بسمه تعالی

آتنا آزادی پور

پروژه : تشخیص خطوط جاده ها – درس پردازش تصویر

دانشکده ملی مهارت دختران دکتر شریعتی تهران – بهمن ۴۰۳

استفاده از سه کتابخانه Opencv,Numpy,Matplotlib : (pip ورژن ۲۵.۱.۱)

pip install matplotlib

pip install numpy

pip install opencv-python

```
import cv2
import numpy as np
import matplotlib.pyplot as plt
```

تصویر ورودی توسط OpenCV خوانده می شود و یک نسخه کپی از آن برای نمایش نتیجه نهایی تهیه می شود. سپس ابعاد تصویر (ارتفاع و عرض) استخراج می شود تا در مراحل بعدی برای تعریف ناحیه مورد نظر (ROI) استفاده شود:

```
def detect_road_lines(image_path):
    image = cv2.imread(image_path)
    result_image = np.copy(image)
    height, width = image.shape[:2]
```

تصویر اصلی به دو فضای رنگی مختلف تبدیل می‌شود: خاکستری و HSV استفاده از این دو فضای رنگی به خاطر استخراج ویژگی های متنوع برای شناسایی خطوط جاده، خصوصا خطوط سفید رنگ، ضروری است. فضای خاکستری برای آستانه گذاری ساده و HSV برای ایجاد یک ماسک رنگی دقیق تر استفاده می شود:

```
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    hsv = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)
```

در فضای خاکستری، با استفاده از آستانه گذاری، نواحی روشن به صورت یک ماسک باینری استخراج می شوند و در فضای HSV هم محدوده ای از رنگ سفید تعریف شده و ماسکی برای آن تولید می شود. در نهایت این دو ماسک با عملگر `bitwise_or` ترکیب می شوند تا یک ماسک نهایی و کاملتر برای نواحی سفید تولید شود:

```
    _, gray_thresh = cv2.threshold(gray, 200, 255, cv2.THRESH_BINARY)
    lower_white_hsv = np.array([0, 0, 200])
    upper_white_hsv = np.array([180, 30, 255])
    hsv_white_mask = cv2.inRange(hsv, lower_white_hsv, upper_white_hsv)
    combined_mask = cv2.bitwise_or(gray_thresh, hsv_white_mask)
```

برای جلوگیری از بررسی کل تصویر و تمرکز فقط بر روی نواحی پایین تصویر که معمولا محل قرارگیری خطوط جاده است، یک ناحیه ذوزنقه ای به عنوان ROI تعریف می شود. این ناحیه توسط مختصات چهار نقطه مشخص شده و سپس به کمک fillPoly به صورت یک ماسک مجزا ایجاد می شود. با ضرب بیتی این ماسک با ماسک قبلی، تنها بخش‌هایی از تصویر باقی می ماند که هم سفید هستند و هم در ناحیه مورد نظر قرار دارند:

```
    roi_vertices = np.array([
        [(0, height),
         (width * 0.35, height * 0.65),
         (width * 0.60, height * 0.65),
         (width, height)]
    ], dtype=np.int32)
    roi_mask = np.zeros_like(combined_mask)
    cv2.fillPoly(roi_mask, roi_vertices, 255)
    masked_image = cv2.bitwise_and(combined_mask, roi_mask)
```

با هدف کاهش نویز و افزایش پیوستگی خطوط، عملیات MorphologyEx انجام می شود. ابتدا با استفاده از MORPH_OPEN نویزهای کوچک حذف می شود، و سپس با MORPH_CLOSE شکاف های موجود در خطوط پر می شوند تا مسیرهای ممتدتری به دست آید:

```
    kernel = np.ones((3, 3), np.uint8)
    opening = cv2.morphologyEx(masked_image, cv2.MORPH_OPEN, kernel, iterations=1)
    closing = cv2.morphologyEx(opening, cv2.MORPH_CLOSE, kernel, iterations=2)
```

