KCF Object Following

This lesson uses a Raspberry Pi 5 as an example.

For Raspberry Pi and Jetson Nano boards, you need to open a terminal on the host computer and enter the command to enter the Docker container. Once inside the Docker container, enter the commands mentioned in this lesson in the terminal. For instructions on entering the Docker container from the host computer, refer to **[01. Robot Configuration and Operation Guide] --[5.Enter Docker (For JETSON Nano and RPi 5)]**.

For Orin boards, simply open a terminal and enter the commands mentioned in this lesson.

1. Program Functionality

> After starting the program, use your mouse to select the object you want to follow on the screen. Press the spacebar to enter tracking mode. The robot will maintain a distance of 0.4 meters from the object and keep it centered on the screen at all times. Press the R key to enter selection mode, and press the Q key to exit the program.

⚠ This function works best with large selections; moving slowly can easily cause the target to be lost.

2. Program Code Reference Path

```
/root/yahboomcar_ros2_ws/yahboomcar_ws/src/yahboomcar_depth/yahboomcar_depth/kcf
/KCF_Tracker.py
/root/yahboomcar_ros2_ws/yahboomcar_ws/src/yahboomcar_depth/yahboomcar_depth/ast
ra_common.py
```

- mono_Tracker.py

This function primarily performs object detection and tracking. Based on the detected object center coordinates, it calculates the vehicle's turning angle, distance, and speed, and sends servo angle and motor speed control data to the vehicle.

- astra_common.py
    - Function (Tracker) class: Implements various OpenCV tracking algorithms (BOOSTING, KCF, CSRT, etc.)
    - Provides tracker initialization (initWorking) and real-time update (track) interfaces

3. Program Startup

3.1. Startup Commands

This lesson uses the Raspberry Pi 5 as an example.

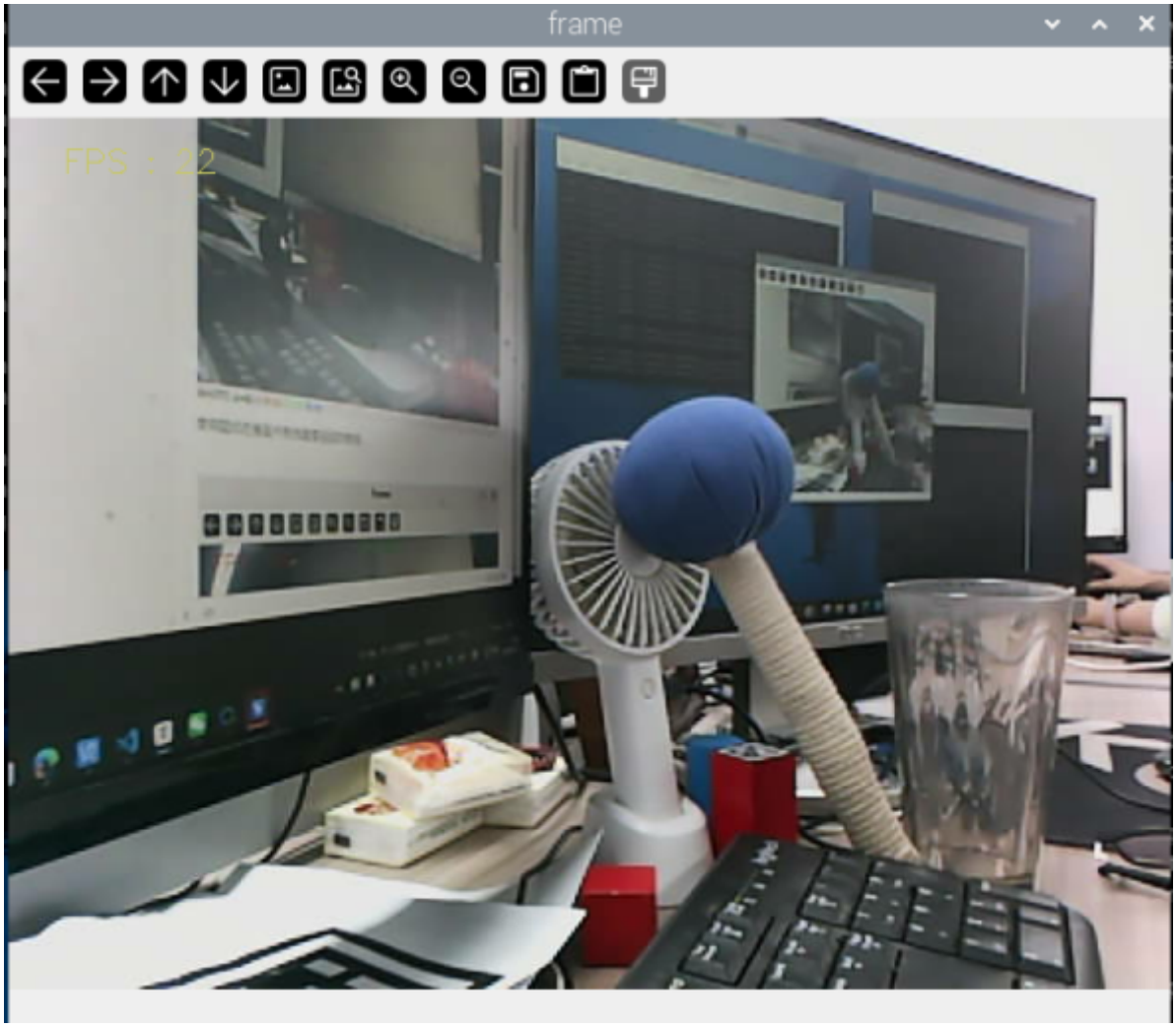
For Raspberry Pi and Jetson Nano boards, you need to open a terminal on the host computer and enter the command to enter the Docker container. Once inside the Docker container, enter the commands mentioned in this lesson in the terminal. For instructions on entering the Docker container from the host computer, refer to [5. Entering the Car Docker (Jetson Nano and Raspberry Pi 5 users, see here)] in this product tutorial [Robot Robot Configuration and Operation Guide].

