4. Astra object tracking

- 4. Astra object tracking
  - 4.1. Introduction
  - 4.2, KCF object tracking
    - 4.2.1. How to use
    - 4.2.2. Keyboard control
    - 4.2.3. Node analysis

4.1. Introduction

Function package: ~/yahboomcar_ws/src/yahboomcar_astra

Official website: https://learnopencv.com/object-tracking-using-opencv-cpp-python/#opencv-tracking-api

- Object Tracking

Object tracking is to locate an object in consecutive video frames. This definition sounds straightforward, but in computer vision and machine learning, tracking is a very broad term that encompasses conceptually similar but technically different concepts. For example, the following are all different but related ideas commonly studied under object tracking:

(1) Dense Optical flow DOF: These algorithms help estimate the motion vector of each pixel in a video frame.

(2) Sparse optical flow: For example, the Kanade-Lucas-Tomashi (KLT) feature tracking algorithm tracks the positions of several feature points in the image.

(3) Kalman Filtering: A very popular signal processing algorithm based on prior motion information, used to predict the position of moving targets. One of the early applications of this algorithm was in missile guidance! The onboard computer that guided the Apollo 11 lunar module to the moon had a Kalman filter.

(4) Meanshift and Camshift: This is an algorithm for locating the maximum value of the density function. They are also used for tracking.

(5) Single object trackers: In this type of trackers, the first frame is marked with a rectangle to indicate the location of the object to be tracked. The object is then tracked in subsequent frames using a tracking algorithm. In most real-world applications, these trackers are used together with object detectors.

(6) Multiple object track finding algorithms: When we have a fast object detector, detect multiple objects in each frame, and then run a tracking finding algorithm to identify which object in a frame It makes sense for the rectangle to correspond to the rectangle in the next frame.

- Comparison of various algorithms

Algorithm	Speed	Accuracy	Description
BOOSTING	Slow	Poor	The same machine learning algorithm used behind Haar casades (AdaBoost), but it has been around for more than ten years and is a veteran algorithm.
MIL	Slow	Poor	More accurate than BOOSTING, but has a higher failure rate.
KCF	Fast	High	Faster than BOOSTING and MIL, but does not perform well in occlusion situations.
TLD	General	General	There are many false positives, and the phenomenon of skewing is serious.
MEDIANFLOW	General +	General	There are very few false positives. For fast jumping or fast moving objects, the model will fail.
GOTURN	General	General	Deep learning based object detector, which requires additional models to run.
MOSSE	Fastest	High -	The speed is really fast, but it is not as accurate as CSRT and KCF. If you are pursuing speed, you can choose it.
CSRT	Fast -	Highest	Slightly more accurate than KCF, but not as fast.

4.2, KCF object tracking

KCF stands for Kernel Correlation Filter Kernel Correlation Filtering Algorithm. It was proposed by Joao F. Henriques, Rui Caseiro, Pedro Martins, and Jorge Batista in 2014. After the algorithm came out, it was considered a sensation. This algorithm has very outstanding performance in both tracking effect and tracking speed. Therefore, a large number of scholars have been studying this algorithm, and the industry has also gradually applied this algorithm in actual scenarios. This [Algorithm Home Page] (http://www.robots.ox.ac.uk/~joao/circulant/index.html) contains papers and codes that can be downloaded here, as well as some introductions, etc. This article was published by the author on TPAMI in 2015, so you may see two versions, but there are no changes and both can be seen. Paper download address The related filtering algorithm is considered discriminative tracking. It mainly uses the given samples to train a discriminative classifier to determine whether the target or the surrounding background is tracked. information. The rotation matrix is mainly used to collect samples, and the fast Fourier transform is used to accelerate the calculation of the algorithm.

