KCF Object Tracking

1. Program Functionality

> After launching the program, use your mouse to select the object you want to track. Press the spacebar to enter tracking mode. The servo gimbal will lock onto the selected object, keeping it in the center of the screen. Press the q/Q key to exit the program.

⚠ This function works best with larger selections. Move slowly to avoid losing the target.

2. Program Code Reference Path

```
~/yahboomcar_ros2_ws/yahboomcar_ws/src/yahboomcar_astra/yahboomcar_astra/mono_Tracker.py
~/yahboomcar_ros2_ws/yahboomcar_ws/src/yahboomcar_astra/yahboomcar_astra/common/track_common.py
```

- mono_Tracker.py
  Mainly performs object detection and tracking. Based on the detected object center coordinates, it calculates the desired servo rotation angle and sends the servo angle control data to the robot.
- track_common.py
    - Implements multiple OpenCV tracking algorithms (BOOSTING, KCF, CSRT, etc.)
    - Provides tracker initialization (initWorking) and real-time update (track) interfaces

3. Program Startup

3.1. Startup Command

For Raspberry Pi and Jetson-Nano boards, you need to enter the Docker container first. For RDKX5 and Orin controllers, this is not necessary.

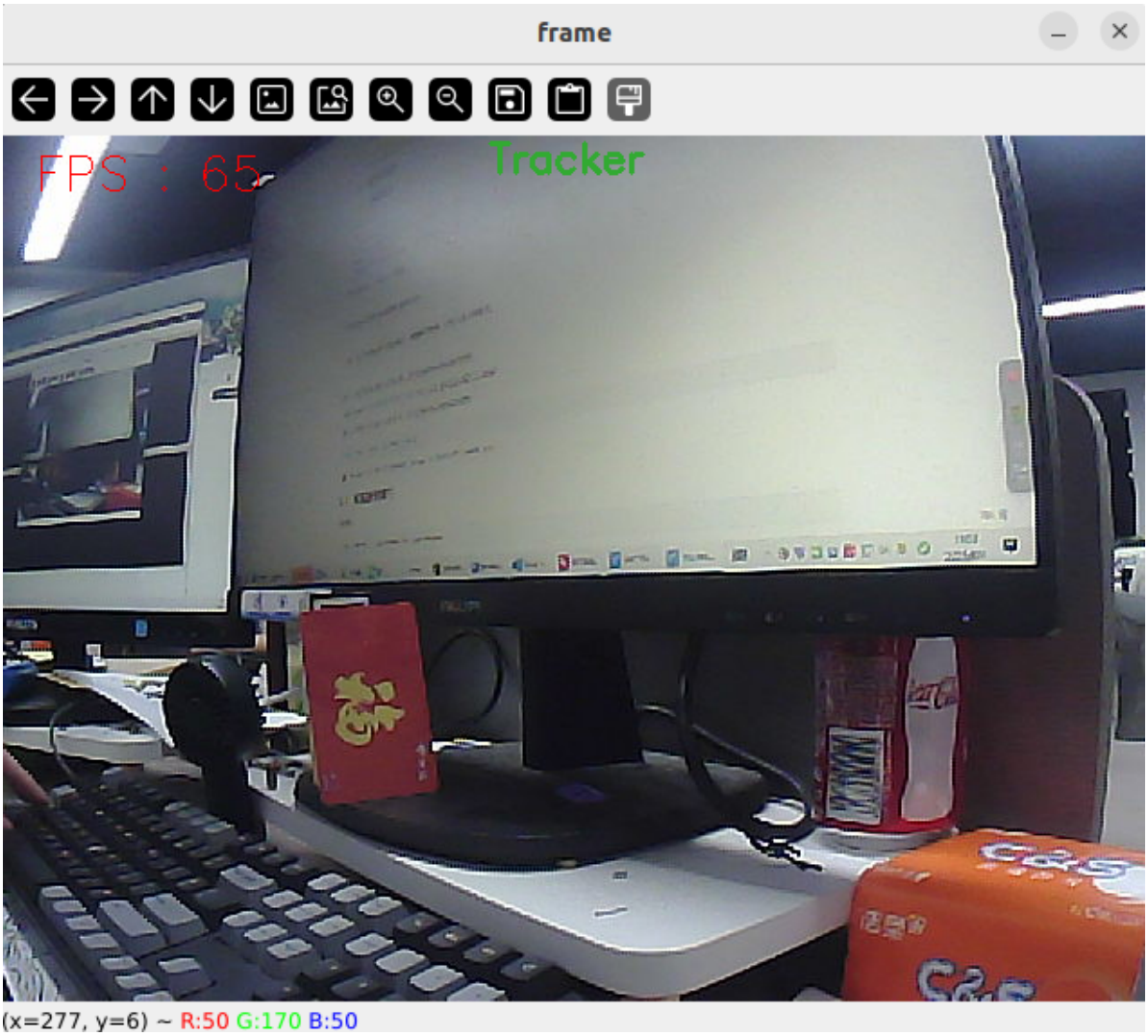
Enter the Docker container (see [Docker course] --- [4. Docker Startup Script] for steps).

