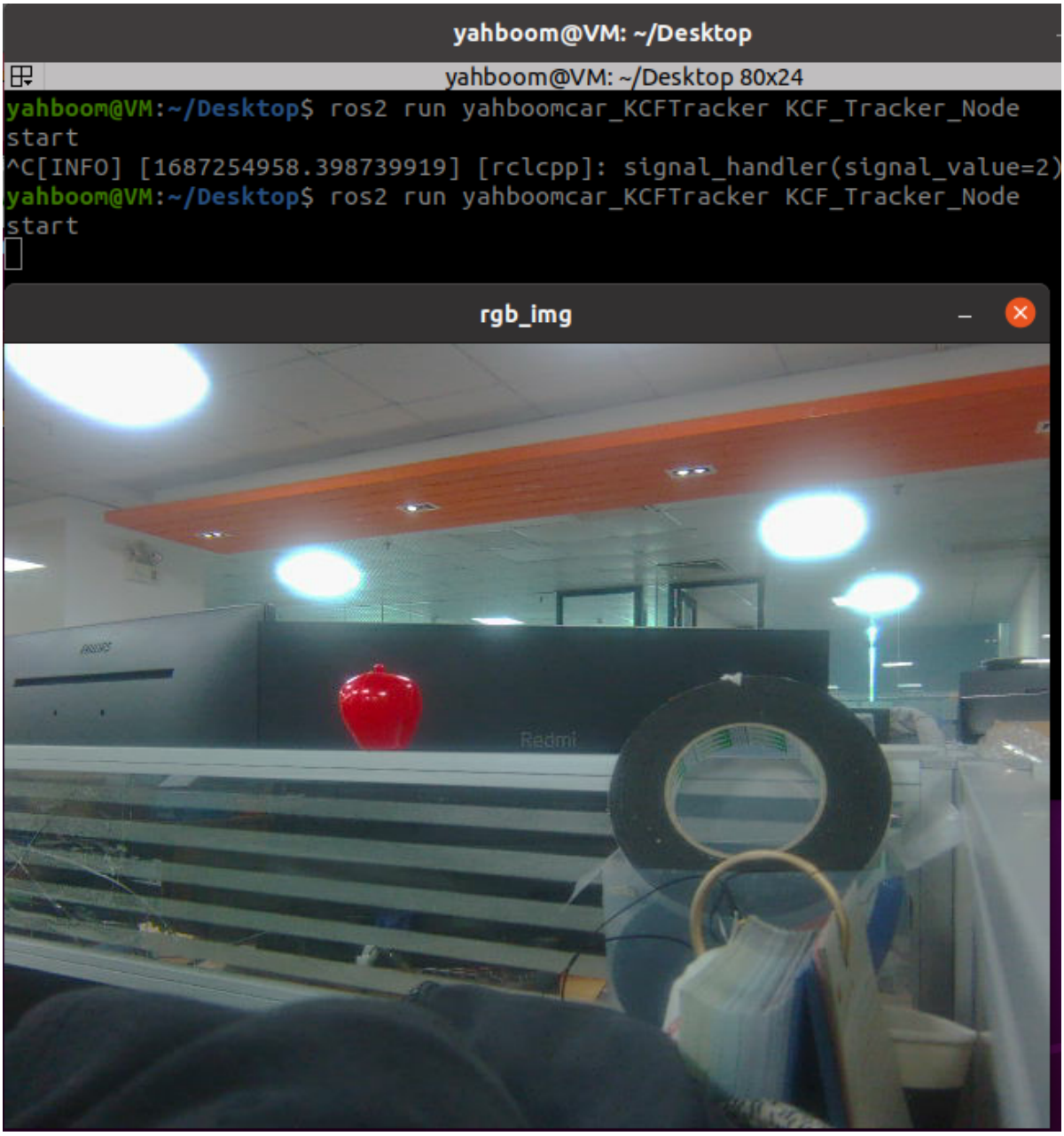
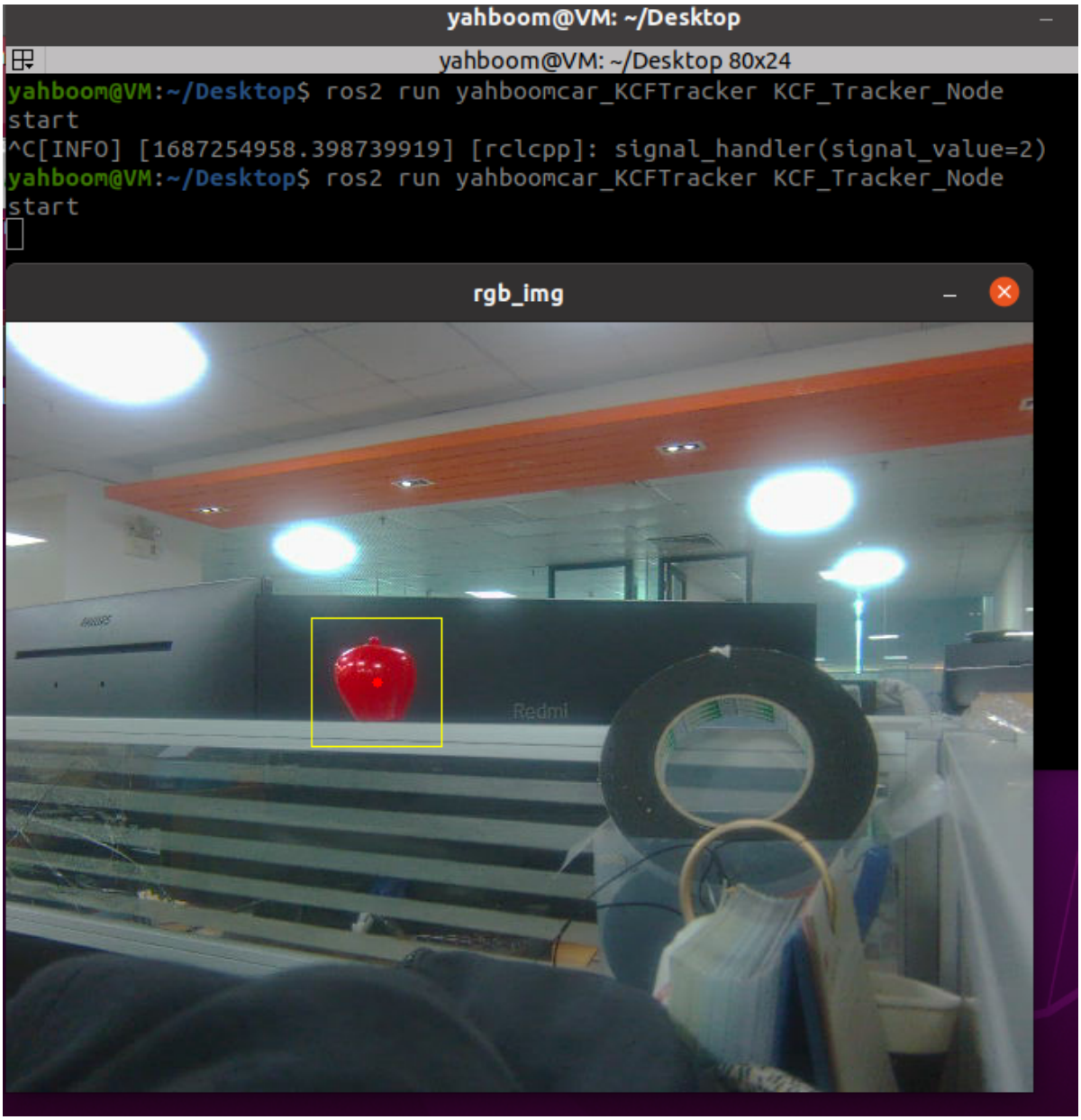
4. KCF object tracking