For the Orin board, simply open a terminal and enter the commands mentioned in this lesson.

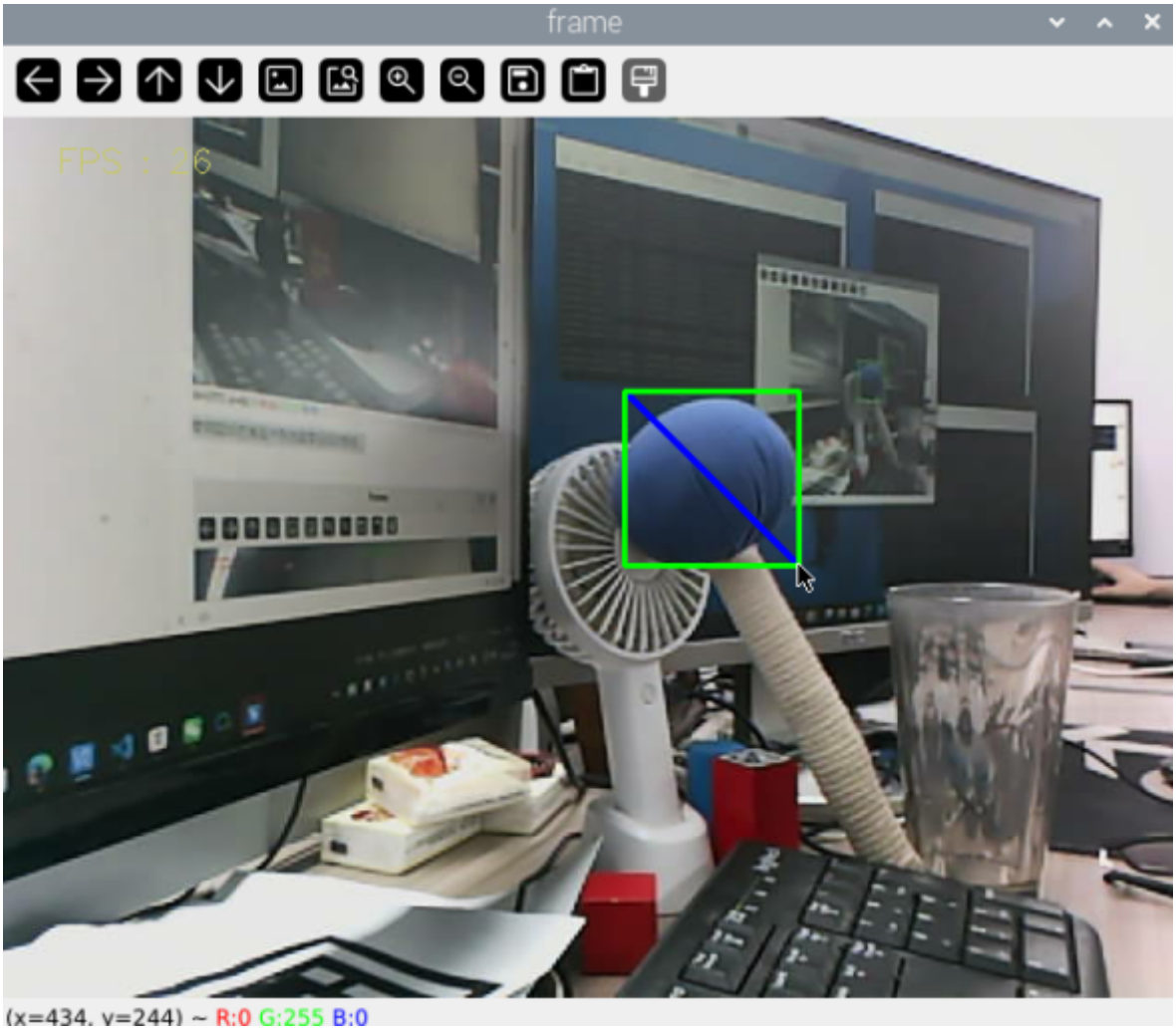
Enter in the terminal:

```
# Start the depth camera data
ros2 launch ascamera hp60c.launch.py
# Start the car chassis and radar
ros2 launch yahboomcar_bringup laser_bringup_launch.py
# Start the color tracking program
ros2 run yahboomcar_depth KCF_Tracker
```

