Face Following

1. Program Functionality

> After starting the program, it automatically detects faces in the image, and the robot enters follow mode. The servos and gimbal will lock onto the detected face and keep it centered in the frame. Press the `q/Q key` to exit the program. The remote controller's `R2` key has the [Pause/Start] function for this gameplay.

⚠ This function has a high recognition rate for faces in 2D images, but has a poor recognition rate for real people!

📝 The following object example uses a mobile phone image. If you change the following object, you will need to adjust the corresponding parameters to achieve your desired effect!

2. Program Code Reference Path

```
~/yahboomcar_ros2_ws/yahboomcar_ws/src/yahboomcar_astra/yahboomcar_astra/faceFol
low.py
```

- faceFollow.py

  Mainly performs face detection. Based on the detected face center coordinates, it calculates the required servo rotation angle and the required vehicle angular velocity. It also calculates the required vehicle linear velocity based on the size of the face area. After PID calculation, the data is sent to the vehicle chassis.

3. Program Startup

3.1. Startup Commands

For the Raspberry Pi 5 controller, you must first enter the Docker container. For the Orin board, this is not necessary.

Enter the Docker container (for steps, see the Docker section [3. Docker Submission and Multi-Terminal Access]).

Enter the terminal:

```
# Start the car chassis and joystick nodes
ros2 launch yahboomcar_bringup yahboomcar_bringup_A1_launch.py
# Start the face following program
ros2 run yahboomcar_astra faceFollow
```
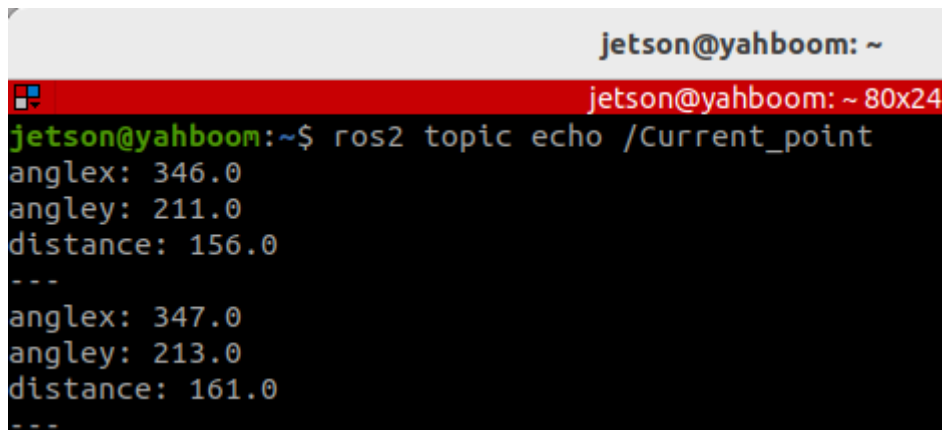
After the program starts, the camera image will appear and faces will be detected in real time.

