6. Visual tracking

6.1. Experimental Objectives

By processing the camera image, visual line patrol is realized. The line patrol principle is as follows:

(1) OpenCV coordinate system

row == heigh == Point.y

col == width == Point.x

Mat::at(Point(x, y)) == Mat::at(y,x)

Get the image:

ret, frame = image.read()

Image resize

frame = cv2.resize(frame,(320,240))

(2) Perspective transformation

Because the image seen by the camera is distorted, we need to transform the area in front of the car body into a top view through perspective transformation, and then zoom in to the entire image area. As shown in the figure below, the left side is the original camera image, and the right side is the image after perspective transformation.

Function method:

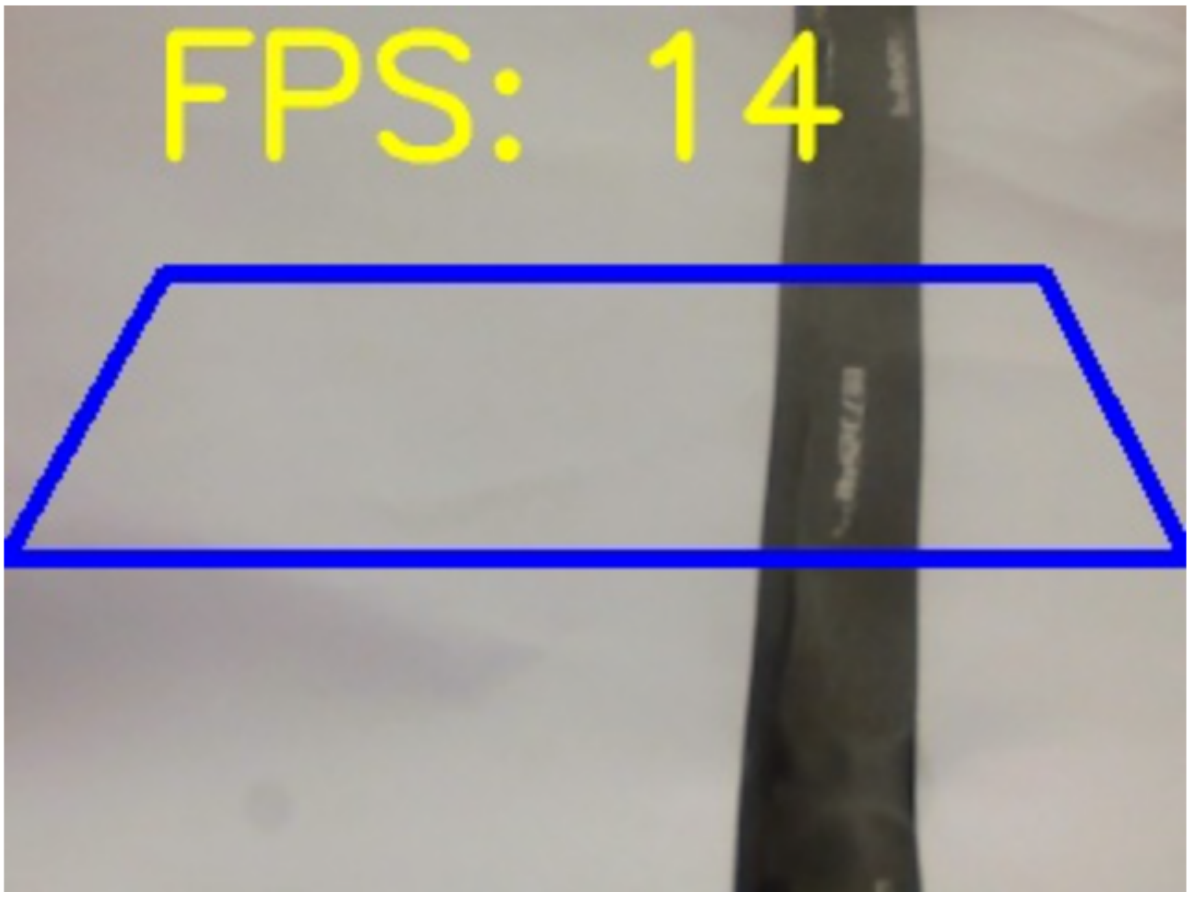
```
matSrc = np.float32([[0, 149],  [310, 149], [271, 72], [43, 72]])
```

```
matDst = np.float32([[0,240], [310,240], [310,0], [0,0]])
matAffine = cv2.getPerspectiveTransform(matSrc,matDst)# mat 1 src 2 dst
dst = cv2.warpPerspective(frame,matAffine,(320,240))
```

