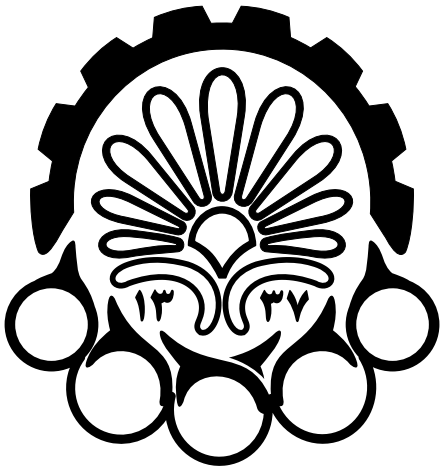


دانشگاه صنعتی امیرکبیر

(پلی تکنیک تهران)

دانشکده مهندسی کامپیوتر

تمر ین دوم درس بینایی ماشین

دکتر صفابخش

غلامرضا دار ۴۰۰۱۳۱۰۱۸

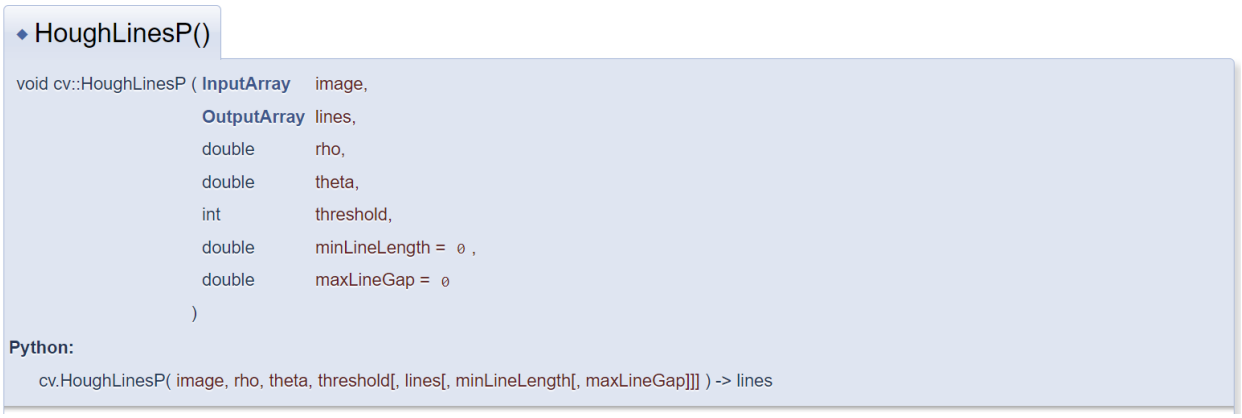
بهار ۱۴۰۱

فهرست مطالب

سوال ۱)

با توجه به شکل ۱ که از وبسایت رسمی OpenCV آورده شده است، این تابع پارامترهای زیر را به عنوان ورودی می‌گیرد.

شماره	نام پارامتر	توضیح
۱	image	تصویر ورودی جهت یافتن خطوط
۲	rho	رزولوشن مکانی جدول A[۱]
۳	theta	رزولوشن زاویه ای جدول A
۴	threshold	حداقل مقدار مورد نیاز در جدول A جهت تایید شدن یک خط
۵	minLineLength	حداقل طول خط جهت تایید شدن
۶	maxLineGap	حداکثر فاصله خالی بر روی یک خط جهت تایید شدن



پارامترهای تابع HoughLinesP

تفاوت این تابع با تابعی که در درس مورد بحث واقع شد، این است که تابع HoughLinesP نسخه احتمالاتی این تابع است و برخلاف تابع درس که به ازای هر خط θ, ρ را خروجی می‌داد، این تابع موقعیت دکارتی نقاط دو سر خط یعنی x_1, y_1, x_2, y_2 را خروجی می‌دهد.

[۱] طبق اسلایدهای درس، جدولی که به عنوان شمارنده خطوط عمل میکند.

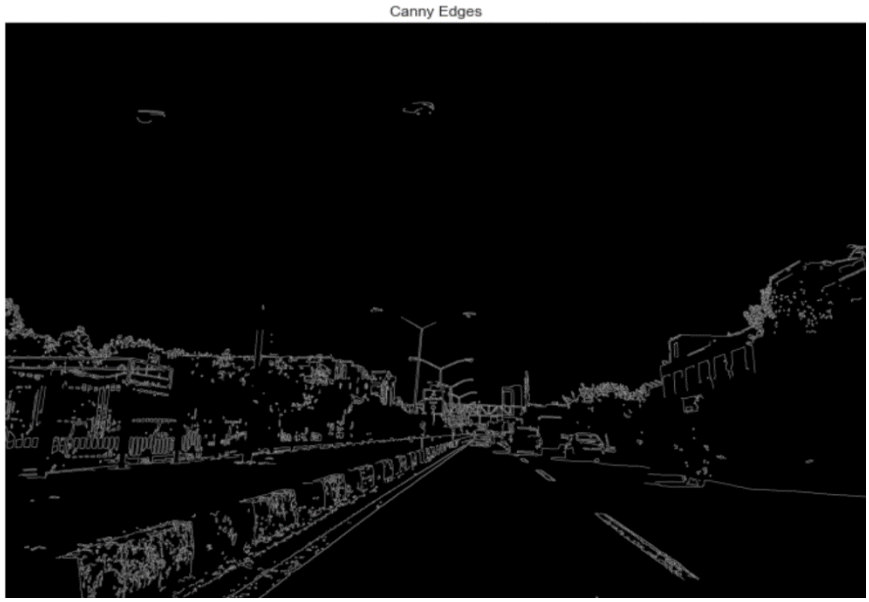
سوال ۲)

یافتن خطوط جاده(Lanes) طی چندین مرحله پی در پی انجام شد. در اکثر مرحله ها، با استفاده از قابلیت ویجت‌های Jupyter، مقدار پارامترهای مناسب پیدا شد. در ادامه این مراحل را به صورت تصویری بررسی خواهیم کرد.