All the following commands must be executed within the same Docker container **(see [Docker course] --- [3. Docker Submission and Multi-Terminal Access] for steps).**

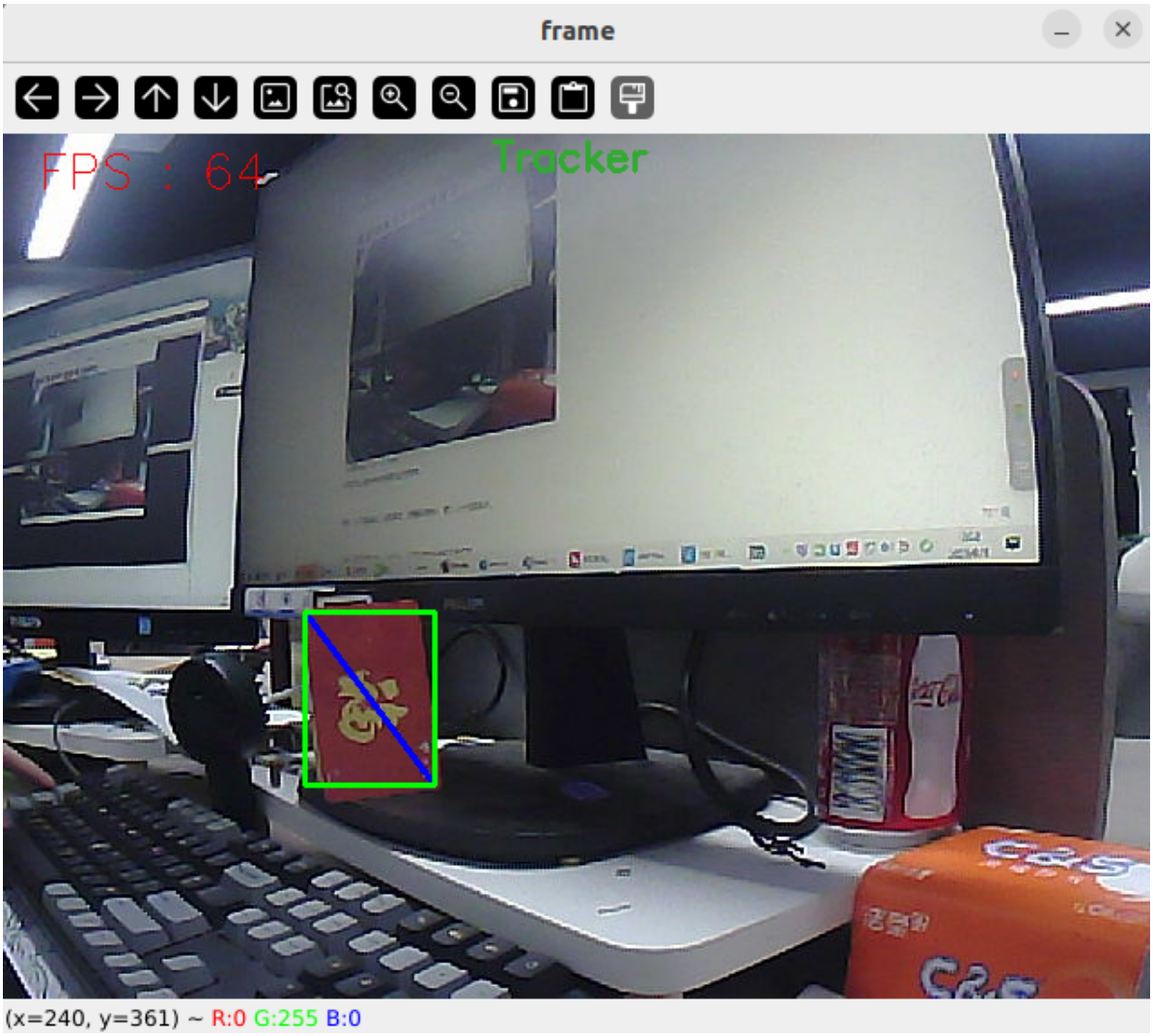
Enter the terminal:

```
# Start the car chassis
ros2 run yahboomcar_bringup Mcnamu_driver_M1
# Start the KCF object tracking program
ros2 run yahboomcar_astra monoTracker
```