When a face is detected, a green rectangle will be drawn around the face, and the face center coordinates and area size will be displayed.
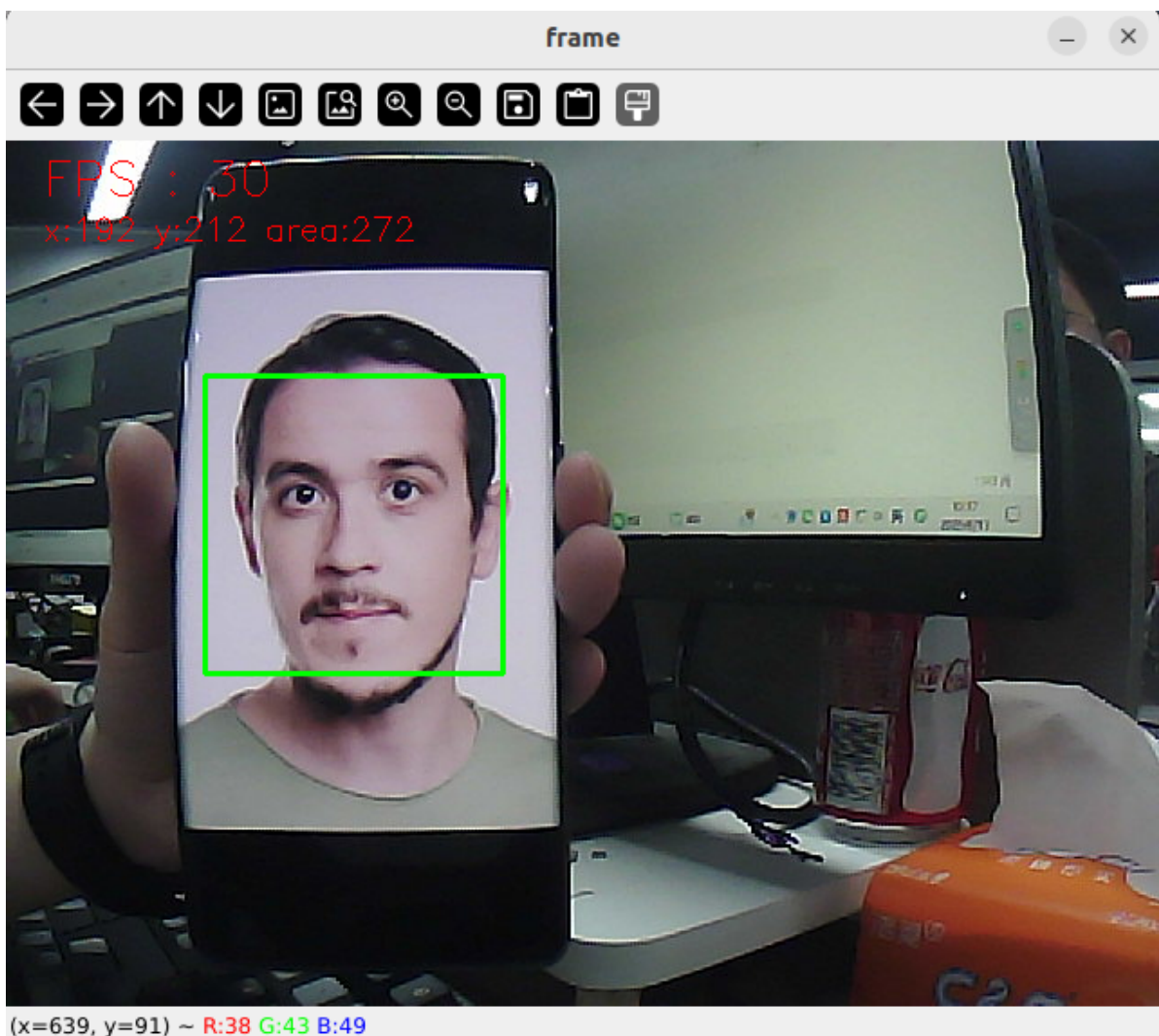
Then, the robot enters follow mode. Slowly move the object, and the robot and servo gimbal will follow.

Alternatively, we can enter the following command to print information about the target center coordinates and radius:

```
ros2 topic echo /Current_point
```


3.2 Dynamic Parameter Adjustment

Enter in the terminal:

```
ros2 run rqt_reconfigure rqt_reconfigure
```
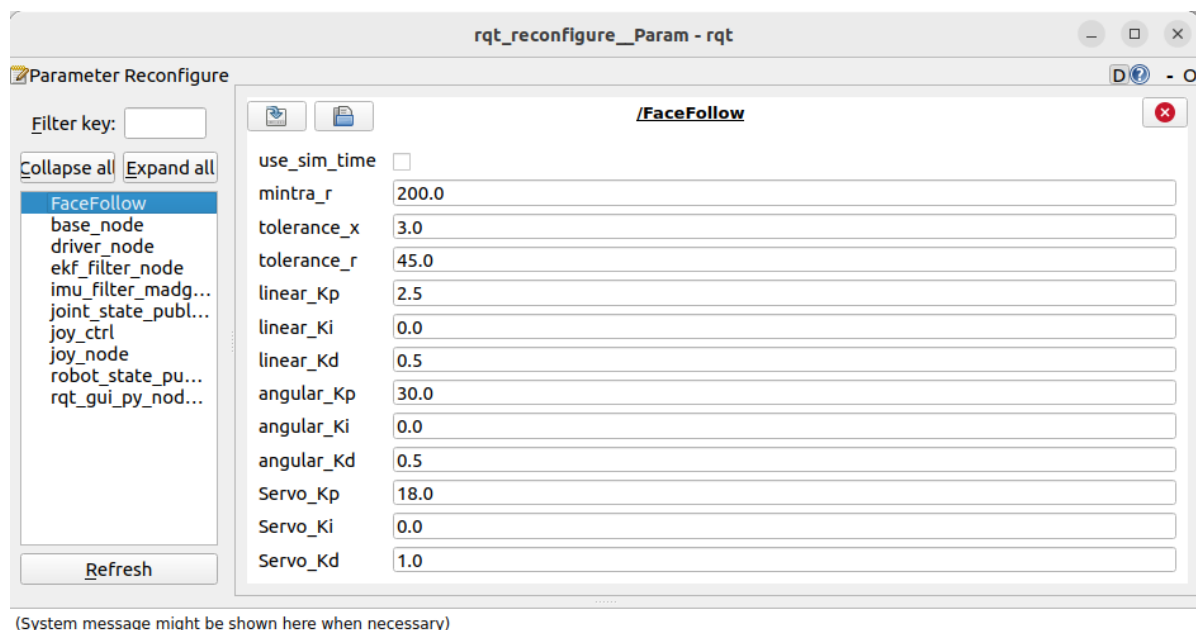

☑ After modifying the parameters, click a blank area in the GUI to enter the parameter value. Note that this will only take effect for the current boot. To permanently take effect, you need to modify the parameters in the source code.