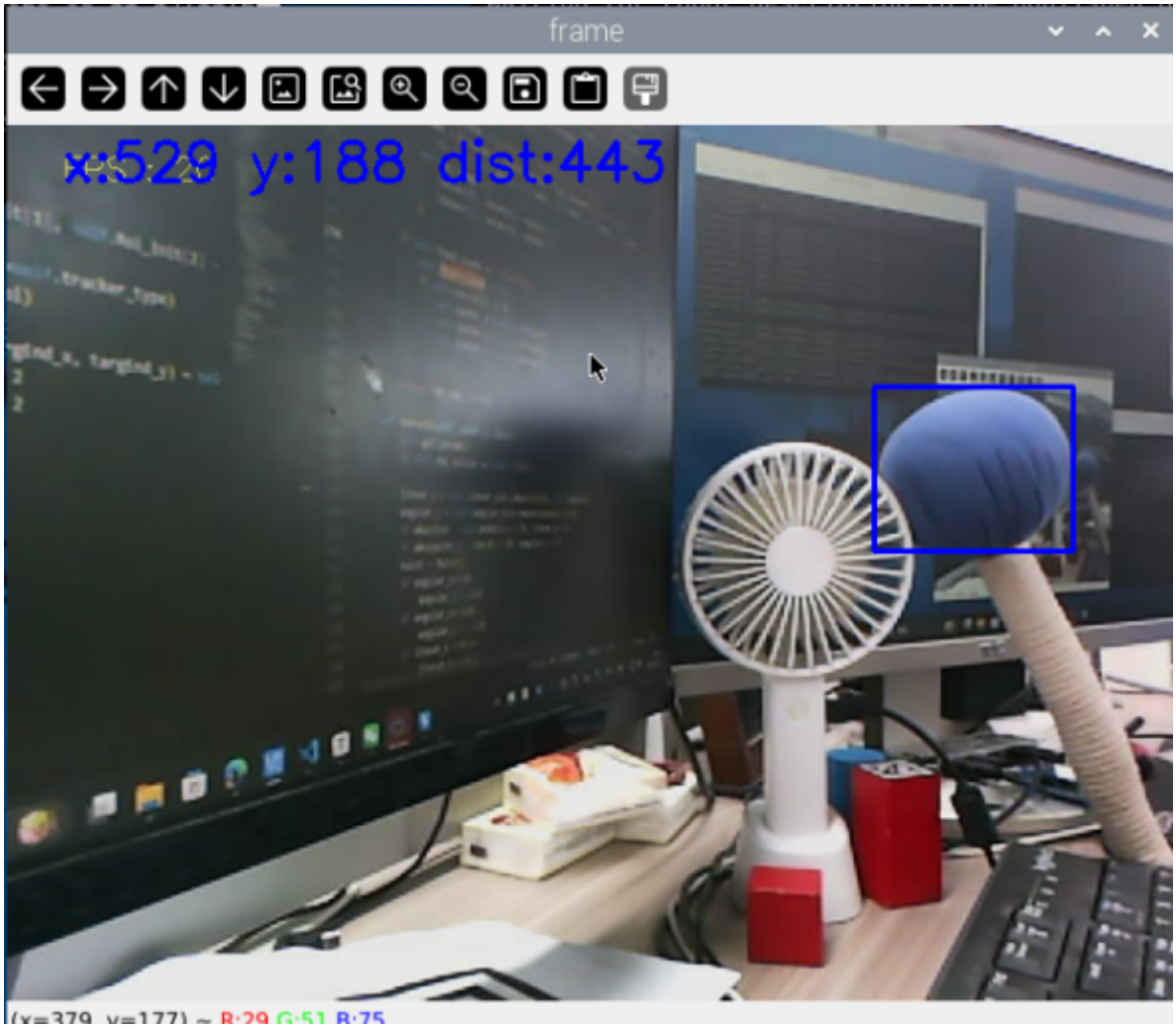
After the program starts, the following screen will appear:



