Object recognition

Note: The virtual machine and ROS wifi image transmission module need to be set to 20.

1、Program Function Description

After the program starts, select the object you want to recognize with the mouse box, and then release the mouse to frame the recognized content.

2、Operation steps

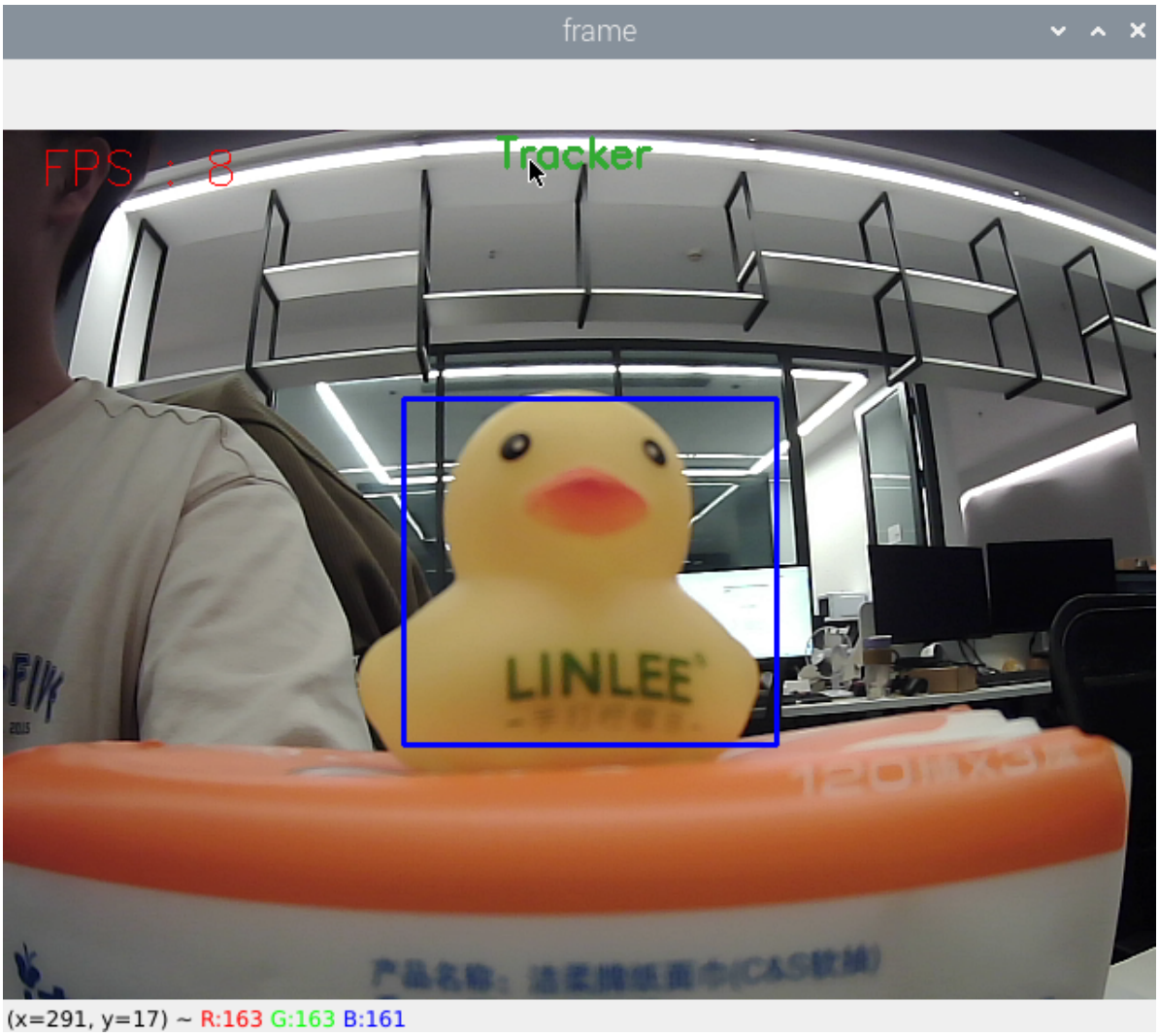
2.1、start

```
ros2 run yahboom_esp32ai_car mono_Tracker
```

**If the image of the camera is inverted, you need to refer to the * * 3. Camera Image Change * * document and correct it yourself. This experiment will not be elaborated on further.

2.2、recognition

After startup, enter the selection mode and use the mouse to select the location of the target, as shown in the following figure. Release to start recognition.