4.2.1. How to use

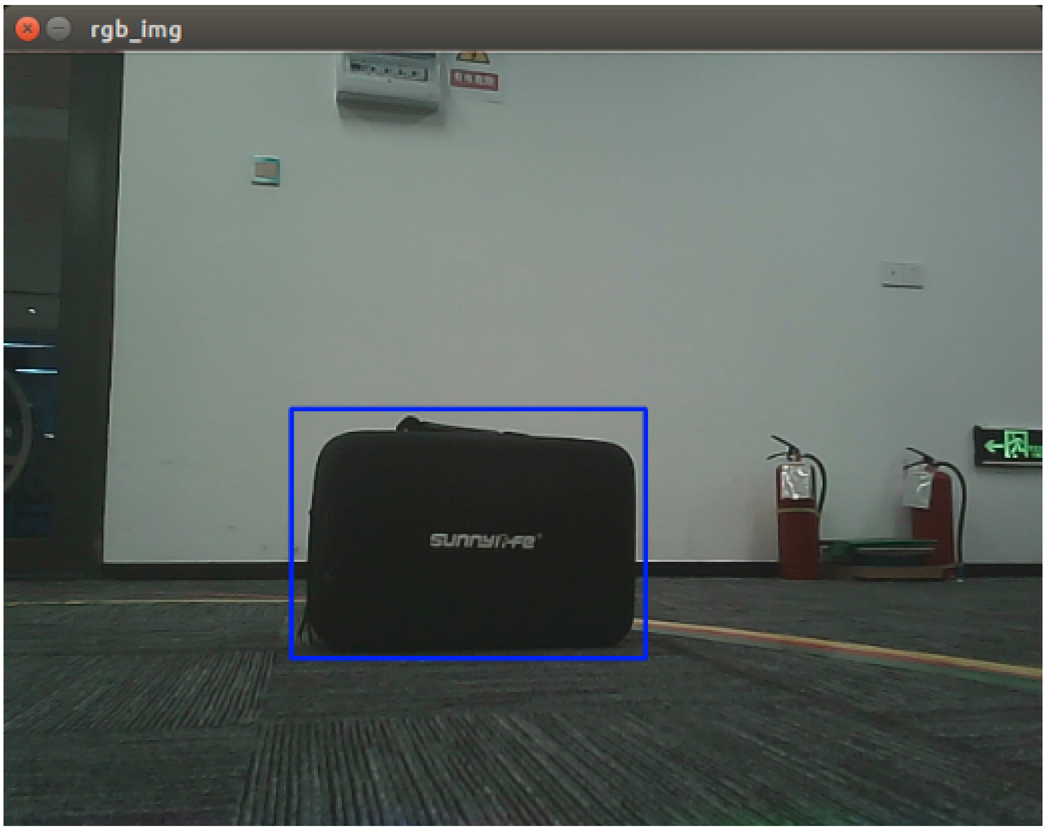
Note: [R2] on the remote control handle has the [pause/start] function for this gameplay.

```
#Raspberry Pi 5 master needs to enter docker first, please perform this step
#If running the script into docker fails, please refer to ROS/07, Docker tutorial
~/run_docker.sh
```

One-click startup (robot side)

```
roslaunch yahboomcar_astra KCFTracker.launch
```

After starting, enter the selection mode, use the mouse to select the location of the object, as shown in the figure below, release it to start recognition.

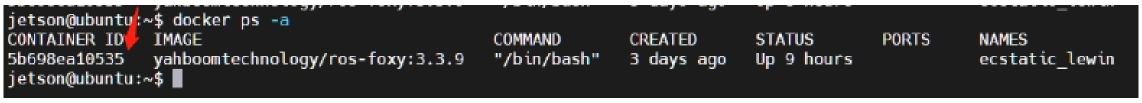


4.2.2. Keyboard control

<PI5 needs to open another terminal to enter the same docker container

1. In the above steps, a docker container has been opened. You can open another terminal on the host (car) to view:

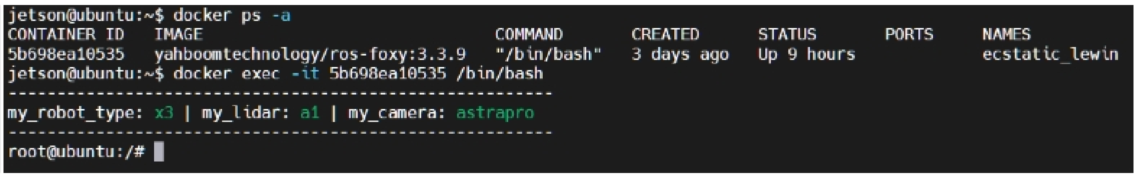
```
docker ps -a
```



```
jetson@ubuntu:~$ docker ps -a
CONTAINER ID   IMAGE                               COMMAND       CREATED      STATUS      PORTS     NAMES
5b698ea10535   yahboomtechnology/ros-foxy:3.3.9   "/bin/bash"   3 days ago   Up 9 hours            ecstatic_lewin
jetson@ubuntu:~$
```