Use the mouse to select the object to be tracked in the screen.



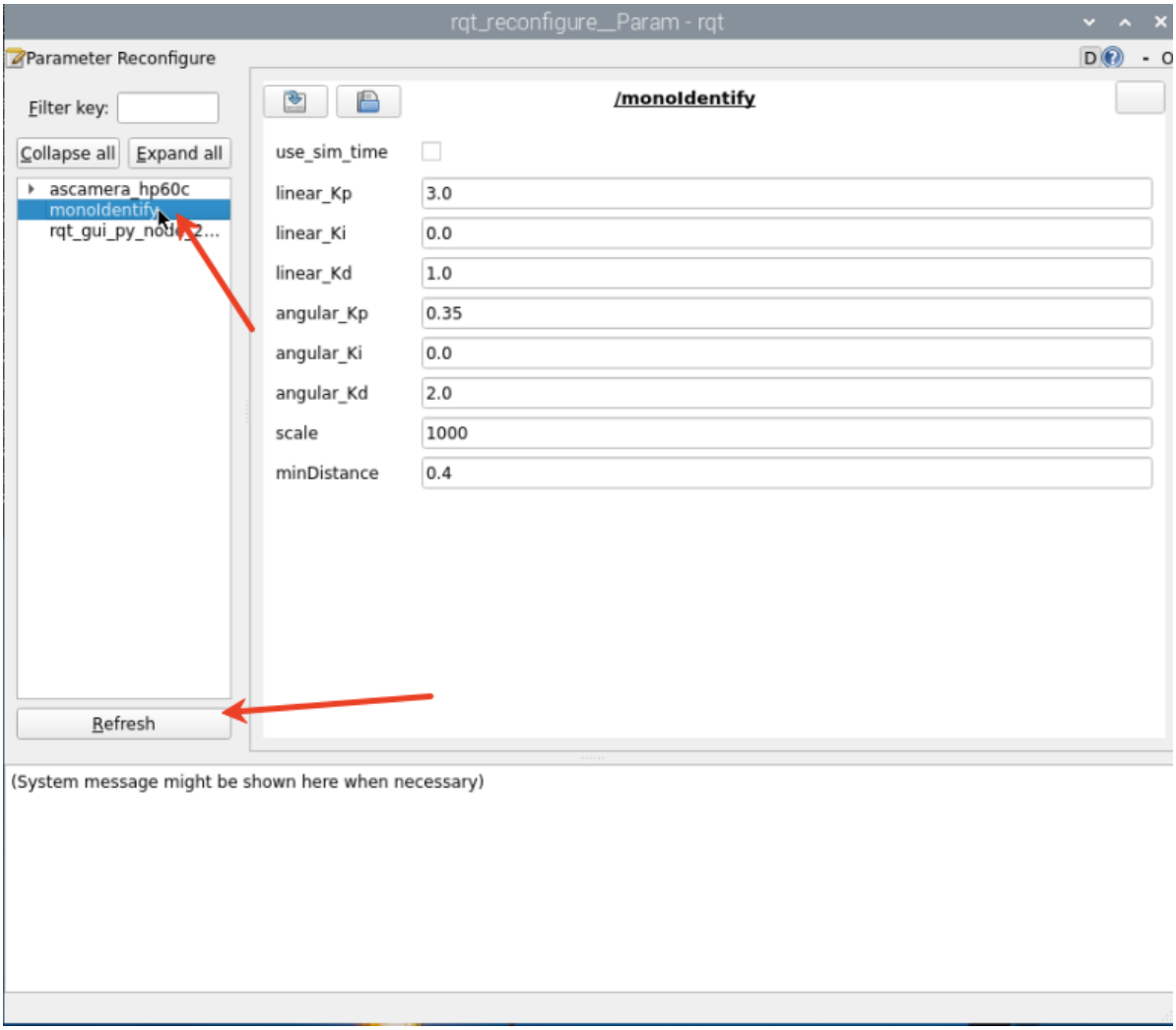
Then, press the spacebar to enter follow mode. Slowly move the object, and the car will follow, maintaining a distance of 1 meter from the object.