4.1. Program startup

```
#gemini camera startup
ros2 launch astra_camera gemini.launch.xml
ros2 run yahboomcar_KCFTracker KCF_Tracker_Node
```



After the program is successfully started, the above figure is shown. Select the object to be tracked with the mouse. After releasing it, it will be framed, as shown in the figure below.



Then press the space bar to start tracking the object. The terminal will print out the center coordinates and distance of the tracked object.

Similarly, since there is no robot chassis driver, the object tracking phenomenon cannot be seen intuitively, but the object tracking can be reflected indirectly through the terminal information and the changes in the /cmd_vel topic data. To view the speed topic data, enter the following command,

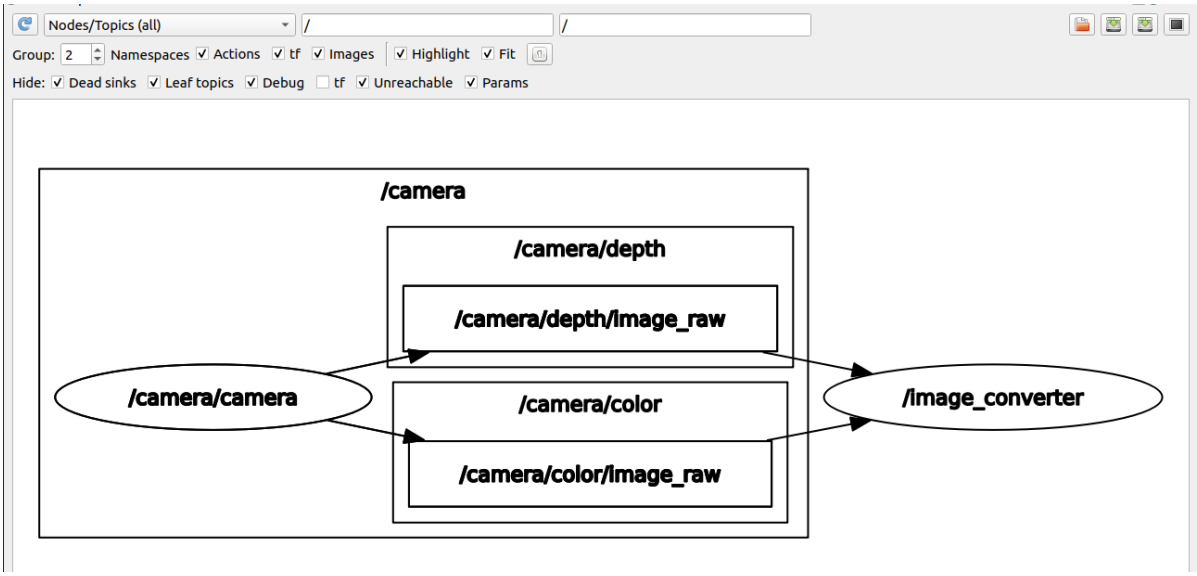
```
ros2 topic echo /cmd_vel
```

```
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
```

Move the tracked object, and the speed data here will also change.

Check the communication between nodes and input in the terminal.

```
ros2 run rqt_graph rqt_graph
```



4.2, core code

Code reference path,

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

The principle of function implementation is similar to color tracking. It calculates 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 are as follows,

```
//This part is to obtain the center coordinates after selecting the object, which
is 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);
//This part is to calculate the values of center_x and distance, which are used
to calculate the 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] > 0.4 && dist_val[i] < 10.0) distance += dist_val[i];
else num_depth_points--;
}
distance /= num_depth_points;
//Calculate linear and angular speeds
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)
```