2. Now enter the docker container in the newly opened terminal:

```
docker exec -it 5b698ea10535 /bin/bash
```



```
jetson@ubuntu:~$ docker ps -a
CONTAINER ID   IMAGE                               COMMAND       CREATED      STATUS      PORTS     NAMES
5b698ea10535   yahboomtechnology/ros-foxy:3.3.9   "/bin/bash"   3 days ago   Up 9 hours            ecstatic_lewin
jetson@ubuntu:~$ docker exec -it 5b698ea10535 /bin/bash
-------------------------------------------------------
my_robot_type: x3 | my_lidar: a1 | my_camera: astrapro
-------------------------------------------------------
root@ubuntu:/#
```

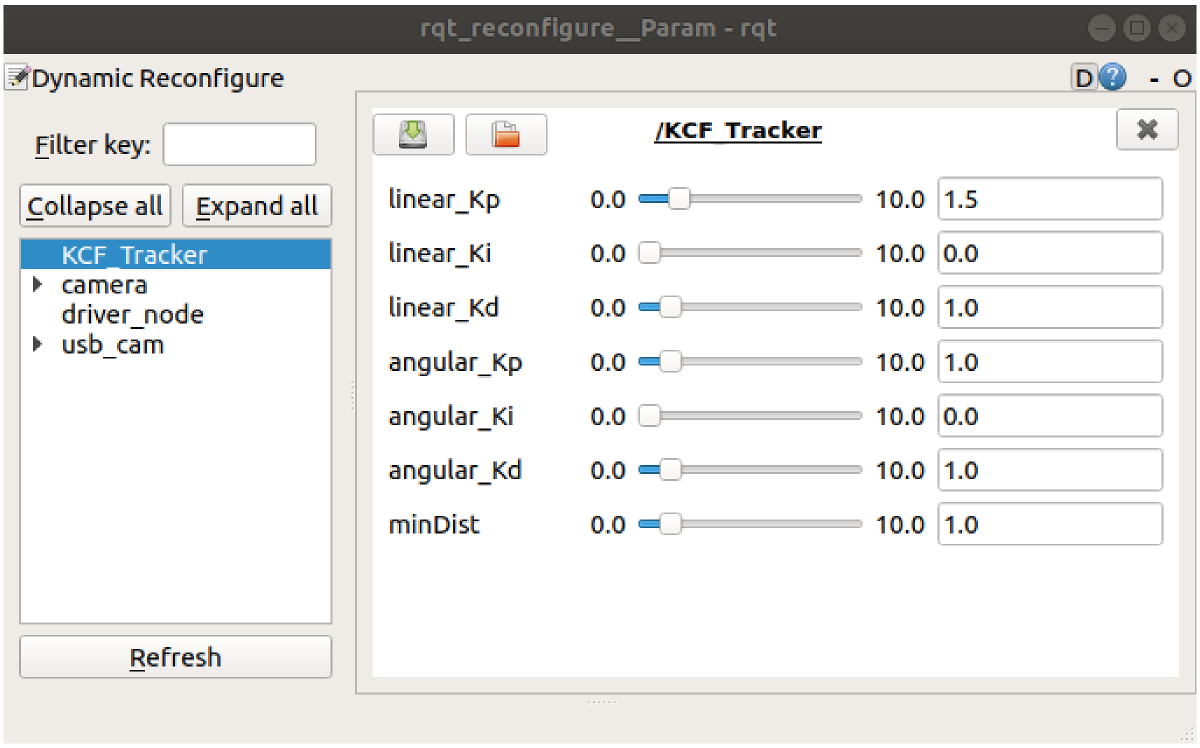
After successfully entering the container, you can open countless terminals to enter the container.

【r】: Reset mode, you can use the mouse to select the area to identify the target.

[q]: Exit the program.