مرحله اول – خواندن تصویر) خواندن تصویر

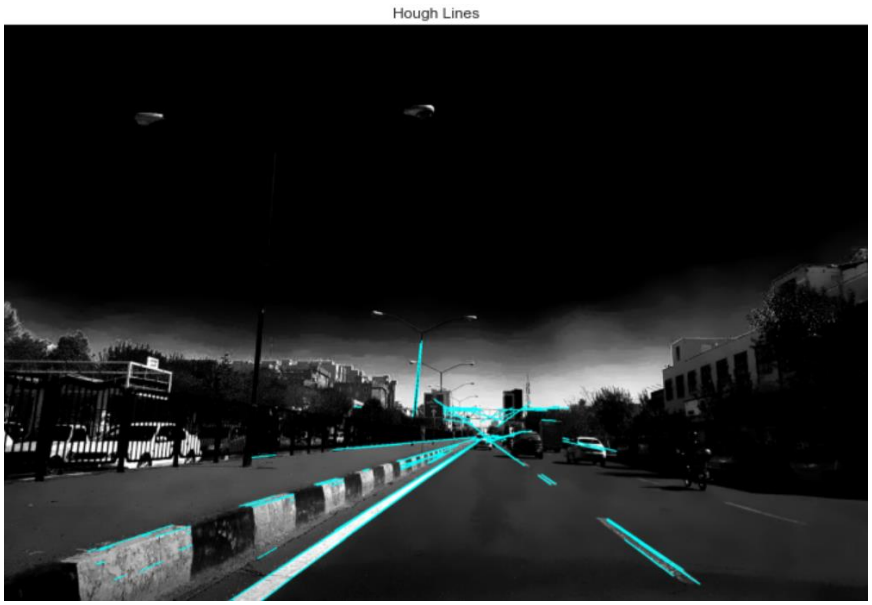
مرحله دوم – بهبود تصویر برای انجام مراحل بعد) اعمال عملیات مقدماتی رنگی بر روی تصویر

مرحله سوم – لبه یابی) به کمک الگوریتم Canny لبه‌های تصویر پیدا شدند.



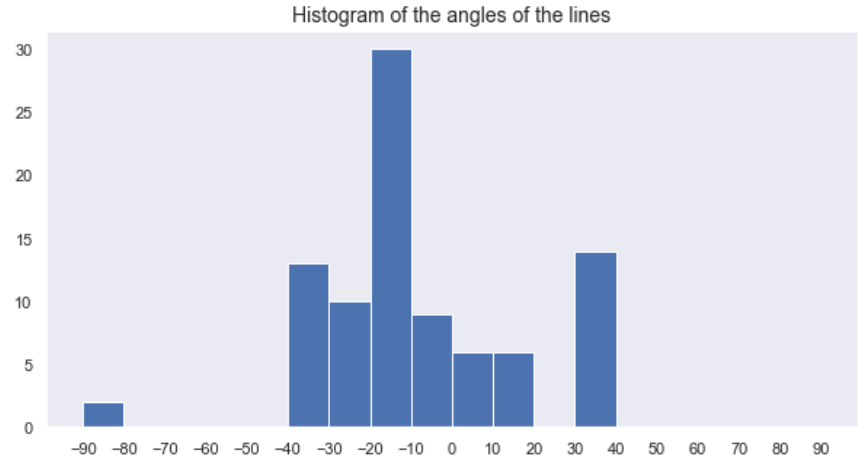
مرحله چهارم – ماسک کردن ناحیه پایین تصویر) انتخاب ناحیه مثلثی از پایین تصویر

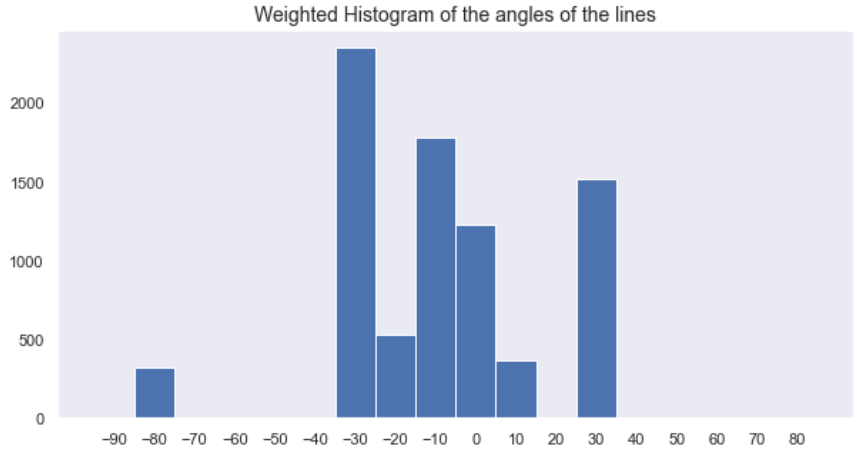
مرحله پنجم – خواندن تصویر) یافتن پاره‌خط های موجود در تصویر به کمک HoughLinesP