Then we draw a rectangular frame based on the perspective coordinates: as shown in the left picture below:

#Draw a rectangular frame

```
pts = np.array([[0, 149],  [310, 149], [271, 72], [43, 72]], np.int32)
pts = pts.reshape((-1, 1, 2))
cv2.polylines(frame, [pts],True, (255, 0, 0), 3)
```



(3) Binarization of images

The core idea of binarization is to set a threshold. Values greater than the threshold are 0 (black) or 255 (white), making the image a black and white image. The threshold can be fixed or adaptive.

Python-OpenCV provides a threshold function:

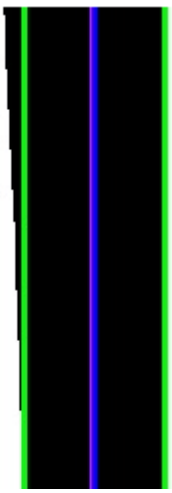
cv2.threshold (src, threshold, maxValue, method)

eg:

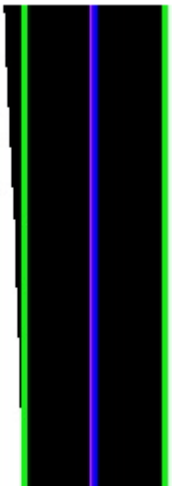
dst_gray = cv2.cvtColor(dst, cv2.COLOR_RGB2GRAY)

dst_retval, dst_binaryzation = cv2.threshold(dst_gray, 110, 255, cv2.THRESH_BINARY)

The above perspective transformed image is transformed into the following image (note that text and lines are temporarily ignored, which will be described in detail below):



(4) Get the edge of the black line and draw the line



The purpose of drawing the line is to facilitate developers to detect whether there is a problem in the above processing process. It is necessary to draw the positions of the two edges of the black line, as well as the position of the image center line and the center line of the two edges of the black line.

The image size after perspective is still 320*240, so the image centerline is: midpoint: 320/2-1=159

Or: midpoint = np.int(histogram.shape[0]/2)

The method for finding the left side of the black line: take the minimum index of the left image, the implementation method is as follows:

leftx_base = np.argmax(histogram[:midpoint], axis = 0)

The method for finding the right side of the black line: after reversing the image, take the minimum index of the left side of the image, and then subtract the index from the image width:

rightx_base = np.argmin(histogram[::-1][:rightpoint], axis = 0) #Reverse the histogram and take the rightmost value

rightx_base = 319 - rightx_base

Actual midline position calculation: (left line + right line)/2

lane_center = int((leftx_base + rightx_base)/2) #The middle position of the left and right lines

Image center line: purple, cv2.line(dst_binaryzation, (159,0), (159,240), (255,0,255), 2) #Purple

Left measurement line: cv2.line(dst_binaryzation, (leftx_base, 0), (leftx_base, 240), (0,255,0), 2) #Green

Right line: cv2.line(dst_binaryzation, (rightx_base, 0), (rightx_base, 240), (0,255,0), 2) #Green

Actual center line: cv2.line(dst_binaryzation, (lane_center, 0), (lane_center, 240), (255,0,0), 2) # Blue

(5) Calculate offset

After finding the black line of autonomous driving in step (4), we can know the deviation of the vehicle body based on the difference between the actual center line and the center line of the image, so as to control the car to correct the deviation and achieve the purpose of autonomous driving.

Calculation method:

Offset value = image center line - actual center line

Bias = 159 - lane_center

According to the positive and negative and size of the offset value, we limit it to between -25 and 25, which means that the maximum steering speed is 25 to prevent the white line of the track from being lost due to excessive steering.