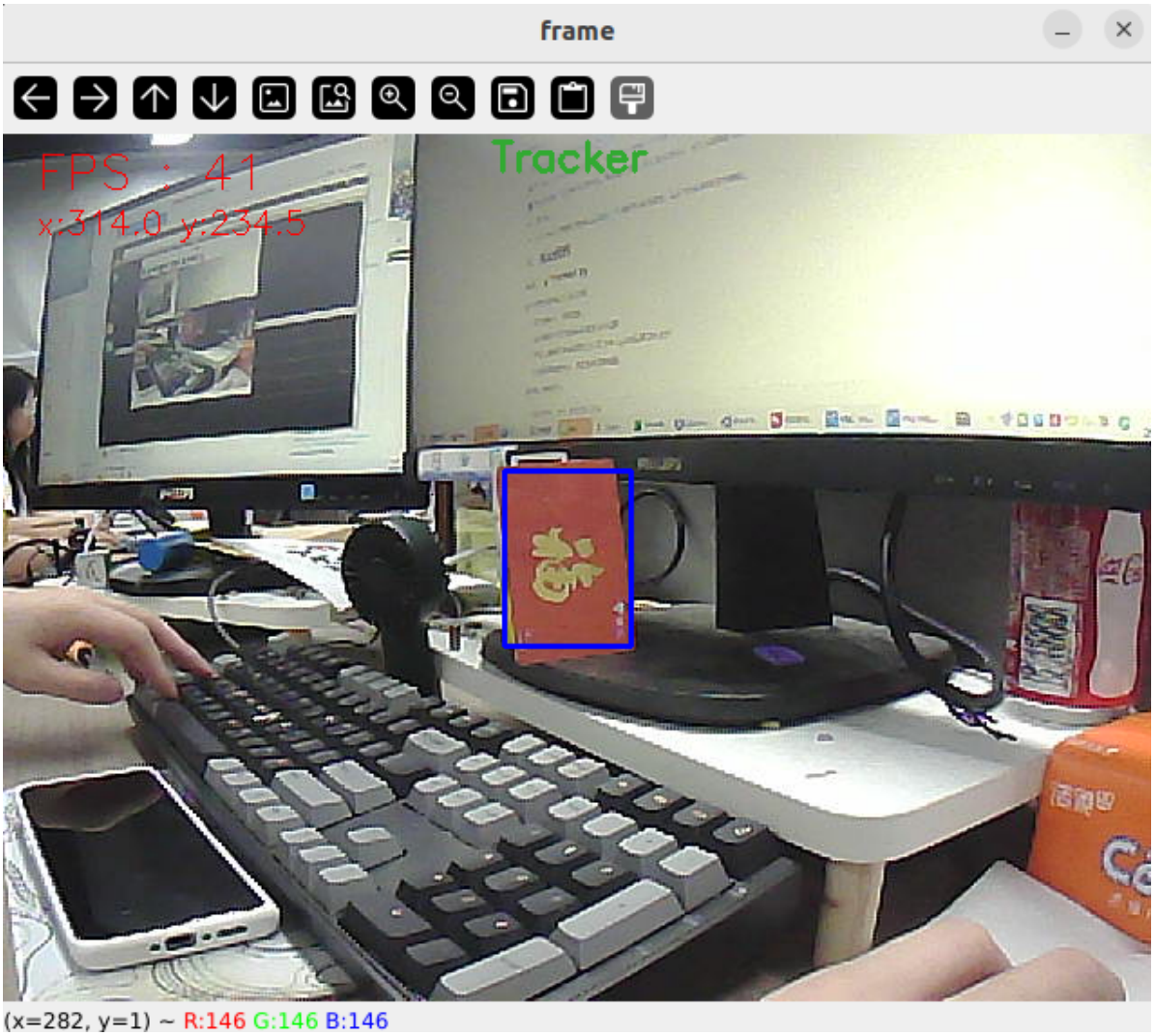
After the program starts, the following screen will appear.



Use your mouse to select the object you want to track.



Then, press the spacebar to enter tracking mode. Slowly move the object, and the servo gimbal will follow.