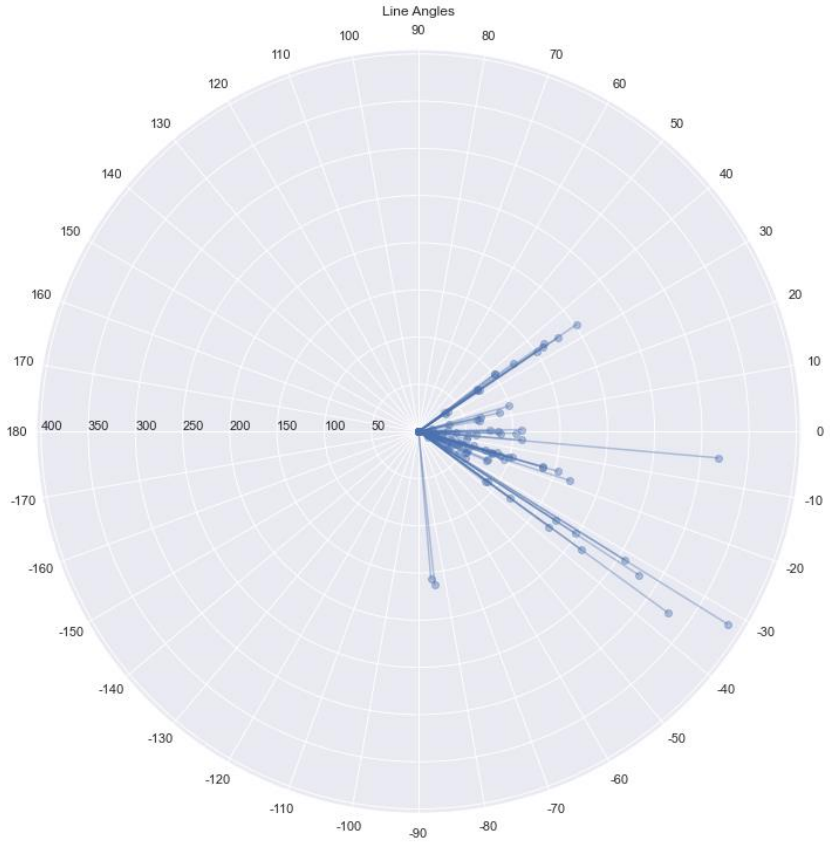
مرحله ششم – تحلیل خطوط از روی زاویه)

در این بخش می‌خواهیم با کمک اطلاعات موجود از خطوط به دست آمده(طول و زاویه آنها) خطوط موجود را به دسته های مختلف طبقه بندی کنیم. ابتدا نمودار تعداد خطوط با زوایای مختلف را رسم میکنیم. این نمودار اطلاعات مفیدی به ما نمی‌دهد زیرا اهمیت اندازه خطوط را در نظر نمی‌گیرد. در این نمودار یک خط بسیار کوچک و یک خط بسیار بزرگ یک میزان اهمیت دارند.

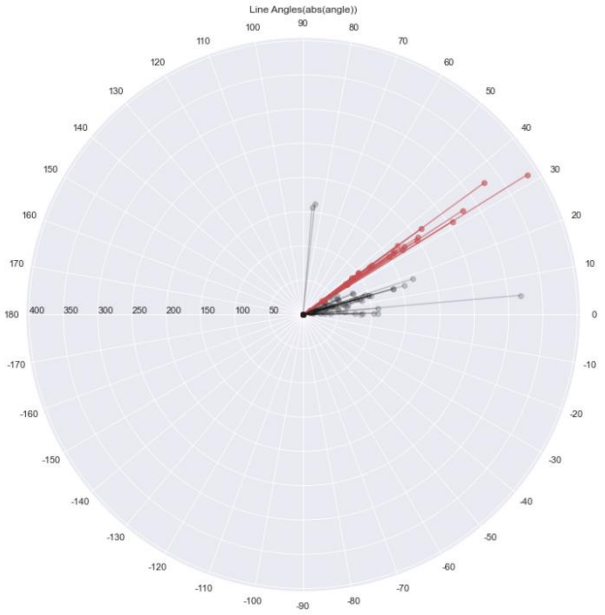


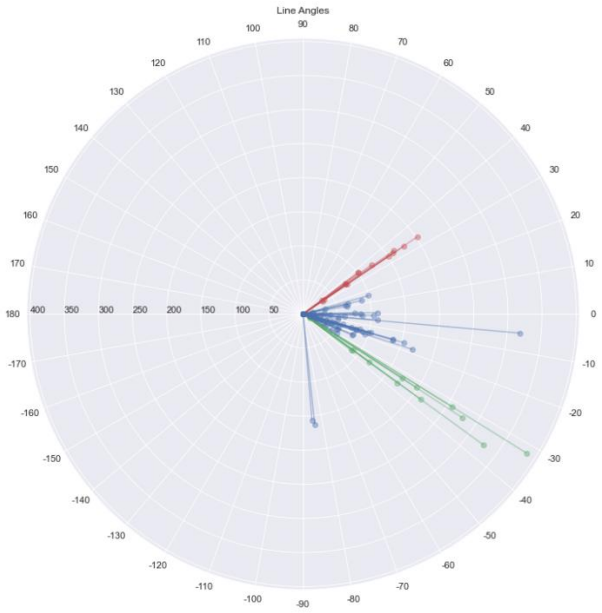


برای رفع این مشکل، ارتفاع barها در این نمودار را برابر با مجموع طول خطوط با آن زاویه قرار می‌دهیم. در این نمودار مشاهده می‌شود که تعداد زیادی از خطوط زاویه ای حدود ۳۰ و تعداد زیاد دیگری زاویه حدود ۳۰ـ دارند. با توجه به دانشی که از مسئله داریم می‌دانیم که این زوایا برای خطوط دو طرف جاده معقول هستند.



راه بهتری که برای نمایش اطلاعات گفته شده وجود دارد نمودار قطبی بالا است. در این نمودار هر پاره خط از مبدا، معادل یک پاره خط در تصویر است. زاویه در این نمودار برابر با زاویه خط و شعاع در این نمودار نشان دهنده‌ی اندازه خط در تصویر است. در این نمودار نیز واضح است که تعداد زیادی از خطوط در حدود ۳۵ و ۳۵ـ درجه قرار دارند.