Steering angle PID adjustment:

```
Z_axis_pid.SystemOutput = Bias

Z_axis_pid.SetStepSignal(0)

Z_axis_pid.SetInertiaTime(0.5, 0.2)
```

After PID calculation, the PID output is also limited to a maximum of ±20.

```
if Z_axis_pid.SystemOutput > 25:

Z_axis_pid.SystemOutput = 25

elif Z_axis_pid.SystemOutput < -25:

Z_axis_pid.SystemOutput = -25
```

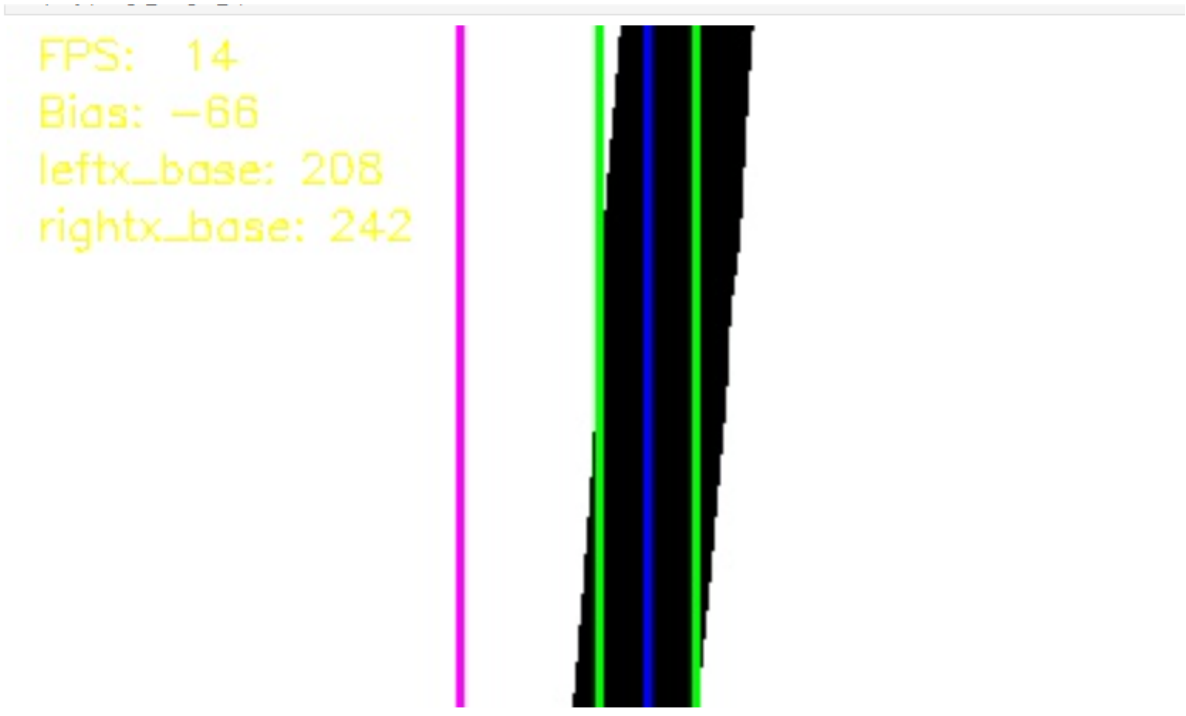
(6) Control the robot to complete automatic driving

According to the value of Bias, the absolute value is within 3: we think the body offset is not large, and we can directly control the car to turn left and right.