In addition, we can enter the following command to print information about the target center coordinates:

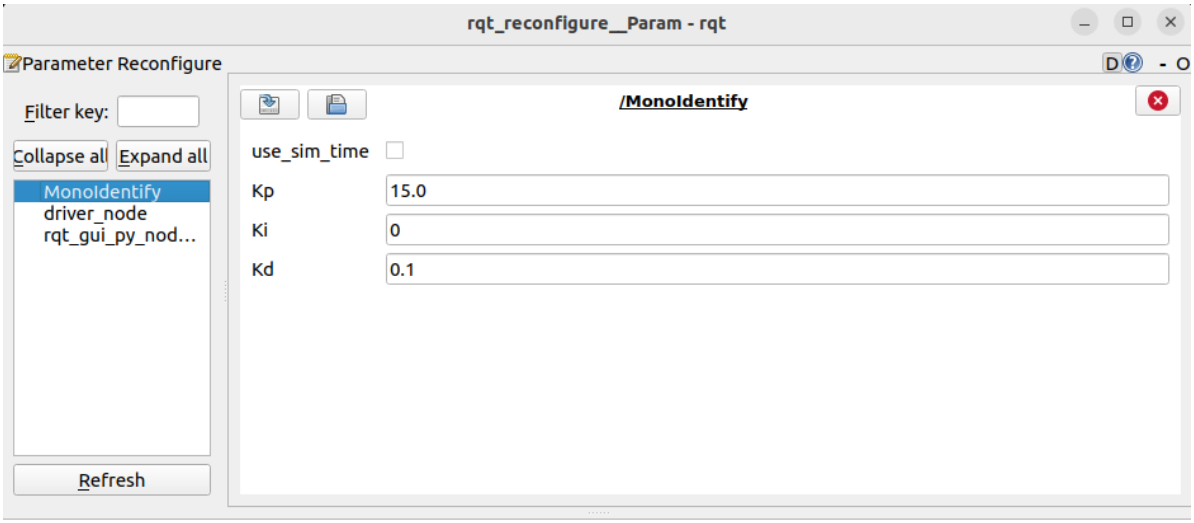
```
ros2 topic echo /Current_point
```

```
jetson@yahboom:~$ ros2 topic echo /Current_point
anglex: 42.0
angley: 220.0
distance: 0.0
---
anglex: 49.0
angley: 222.0
distance: 0.0
---
```

3.2 Dynamic Parameter Adjustment

In the terminal, enter:

```
ros2 run rqt_reconfigure rqt_reconfigure
```



☑ After modifying the parameters, click a blank area in the GUI to write the parameter values. Note that the values will only take effect during the current boot. To permanently apply them, you need to modify the parameters in the source code.

As shown in the figure above,

- mono_Tracker is primarily responsible for servo gimbal movement, adjusting PID-related parameters to achieve optimal gimbal motion.

🛠 Parameter Analysis:

[Kp], [Ki], [Kd]: PID control of the servo gimbal speed during tracking.

4. Core Code

4.1. mono_Tracker.py

This program has the following main functions:

- Initialize the KCF tracker
- Open the camera and acquire an image
- Select the tracking target using mouse interaction
- Calculate the target center coordinates and publish them
- Use the PID algorithm to calculate the servo angle and issue control commands

Some core code is as follows:

```
# Create a publisher to publish the center coordinates of the tracked object
self.pub_position = self.create_publisher(Position, "/Current_point", 10)
# Define the servo data publisher
self.pub_Servo = self.create_publisher(ServoControl, 'Servo', 10)
...
# Tracker initialization
self.tracker_types = ['BOOSTING', 'MIL', 'KCF']
self.tracker_type = ['KCF']
self.gTracker = Tracker(tracker_type=self.tracker_type)
...
# Mouse callback function, used to select the tracking target
def onMouse(self, event, x, y, flags, param):
    if event == 1:  # Left mouse button pressed
        self.Track_state = 'init'
```

```
        self.select_flags = True
        self.Mouse_XY = (x,y)
    if event == 4:  # Left mouse button released
        self.select_flags = False
        self.Track_state = 'identify'
...
# Calculate the servo angle using the PID algorithm based on the x and y values
def execute(self, point_x, point_y):
    position = Position()
    position.anglex = point_x * 1.0
    position.angley = point_y * 1.0
    # Publish the center coordinates message
    self.pub_position.publish(position)
    ...
    # Limit PID polarity and maximum value
    [x_Pid, y_Pid] = self.PID_controller.update([point_x - 320, point_y - 240])
    x_Pid = x_Pid * (abs(x_Pid) <= self.Kp/3)
    y_Pid = y_Pid * (abs(y_Pid) <= self.Kp/3)
    ...
    # Publish the calculated servo angle
    self.pub_Servo.publish(self.servo_angle)
```