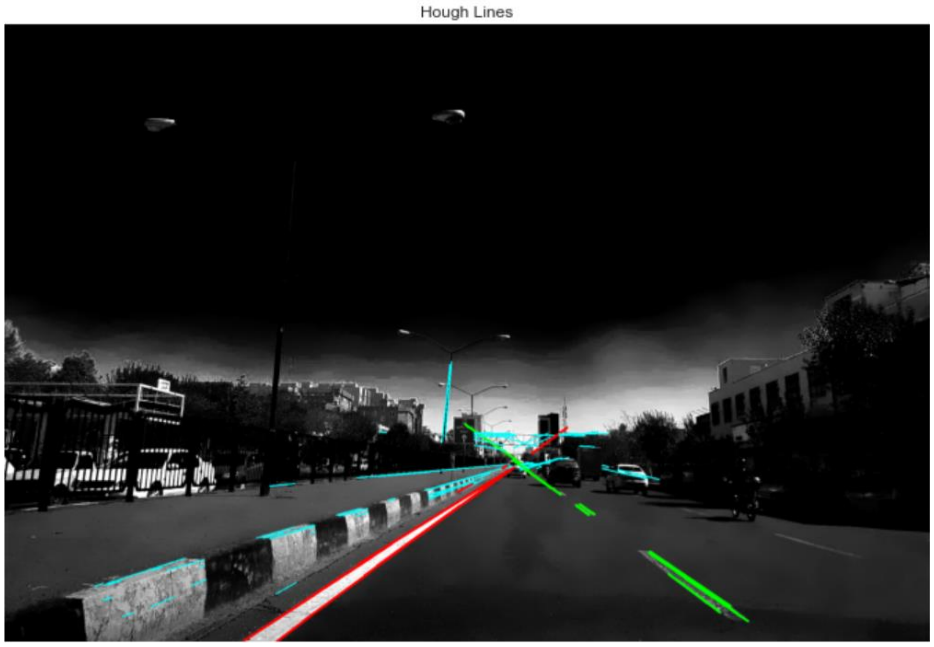




در نمودار سمت راست این خطوط را دسته بندی کرده ایم و در نمودار سمت چپ، قدر مطلق زوایا را نمایش داده ایم که در این نمودار بزرگترین قله مربوط به اندازه زاویه ۳۵ درجه است.

توجه: *دقت کنید که با وجود اینکه می‌توانیم به صورت پویا، برای هر مسئله این زوایا را محاسبه کنیم، اما با توجه به ثابت بودن این زوایا در بسیاری از شرایط، صرفا می‌توانیم با قبول کردن زوایا در بازه ۳۰ تا ۴۰ ، این خطوط را در بسیاری از مسائل به درستی طبقه بندی کنیم. عواملی که در تغییر این زوایا دست دارند عواملی نظیر استفاده از لنزهایی با فاصله کانونی بسیار کم (Wide) یا قرار گیری دوربین در ارتفاع های مختلف هستند. در نهایت منطقی است که شرکت سازنده خودرو، بهترین مقادیر را موقع تولید هر خودرو محاسبه کند.*

مرحله هفتم – دسته بندی خطوط) با استفاده از اطلاعات بدست آمده در بخش قبل خطوط را به دسته های "خطوط چپ"، "خطوط راست" و "سایر خطوط "دسته بندی می‌کنیم.



مرحله هشتم – ترکیب خطوط هر دسته) برای ترکیب خطوط هر دسته راه‌های زیادی وجود دارد. در این پیاده‌سازی، نقاط دوسر هر پاره خط استخراج شد، با استفاده از رگرسیون خطی، یک خط از بین این نقاط استخراج شده عبور داده شد. این کار برای خطوط موجود در دسته‌های **راست** و **چپ** انجام شد.

همچنین در این محله با برخورد دادن دو خط به دست آمده، نقطه محوشدگی افق(Vanishing Point) نیز محاسبه شد و خطوط از پایین تصویر تا این نقطه ترسیم شدند.

نتیجه نهایی

ماسک تهیه شده

نکته آموزشی برای آیندگان!

برای یافتن پارامترهای بهینه الگوریتم‌هایی مانند Canny یا HoughLinesP می‌توان از decorator، **@interact** از کتابخانه ipywidgets استفاده کرد.

طریقه استفاده:

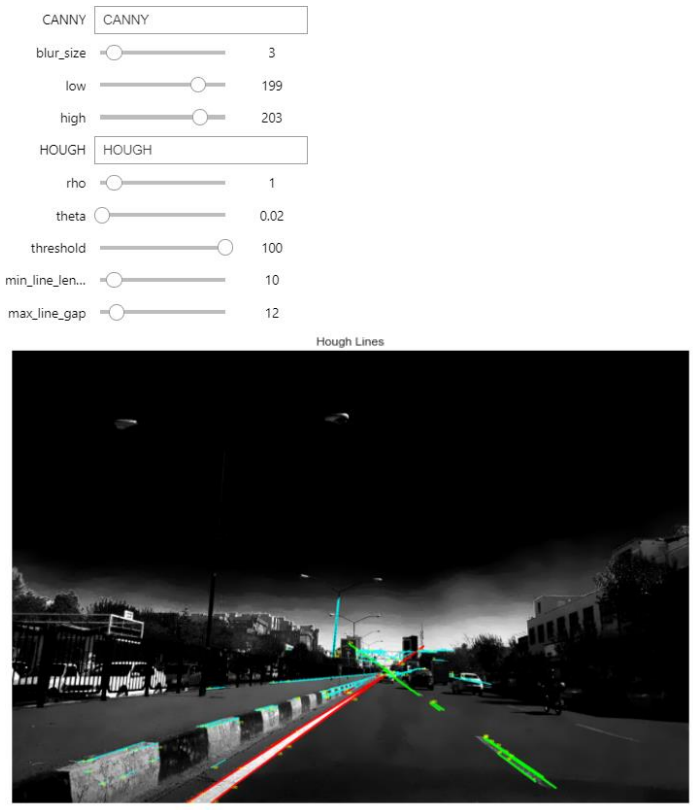
```
from ipywidgets import interact

@interact(x=(0,255,10)) # from 0 to 255, step 10
def function(x=1):
    return x
```

0.3s

x 90

90

نمونه کاربرد:



سوال ۳)

در این سوال می‌خواهیم الگوریتم استفاده شده در بخش قبل را بر روی ویدیو اعمال کنیم. به دلیل ماهیت ویدیویی بودن خروجی ها، برای مشاهده نتیجه الگوریتم به ویدیوهای پیوست شده مراجعه کنید.