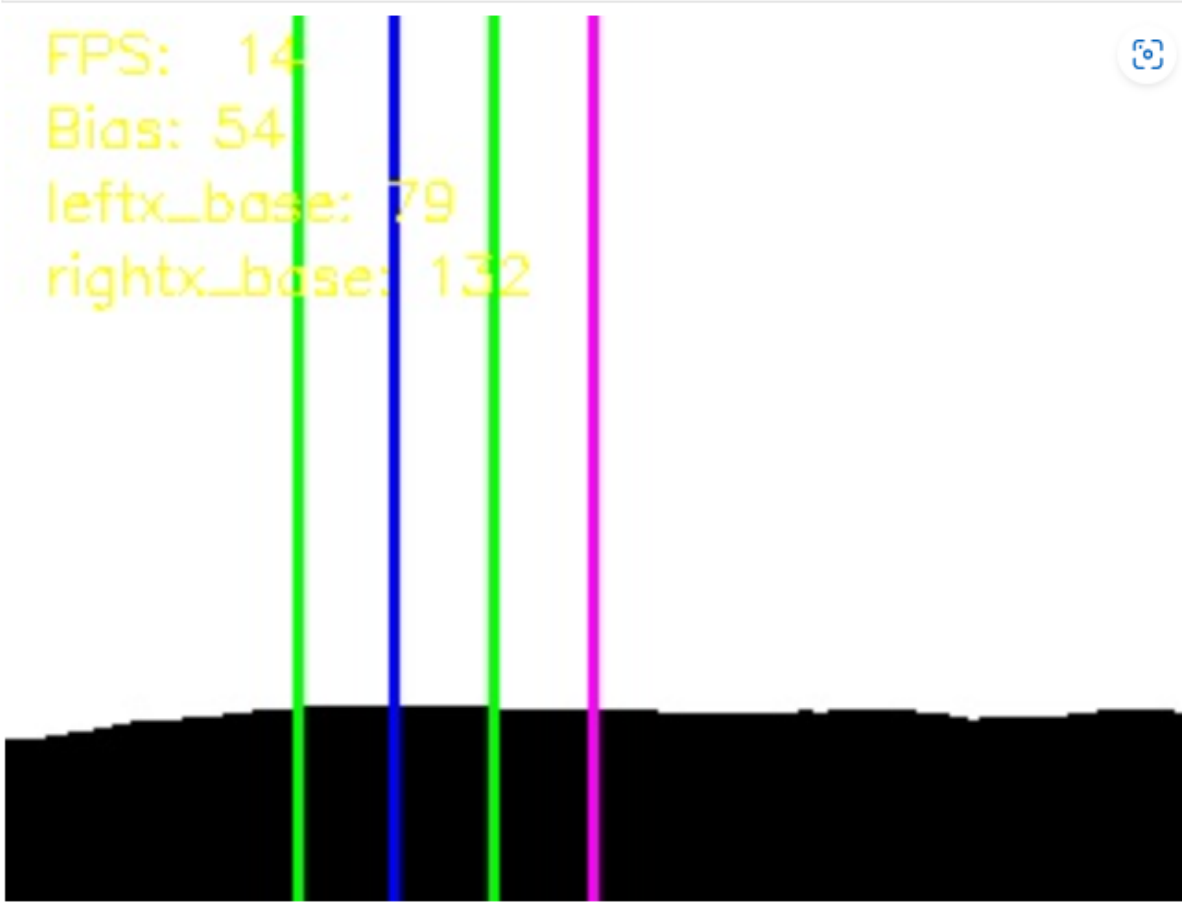
The absolute value of Bias is greater than 140: At this time, the robot offset is large and is about to run out of the black line.

The left line is on the far left, and the right line is on the far right. This situation usually occurs when the right angle curve is entered. The center line deviates a lot to turn at the moment of entering the right angle. However, since the line segment is relatively thin, if this picture is lost, the car will go over the limit, resulting in the following pictures being all white. Therefore, in order to avoid this situation, it is necessary to make a difference between the leftmost and rightmost pixel points in advance when seeing the right angle curve to determine whether it is a left right angle or a right right angle.

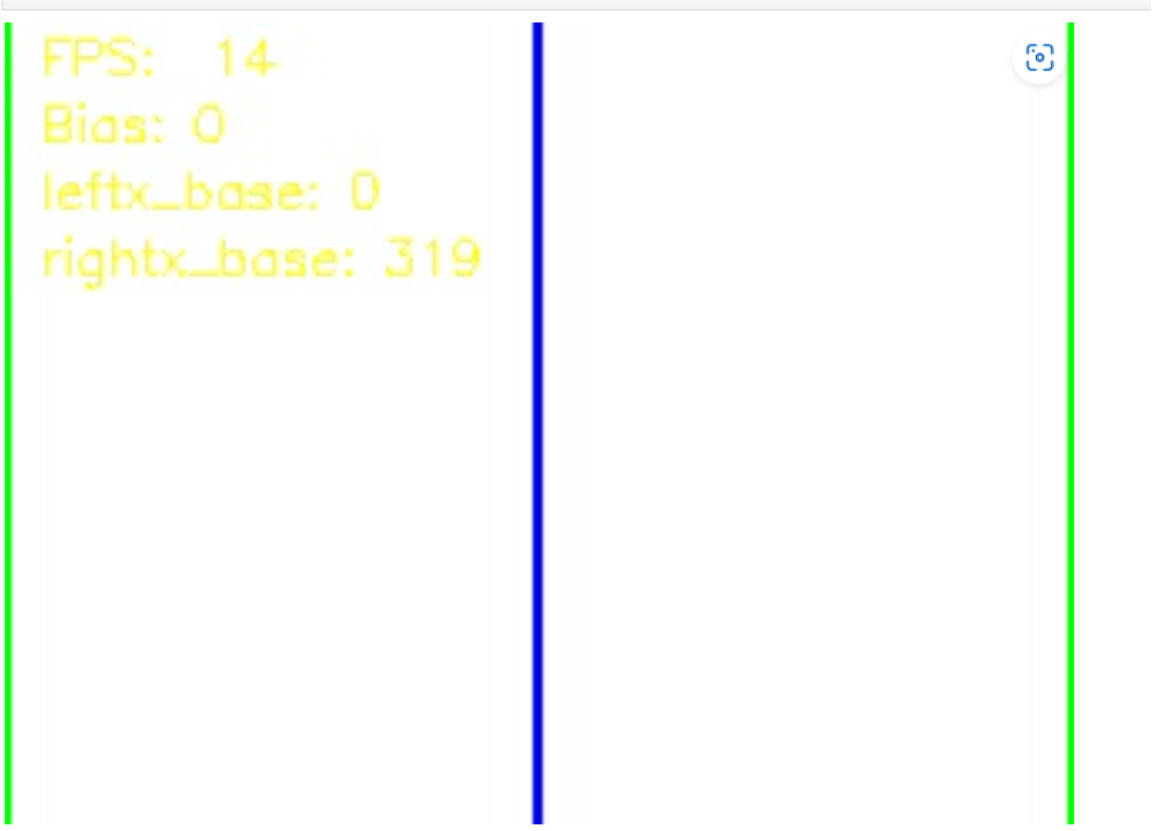
The picture above shows a slight deviation, in this case it goes straight.