[Spacebar]: Target tracking; just move the target slowly while following. If you move too fast, you will lose the target.

```
rosrun rqt_reconfigure rqt_reconfigure
```



linear_Kp	0.0	10.0	1.5
linear_Ki	0.0	10.0	0.0
linear_Kd	0.0	10.0	1.0
angular_Kp	0.0	10.0	1.0
angular_Ki	0.0	10.0	0.0
angular_Kd	0.0	10.0	1.0
minDist	0.0	10.0	1.0

Parameter analysis:

[linear_Kp], [linear_Ki], [linear_Kd]: PID control of linear speed during car following.

[angular_Kp], [angular_Ki], [angular_Kd]: PID control of angular velocity during car following.

[minDist]: Follow the distance and keep this distance.

- Parameter modification

When the parameters are adjusted to the optimal state, the corresponding parameters are modified into the file, and no adjustment is required when using again.

According to the optimal parameters of the [rqt_reconfigure] debugging tool, enter the [src] folder of the [yahboomcar_astra] function package and modify the parameters corresponding to the [KCF_Tracker.cpp] file, as shown below

```
ImageConverter::ImageConverter(ros::NodeHandle &n) {
     KCFTracker tracker(HOG, FIXEDWINDOW, MULTISCALE, LAB);
     float linear_KP=0.9;
     float linear_KI=0.0;
     float linear_KD=0.1;
     float angular_KP=0.5;
     float angular_KI=0.0;
     float angular_KD=0.2;
```

The minDist parameter is modified in the [KCF_Tracker.h] file

```
float minDist = 1.0;
```

[rqt_reconfigure] Modification of the initial value of the debugging tool

```
gen.add("linear_Kp", double_t, 0, "Kp in PID", 0.9, 0, 10.0)
gen.add("linear_Ki", double_t, 0, "Ki in PID", 0.0, 0, 10.0)
gen.add("linear_Kd", double_t, 0, "Kd in PID", 0.1, 0, 10.0)
gen.add("angular_Kp", double_t, 0, "Kp in PID", 0.5, 0, 10.0)
gen.add("angular_Ki", double_t, 0, "Ki in PID", 0.0, 0, 10.0)
gen.add("angular_Kd", double_t, 0, "Kd in PID", 0.2, 0, 10.0)
gen.add("minDist", double_t, 0, "minDist", 1.0, 0, 10.0)
exit(gen.generate(PACKAGE, "KCFTracker", "KCFTrackerPID"))
```

Enter the [cfg] folder of the [yahboomcar_astra] function package and modify the initial values of the parameters corresponding to the [KCFTracker.cfg] file.

```
gen.add("linear_Kp", double_t, 0, "Kp in PID", 0.9, 0, 10.0)
```