تصاویری از چند فریم رندوم از ویدیو ۱ (vid_1_v1.mp4)

همان‌طور که در تصاویر می‌بینید الگوریتم سوال قبل بدون نیاز به تغییر خاصی بر روی فریم‌های مختلف این ویدیو نیز جواب خوبی می‌دهد. اما اگر به ویدیو کامل این تشخیص نگاه کنید متوجه می‌شوید که نتیجه بسیار نویزی و دارای لرزش زیادی است. دلیل این اتفاق این است که الگوریتم برای پیش بینی فقط به اطلاعات موجود در همان فریم نگاه می‌کند. اما با توجه به اینکه در این سوال با یک ویدیو روبه‌رو هستیم، می‌توانیم از اطلاعات فریم‌های قبلی نیز بهره ببریم.

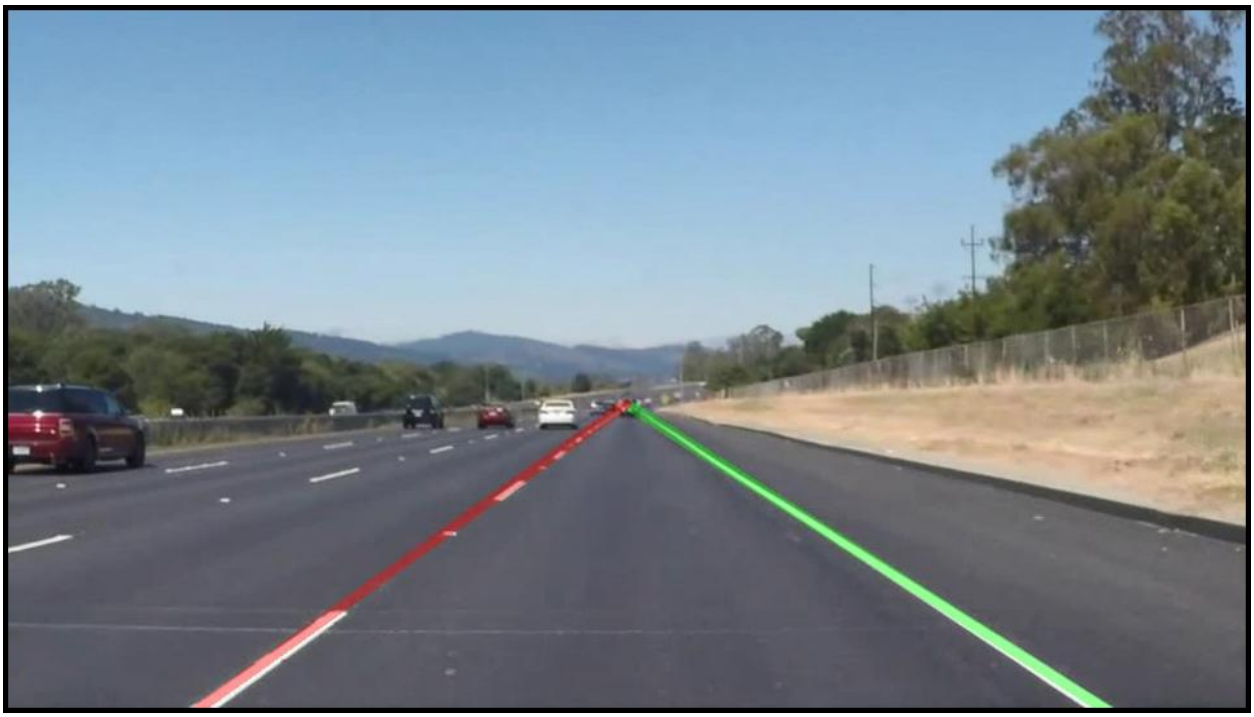
ایده اولی که مورد آزمایش قرار گرفت، استفاده از موقعیت خطوط تشخیص داده شده در چند فریم قبل برای هموارسازی موقعیت فعلی بود. به این ترتیب، پس از محاسبه موقعیت خطوط در فریم t، موقعیت Smooth شده این خطوط از رابطه زیر بدست می‌آید.

$$X'_t = (\alpha)\, X_t + (۱ - \alpha) X'_{t-۱} \qquad (۱$$

که در آن X موقعیت تشخیص داده شده خطوط توسط الگوریتم سوال قبل، X' موقعیت Smooth شده با کمک فریم های قبلی و α ضریب Smooth شدن است.

برای مشاهده نتیجه این تکنیک به ویدیو vid_1_v4_smooth_60.mp4 مراجعه کنید.

نمایی از ویدیوی vid_1_v4_smooth_60.mp4



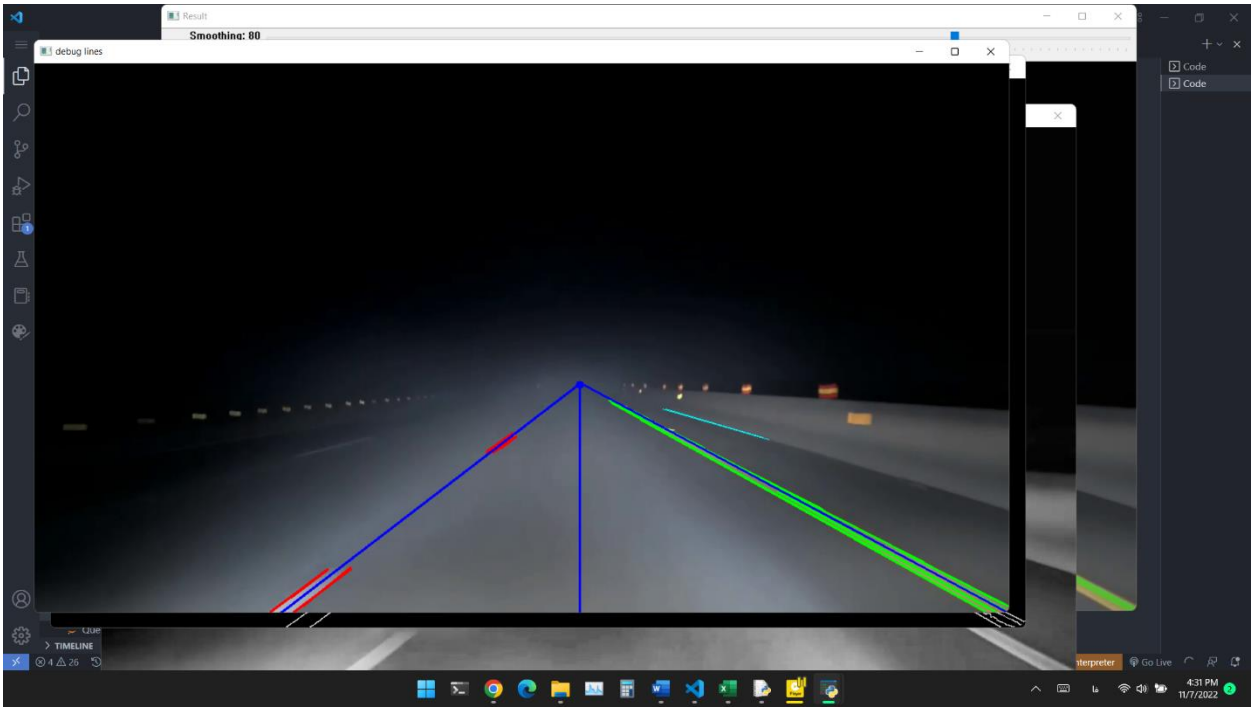
در بخش بعدی سوال سراغ **ویدیوی شماره ۲** می‌رویم. امیدواریم تکنیک‌های استفاده شده تا اینجا بر روی این ویدیو نیز نتیجه قابل قبولی بدهد.