Keyboard key control:
【r】Select the mode and use the mouse to select the area to recognize the target, as shown in the above figure.
【q】: Exit the program.

2、Core code

```
 def process(self, rgb_img, action):
        rgb_img = cv.resize(rgb_img, (640, 480))
        binary = []
        if self.img_flip == True: rgb_img = cv.flip(rgb_img, 1)
        if action == 32: self.Track_state = 'tracking'
        elif action == ord('i') or action == 105: self.Track_state = "identify"
        elif action == ord('r') or action == 114: self.Reset()
        elif action == ord('q') or action == 113: self.cancel()
        if self.Track_state == 'init':
            cv.namedWindow(self.windows_name, cv.WINDOW_AUTOSIZE)
            cv.setMouseCallback(self.windows_name, self.onMouse, 0)
            if self.select_flags == True:
                cv.line(rgb_img, self.cols, self.rows, (255, 0, 0), 2)
                cv.rectangle(rgb_img, self.cols, self.rows, (0, 255, 0), 2)
                if self.Roi_init[0] != self.Roi_init[2] and self.Roi_init[1] !=
self.Roi_init[3]:
                    if self.tracker_type == "color": rgb_img, self.hsv_range =
self.color.Roi_hsv(rgb_img, self.Roi_init)
                    self.gTracker_state = True
                    self.dyn_update = True
                else: self.Track_state = 'init'
        if self.Track_state != 'init':
            if self.tracker_type == "color" and len(self.hsv_range) != 0:
                rgb_img, binary, self.circle = self.color.object_follow(rgb_img,
self.hsv_range)
                if self.dyn_update == True:
                    params = {'Hmin': self.hsv_range[0][0], 'Hmax':
self.hsv_range[1][0],
                              'Smin': self.hsv_range[0][1], 'Smax':
self.hsv_range[1][1],
                              'Vmin': self.hsv_range[0][2], 'Vmax':
self.hsv_range[1][2]}
                    self.dyn_client.update_configuration(params)
                    self.dyn_update = False
            if self.tracker_type != "color":
                if self.gTracker_state == True:
                    Roi = (self.Roi_init[0], self.Roi_init[1], self.Roi_init[2]
- self.Roi_init[0], self.Roi_init[3] - self.Roi_init[1])
                    self.gTracker = Tracker(tracker_type=self.tracker_type)
                    self.gTracker.initWorking(rgb_img, Roi)
                    self.gTracker_state = False
                rgb_img, (targBegin_x, targBegin_y), (targEnd_x, targEnd_y) =
self.gTracker.track(rgb_img)
                center_x = targEnd_x / 2 + targBegin_x / 2
                center_y = targEnd_y / 2 + targBegin_y / 2
                width = targEnd_x - targBegin_x
```

```
                high = targEnd_y - targBegin_y
                self.point_pose = (center_x, center_y, min(width, high))
        if self.Track_state == 'tracking':
            if self.circle[2] != 0: threading.Thread(target=self.execute, args=
(self.circle[0], self.circle[1])).start()
            if self.point_pose[0] != 0 and self.point_pose[1] != 0:
threading.Thread(target=self.execute, args=(self.point_pose[0],
self.point_pose[1])).start()
        if self.tracker_type != "color": cv.putText(rgb_img, " Tracker", (260,
20), cv.FONT_HERSHEY_SIMPLEX, 0.75, (50, 170, 50), 2)
        return rgb_img, binary

 def handleTopic(self, msg):
        self.last_stamp = msg.header.stamp
        if self.last_stamp:
            total_secs = Time(nanoseconds=self.last_stamp.nanosec,
seconds=self.last_stamp.sec).nanoseconds
            delta = datetime.timedelta(seconds=total_secs * 1e-9)
            seconds = delta.total_seconds()*100

            if self.new_seconds != 0:
                self.fps_seconds = seconds - self.new_seconds

            self.new_seconds = seconds

        self.get_param()
        start = time.time()
        frame = self.bridge.compressed_imgmsg_to_cv2(msg)
        frame = cv.resize(frame, (640, 480))

        action = cv.waitKey(10) & 0xFF
        frame, binary =self.process(frame, action)


        end = time.time()
        fps = 1 / ((end - start)+self.fps_seconds)

        text = "FPS : " + str(int(fps))
        cv.putText(frame, text, (10,20), cv.FONT_HERSHEY_SIMPLEX, 0.8,
(0,255,255), 2)
        cv.imshow('frame', frame)

        if action == ord('q') or action == 113:
            cv.destroyAllWindows()
```

The source code is to obtain the camera image of the ROS wifi image transmission module and use OpenCV for object recognition processing.