The above picture shows a large right deviation that is corrected by turning right.



The above picture shows the deviation caused by a normal left right angle, which is corrected by rotation.



The above picture shows that the vehicle may have gone too far due to the loss of the image when passing the right angle, and turned in the direction determined before entering the right angle.

6.2. Implementation code

Source code path:

/home/pi/project_demo/08.AI_Visual_Interaction_Course/06.Vision_Based_Auto_LineFollowing/06_Vision_Based_Auto_LineFollowing.ipynb

```
  if leftx_base == 0 and rightx_base == 319:
                if prev_left > prev_right:
                    #None
                    rotate_left(speed)
                elif prev_left < prev_right:
                    #None
                    rotate_right(speed)

                prev_left = 0
                prev_right = 0

            else: #根据线粗调节Bias值 Adjust the Bias value according to the line thickness
                if Bias > 35:   #黑线在中线的左边，小车左转（左边速度减小，右边速度增大）The black line is to the left of the center line, and the car turns left (the speed on the left decreases, and the speed on the right increases)
                    #prev_left = 1
                    #prev_right = 0
                    if Bias > 140:
                        rotate_left(int((speed-Z_axis_pid.SystemOutput)/5))
                        prev_left = 0
                        prev_right = 0
                    elif(Bias < 50):
                        move_left(int((speed/15)))
                    else:move_forward(speed)
                        #rotate_left(int((speed/15)))
                    time.sleep(0.001)
                elif Bias < -35:    #黑线在中线的右边，小车右转    （左边速度增大，右边速度减小）The black line is to the right of the center line, and the car turns right (the speed increases on the left and decreases on the right)
                    #prev_right = 1
                    #prev_left = 0
                    if(Bias < -140):
                        rotate_right(int((speed+Z_axis_pid.SystemOutput)/5))
                        prev_left = 0
                        prev_right = 0
                    elif(Bias > -45):
                        move_right(int((speed/15)))
                    else:move_forward(speed)
                        #rotate_right(int((speed/15)))
                    time.sleep(0.001)

                else:
                    #None
                    move_forward(speed)
```

```
        if left_sum != right_sum:
            if left_sum < right_sum:
                prev_left = prev_left + 1
            elif right_sum < left_sum:
                prev_right = prev_right + 1
```

Since the code is too long, it is not shown here. For specific code, please refer to the source code.

6.3. Experimental phenomenon

After the code block is run, put the car on the map with white background and black lines where the line needs to be patrolled, and the car will patrol the black line. When patrolling the line, the ultrasonic detection detects an obstacle in front, the car will stop moving, and the buzzer will alarm. Remove the obstacle, and the car will continue to patrol the black line

Note: This program is suitable for simple single curves and single right-angle bends. It may not be suitable for some criss-crossing lines. Users can improve and increase the use scenarios according to their needs to achieve more complex lines.