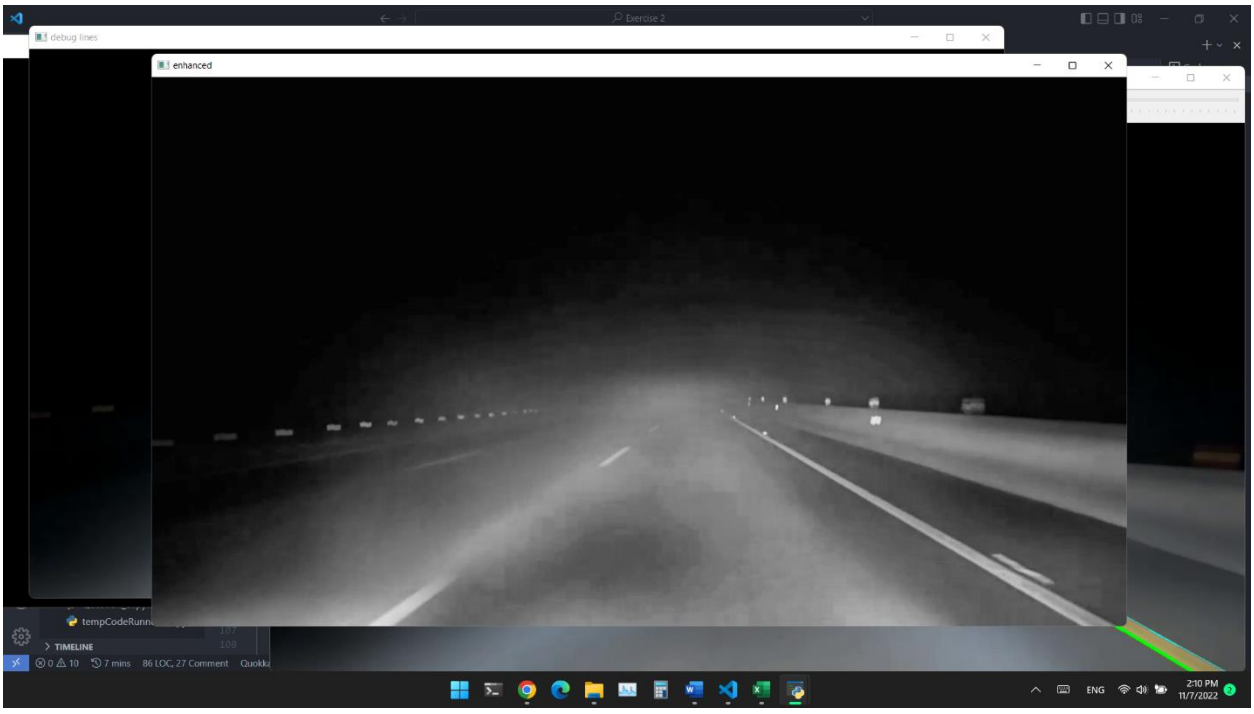
نتایج اولیه بدون تغییر اضافه ای بر روی این ویدیو بسیار خراب بود. برای بهبود نتایج تشخیص خطوط بر روی این ویدیو، نیاز شد تعداد زیادی از پارامترهای توابعی مانند Canny و HoughLinesP را تغییر دهیم. پس از تنظیم کردن پارامترها برای این ویدیو، نتیجه قابل مشاهده در ویدیوی vid_2_v5_smooth_80.mp4 بدست آمد. با مقایسه این ویدیو و ویدیوی vid_2_v5_smooth_0.mp4 می‌توان اهمیت Smoothing را مشاهده کرد. عمل Smoothing باعث می‌شود در فریم‌هایی که اطلاعات کافی برای تشخیص درست موقعیت خطوط راهنمایی وجود ندارد(وجود مه فراوان)، از اطلاعات موجود در فریم‌های قبل استفاده شود.

نمونه ای از مشکلات در صورت عدم استفاده از Smoothing

حل مشکل بالا به کمک Smoothing

تصاویری از محاسبات میانی برای حل این سوال را در این صفحه مشاهده می‌کنید.