Take the above article as an example to analyze

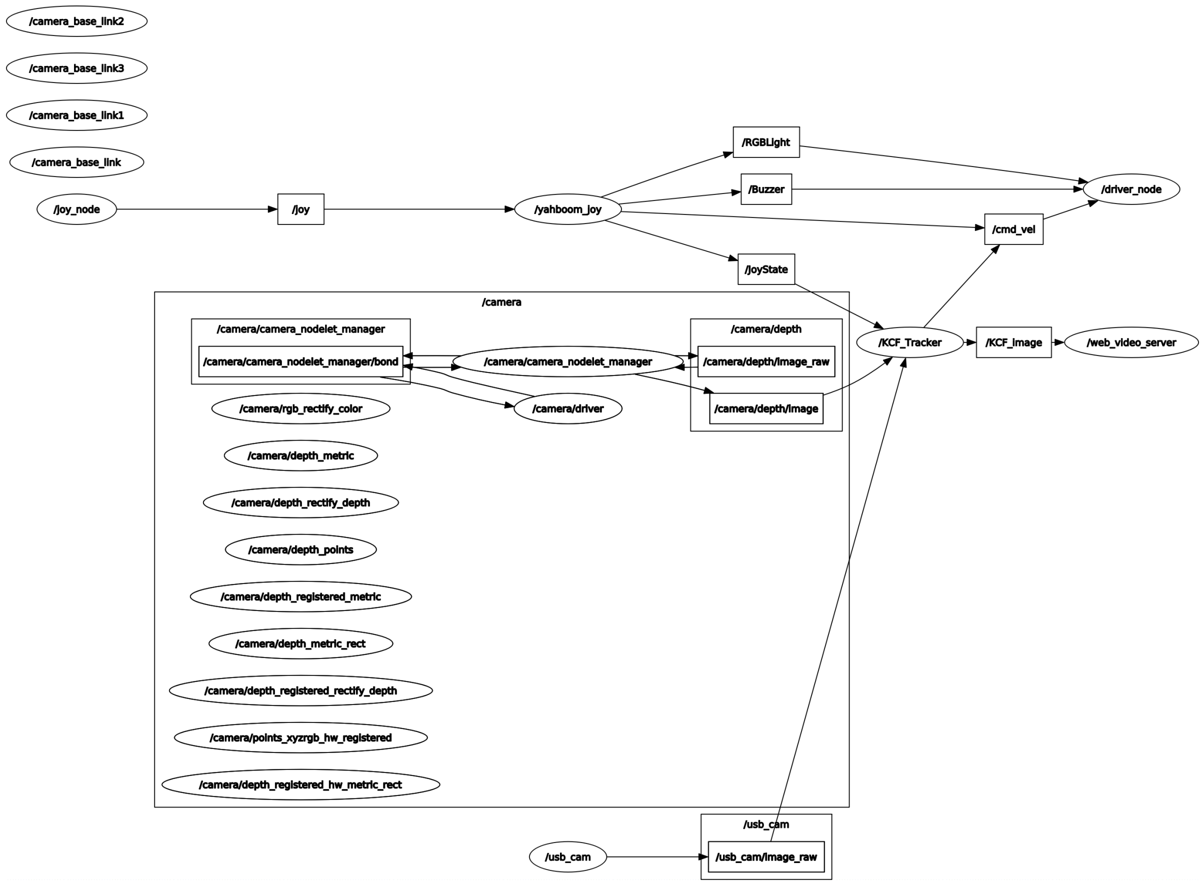
Parameters	Analysis	Corresponding parameters
name	name of the parameter	"linear_Kp"
type	parameter data type	double_t
level	a bitmask passed to the callback	0
description	A description parameter	"Kp in PID"
default	Initial value for node startup	0.9
min	parameter minimum value	0
max	parameter maximum value	10.0

Note: After modification, you must recompile and update the environment to be effective.

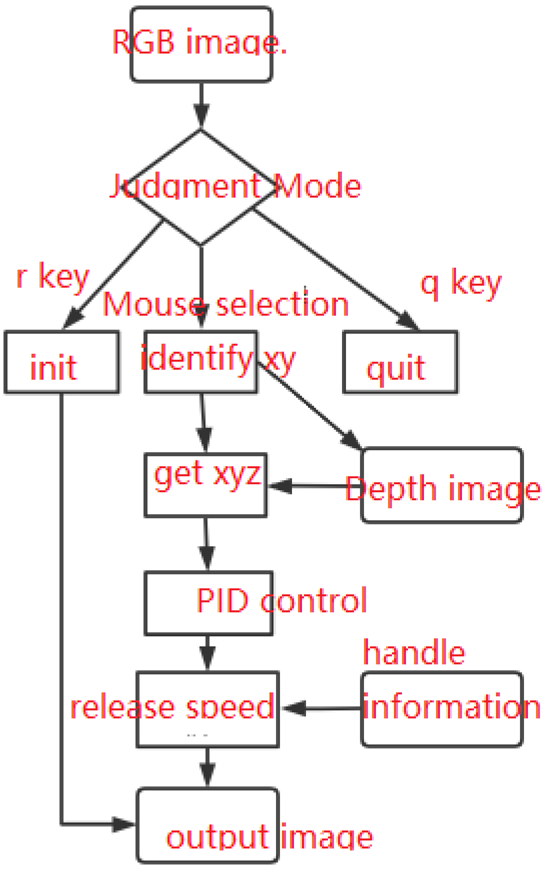
```
cd ~/yahboomcar_ws
catkin_make
source devel/setup.bash
```

4.2.3. Node analysis

```
rqt_graph
```



【KCF_Tracker】Node analysis



- Subscribe to RGB color images
- Subscribe to depth images
- Subscribe to handle control information
- Issue car speed control instructions