3.2 Dynamic Parameter Adjustment

Enter in the terminal:

```
ros2 run rqt_reconfigure rqt_reconfigure
```



☑ After modifying the parameters, click a blank space in the GUI to enter the parameter value. Note that this will only take effect for the current startup. To permanently take effect, you need to modify the parameters in the source code.

Parameter Explanation:

[linear_Kp], [linear_Ki], [linear_Kd]: PID control of linear velocity during the robot's following process.

[angular_Kp], [angular_Ki], and [angular_Kd]: PID control of the angular velocity during the car's following process.

[minDistance]: Following distance, which is maintained at that distance.

[scale]: PID scaling.

4. Core Code

4.1. KCF_Tracker.py

This program has the following main functions:

- Initializes the KCF tracker
- Subscribes to a depth topic and obtains images
- Selects a tracking target using mouse interaction
- Calculates the target center coordinates and publishes them
- Uses the PID algorithm to calculate the servo angle and forward speed and issues control commands

Some of the core code is as follows.

```
#追踪初始化 Tracking Initialization
self.tracker_type = 'KCF'

#距离计算 #Distance calculation
points = [
        (int(self.cy), int(self.cx)),          # 中心点 Center Point
        (int(self.cy +1), int(self.cx +1)),  # 右下偏移点 Lower right offset point
        (int(self.cy - 1), int(self.cx - 1))   # 左上偏移点 Upper left offset
point
    ]
valid_depths = []
for y, x in points:
    depth_val = depth_image_info[y][x]
    if depth_val != 0:
        valid_depths.append(depth_val)
if valid_depths:
    dist = int(sum(valid_depths) / len(valid_depths))
else:
    dist = 0
#控制执行 #Control execution
if dist > 0.2:  # 有效距离阈值(0.2米) Effective distance threshold (0.2 meters)
    self.execute(self.Center_x, dist)  # 执行控制 Execution control
# 根据x值、y值，使用PID算法，计算运动速度，转弯角度
# Calculate the movement speed and turning angle using the PID algorithm based on
the x and y values
def execute(self, rgb_img, action):
    #PID计算偏差 PID calculation deviation
    linear_x = self.linear_pid.compute(dist, self.minDist)
    angular_z = self.angular_pid.compute(point_x,320)
    # 小车停车区间，速度为0 The car is in the parking area and the speed is 0
    if abs(dist - self.minDist) < 30: linear_x = 0
    if abs(point_x - 320.0) < 30: angular_z = 0
    twist = Twist()
    ...
    # 将计算后的速度信息发布 Publish the calculated speed information
    self.pub_cmdVel.publish(twist)
```