As shown in the figure above:

faceFollow is primarily responsible for robot motion. PID-related parameters can be adjusted to optimize robot motion.

✖ Parameter Analysis:

[mintra_r]: Critical tracking area. When the face area is smaller than this value, the robot moves forward; when it is larger than this value, the robot moves backward.

[tolerance_x] and [tolerance_r]: X-axis position tolerance. When the deviation between the face center's x-coordinate and the screen center (`320`) is smaller than this value, alignment is considered complete. Area tolerance: When the deviation between the detected face area and the critical tracking area (`mintra_r`) is smaller than this value, the distance is considered appropriate.

[linear_Kp], [linear_Ki], [linear_Kd]: Linear velocity PID control during robot following.

[angular_Kp], [angular_Ki], [angular_Kd]: Angular velocity PID control during robot following.

[Servo_Kp], [Servo_Ki], [Servo_Kd]: PID control of the servo gimbal speed during the following process.

4. Core Code

4.1. faceFollow.py

This program has the following main functions:

- Opens the camera and acquires an image;
- Uses OpenCV's Haar feature classifier to detect faces;
- Calculates the center coordinates and face area size;
- Calculates the robot's movement speed and servo angle, and issues control commands.

The following is some of the core code:

```
# Define the servo data publisher
self.pub_Servo = self.create_publisher(ServoControl, 'Servo', 10)
# Define the vehicle velocity publisher
self.pub_cmdVel = self.create_publisher(Twist, '/cmd_vel', 10)
# Create a publisher to publish the center coordinates and radius of the tracked
object
self.pub_position = self.create_publisher(Position, "/Current_point", 10)
# Define the controller node data subscriber
self.sub_JoyState = self.create_subscription(Bool,"/JoyState",
self.JoyStateCallback, 1)
...
# Initialize the face detection classifier
xml_path =
os.path.join(get_package_share_directory('yahboomcar_astra'),'config','haarcasca
de_frontalface_default.xml')
self.face_patterns = cv.CascadeClassifier(xml_path)
...
# face detection
faces = self.face_patterns.detectMultiScale(frame, scaleFactor=1.2,
minNeighbors=10, minSize=(50, 50))
if len(faces) == 1:
    for (x, y, w, h) in faces:
        # Calculate the face center coordinates and area size
        self.Center_x = x + w // 2
        self.Center_y = y + h // 2
        self.Center_r = w * h // 100
        cv.rectangle(frame, (x, y), (x + w, y + h), (0, 255, 0), 2)
    # Issue control instructions
    self.execute(self.Center_x, self.Center_y, self.Center_r)
...

# According to the x value, y value and area size, use the PID algorithm to
calculate the car speed and servo angle
```

```python
def execute(self, x, y, z=None):
    position = Position()
    position.anglex = x * 1.0
    position.angley = y * 1.0
    position.distance = z * 1.0
    self.pub_position.publish(position)
    ...
    # Calculate PID control quantity
    [linear_Pid, Servox_Pid, Servoy_Pid] = self.PID_controller.update([z -
self.mintra_r, x - 320, y - 240])
    angular_Pid = self.angular_PID_controller.update([self.PWMServo_X -
self.init_servos1])[0]
    ...
    # Limit output range
    linear_Pid = max(-0.8, min(linear_Pid, 0.8))
    angular_Pid = max(-3.0, min(angular_Pid, 3.0))
    ...
    # Limit the servo PID polarity and maximum value
    Servox_Pid = Servox_Pid * (abs(Servox_Pid) <=self.Servo_Kp/3.6)
    Servoy_Pid = Servoy_Pid * (abs(Servoy_Pid) <=self.Servo_Kp/3.6)
    ...
    # Set the car speed and servo angle
    self.twist.linear.x = -linear_Pid
    self.twist.angular.z = -angular_Pid if self.img_flip else angular_Pid
    self.PWMServo_X += Servox_Pid if not self.img_flip else -Servox_Pid
    self.PWMServo_Y += Servoy_Pid if not self.img_flip else -Servoy_Pid
    ...
    # Issue control instructions
    self.pub_Servo.publish(self.servo_angle)
    self.pub_cmdVel.publish(self.twist)
```