با استفاده از تابع findContours ، کانتورها یا همان مرز های اجسام سفید شناسایی شده در تصویر به دست می آیند. سپس هر کانتور از نظر مساحت و شکل مورد بررسی قرار می‌گیرد. کانتور هایی که مساحت کافی نداشته باشند نادیده گرفته می شوند. علاوه بر آن، ویژگی گردی (circularity) نیز برای حذف اشکال دایره ای یا گرد استفاده می شود، چون که خطوط جاده دایره ای شکل نیستند:

```
    contours, _ = cv2.findContours(closing, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
    min_area = 20
    road_marking_contours = []
    for contour in contours:
        area = cv2.contourArea(contour)
        if area > min_area:
            perimeter = cv2.arcLength(contour, True)
            if perimeter > 0:
                circularity = 4 * np.pi * area / (perimeter * perimeter)
                # نشانه‌های جاده معمولا کشیده هستند (دایره‌ای نیستند)
                if circularity < 0.6:
                    road_marking_contours.append(contour)
```

و در آخر کانتورهایی که معیارهای مورد نظر را دارند، روی تصویر اصلی با رنگ سبز رسم می شوند و برای ظاهری بهتر در خروجی تصویر به فضای رنگی RGB تبدیل شده و در کنار تصویر اصلی توسط کتابخانه Matplotlib به کاربر نمایش داده می شود:

```
    cv2.drawContours(result_image, road_marking_contours, -1, (0, 255, 0), 2)
    result_rgb = cv2.cvtColor(result_image, cv2.COLOR_BGR2RGB)
    return result_rgb

def show_results(original_path, processed_image):
    original = cv2.imread(original_path)
    original_rgb = cv2.cvtColor(original, cv2.COLOR_BGR2RGB)
    plt.figure(figsize=(14, 7))
    plt.subplot(1, 2, 1)
    plt.imshow(original_rgb)
    plt.title('original', fontsize=14,)
```

```
    plt.axis('off')
    plt.subplot(1, 2, 2)
    plt.imshow(processed_image)
    plt.title('Road Lane Line Detection', fontsize=14)
    plt.axis('off')

    plt.tight_layout()
    plt.show()
```

در آخر مراحل میانی پردازش تصویر مثل ماسک ها، ناحیه مورد نظر و تصویر نهایی پس از فیلتر MorphologyEx قابل نمایش هستند. این تابع به منظور بررسی دقیق تر عملکرد فیلترها و ماسک ها طراحی شده و برای رفع اشکال مفید است:

```
def show_processing_steps(image_path):
    image = cv2.imread(image_path)
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    hsv = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)
    _, gray_thresh = cv2.threshold(gray, 200, 255, cv2.THRESH_BINARY)
    lower_white_hsv = np.array([0, 0, 200])
    upper_white_hsv = np.array([180, 30, 255])
    hsv_white_mask = cv2.inRange(hsv, lower_white_hsv, upper_white_hsv)
    combined_mask = cv2.bitwise_or(gray_thresh, hsv_white_mask)
    height, width = image.shape[:2]
    roi_vertices = np.array([
        [(0, height),
         (width * 0.55, height * 0.85),
         (width * 0.85, height * 0.85),
         (width, height)]
    ], dtype=np.int32)
    roi_mask = np.zeros_like(combined_mask)
    cv2.fillPoly(roi_mask, roi_vertices, 255)
    masked_image = cv2.bitwise_and(combined_mask, roi_mask)
    kernel = np.ones((3, 3), np.uint8)
    opening = cv2.morphologyEx(masked_image, cv2.MORPH_OPEN, kernel, iterations=1)
    closing = cv2.morphologyEx(opening, cv2.MORPH_CLOSE, kernel, iterations=2)
    plt.tight_layout()
    plt.show()

if __name__ == "__main__":
    image_path = "6.jpg"
    result = detect_road_lines(image_path)
    if result is not None:
        show_results(image_path, result)
```

آدرس گیت هاب :

https://github.com/atnaazdi/Road-Lane-Line-Detection.git