سوال ۴)

پارامترهای تابع active_contour از کتابخانه scikit_image

شماره	نام پارامتر	توضیح
۱	image	تصویر ورودی شامل اطلاعات مورد نیاز برای ردیابی لبه
۲	snake	موقعیت ابتدایی کانتور
۳	alpha	سرعت حرکت کانتور به سمت لبه‌ها
۴	beta	ضریب نرم بودن کانتور پیدا شده
۵	w_line	وزن تاثیرگذاری خودِ مقادیر پیکسل‌ها
۶	w_edge	وزن تاثیرگذاری لبه‌ها(بر روی image الگوریتم لبه‌یابی اجرا می‌شود)
۷	gamma	پارامتر قدم زمانی
۸	max_px_move	بیشینه اندازه مجاز حرکت در هر مرحله(پیکسل)
۹	max_num_iter	بیشینه تعداد مرحله مجاز برای جستجو
۱۰	convergence	معیار همگرایی(اگر از این حد نزدیک تر شدیم کافی است)
۱۱	boundary_condition	شرایط انتهای snake(با مشخص کردن فلگ‌های مختلف تعیین می‌کنیم الگوریتم جستجو با نقاط انتهایی snake اولیه چگونه برخورد کند، به عنوان مثال در حالت fixed، الگوریتم تنها نقاط میانی را جابجا می‌کند)

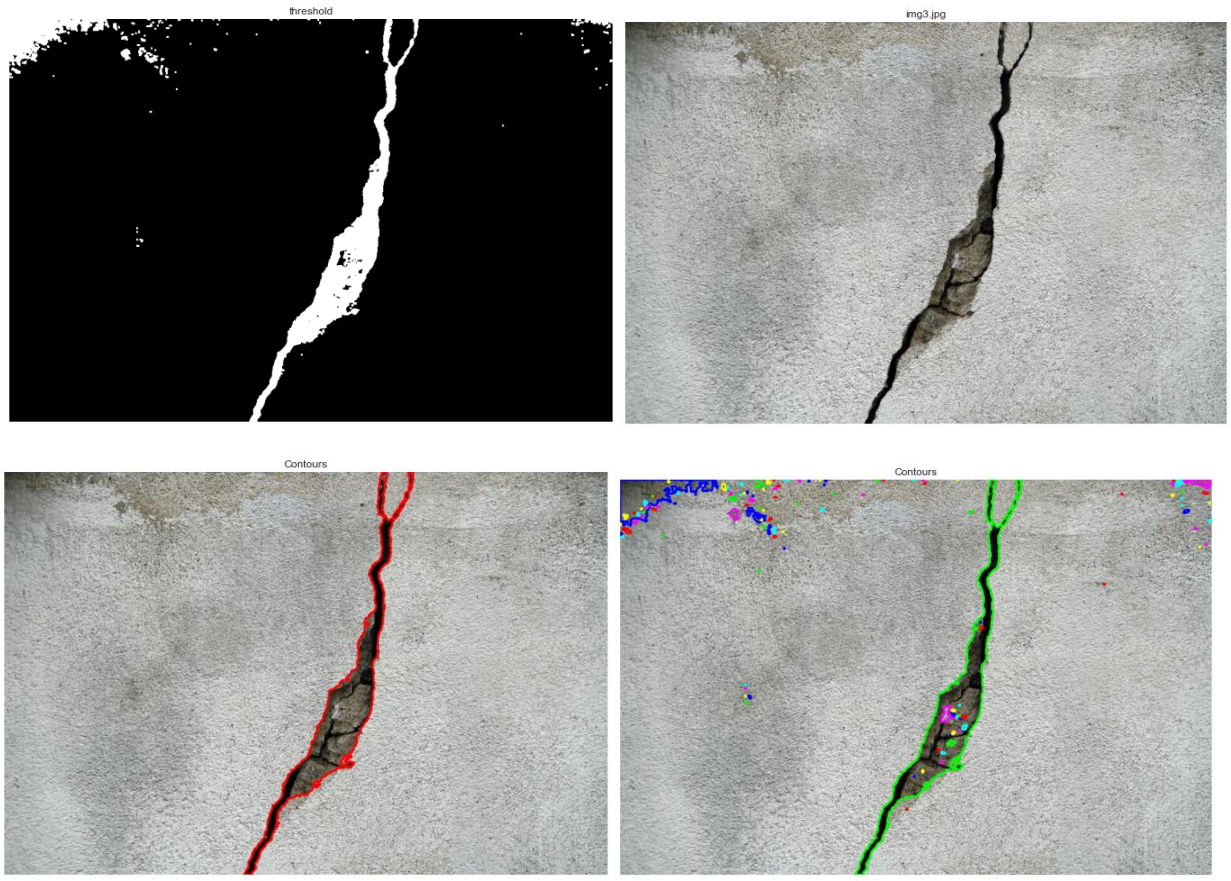
active_contour

```
skimage.segmentation.active_contour(image, snake, alpha=0.01, beta=0.1, w_line=0, w_edge=1,
gamma=0.01, max_px_move=1.0, max_num_iter=2500, convergence=0.1, *,
boundary_condition='periodic', coordinates='rc')                                    [source]
```

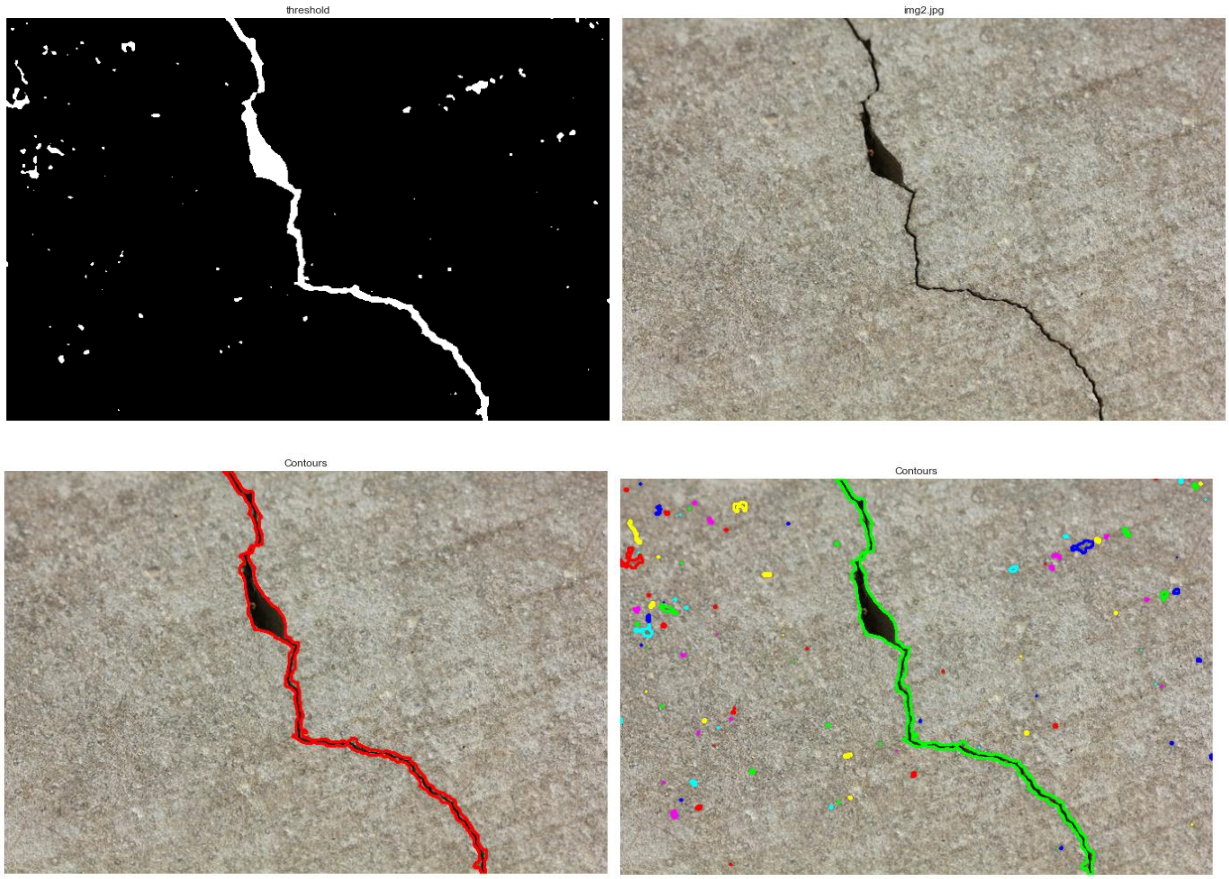
سوال ۵)

طبق آزمایش‌های انجام شده، به این نتیجه رسیدیم که این الگوریتم برای یافتن ترک‌ها مناسب نیست. این الگوریتم قسمت زخیم این ترک‌ها را تقریبا خوب جداسازی می‌کند اما در بخش‌های باریک به مشکل می‌خورد.

روش جایگزین پیشنهادی، استفاده از آستانه‌گیری تصویر و یافتن کانتورهای تصویر به کمک الگوریتم find_contours از کتابخانه OpenCV است که به خوبی این ترک‌ها را جداسازی می‌کند.



تصویر بالا راست: تصویر اصلی، **تصویر بالا چپ:** آستانه‌گیری، **تصویر پایین راست:** یافتن کانتورها، **تصویر پایین چپ:** فیلترکردن کانتورهای بسیار کوچک

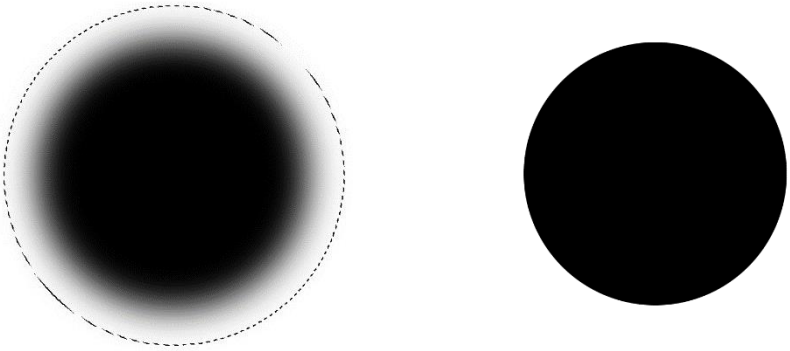


تصویر بالا راست: تصویر اصلی، **تصویر بالا چپ**: آستانه‌گیری، **تصویر پایین راست**: یافتن کانتورها، **تصویر پایین چپ:** فیلترکردن کانتورهای بسیار کوچک

نتایج نامناسب با استفاده از active_contour:

نتیجه مناسب active_contour برای اشکال محدب و غیرنازک:

دلیل عملکرد نامناسب الگوریتم active_contours برای بخش‌های باریک ترک‌ها را در ادامه بررسی می‌کنیم.



در تصویر دایره بالا پس از اعمال مقداری Blur اطلاعات لازم برای محاسبه جهت حرکت کانتور اولیه موجود می‌شود و نقاط مختلف کانتور به سمت رسیدن به بیشترین امتیاز هدایت می‌شوند و در نهایت در لبه‌های تصویر متوقف می‌شوند.

اما در تصویر خط، به دلیل باریک بودن خط، پس از اعمال Blur، ناحیه مشکی رنگ وسط خط از بین رفته و نقاط روی کانتور از هر دو سمت خط سعی میکنند از خط عبور کنند و در این بین نوسان می‌کنند. اگر مقدار Blur را کمتر در نظر بگیریم، باید کانتورهای اولیه را بسیار به خط نزدیک کنیم تا بتوانند از اطلاعات ساخته شده توسط Blur با اندازه کم استفاده کنند و به سمت خط حرکت کنند.

منبع بخشی از توضیح : اسلایدهای Shree Nayar (https://fpcv.cs.columbia.edu/Monographs)

Attracting Contours to Edges

Image with Initial Contour

Gradient Magnitude Squared $\|\nabla I\|^2$

Blurred Gradient Magnitude Squared $\|\nabla n_\sigma * I\|^2$

Maximize Sum of Gradient Magnitude Square

≡ Minimize –ve (Sum of Gradient Magnitude Square)

≡ Minimize $\boxed{E_{image} = -\sum_{i=0}^{n-1} \|\nabla n_\sigma * I(v_i)\|^2}$ $\boxed{1}$

ویدیوی مکمل: https://www.youtube.com/watch?v=FROJUMk9P3Y

پایان