اصول پردازش تصاویر

نیم‌سال اول ۱۴۰۰-۱۴۰۱

مدرس: دکتر مصطفی کمالی تبریزی

دانشکده علوم ریاضی

گزارش تمرین سری چهارم

نام و نام‌خانوادگی: -

شماره دانشجویی: -

کد های پاسخ درون فایل ZIP پاسخ به عنوان $q1.py$ تا $q5.py$ و نیز نتایج حاصل شده از آنها از $res01.jpg$ تا $res11.jpg$ ذخیره شده‌اند. در این سری از تمرین ها تنها از محمدعلی علما مشورت های جزئی در فرمول بندی و الگوریتم viterbi در سوال ۵ گرفته شد.

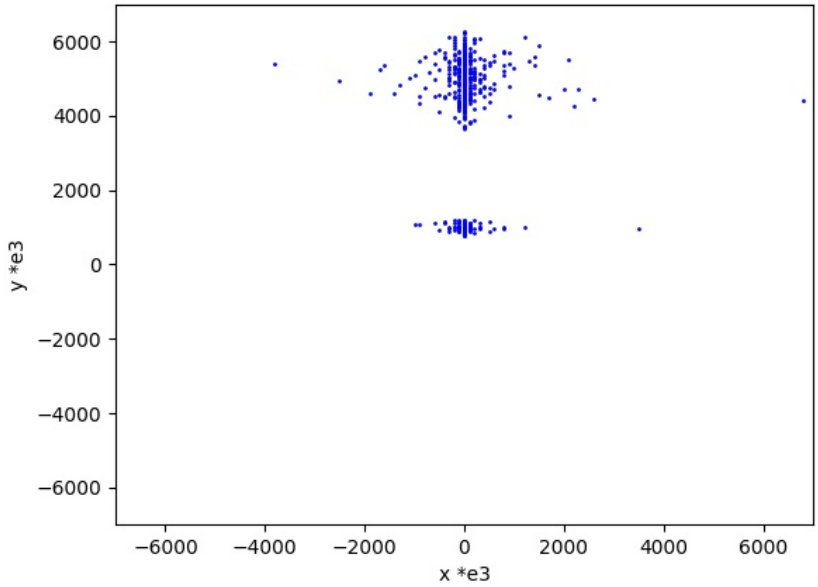
پرسش ۱) K-means

برای حل این سوال، نام فایل txt شامل داده ی مورد نظر را از کاربر میگیریم و آنرا به تابع $q1$ میدهیم. در این تابع تابع getdata را صدا میزنیم و از فایل با استفاده از یک حلقه و توابع پایتون، کل داده را دریافت میکنیم و آنرا به عنوان ماتریس Data از تابع خارج میکنیم. از آنجا که عدد ۷ برای داده ها یک کران بالا به حساب می آید، ما آرایه Data و عدد ۷۰۰۰ را به تابع drawplot میدهیم که ۱۰۰۰ برابر هر داده را برای ما رسم کند و نمایش دهد. تصویر حاصل از این ترسیم که اعداد همانطور که در مخور ها گفته شده است ۱۰۰۰ برابر شده اند، ذخیره میگردد.

در مرحله بعد داده ها به همان صورتی که هستند به تابع kmean داده میشود. این تابع داده، اندازه داده و تعداد خوشه را میگیرد و بعنوان خروجی یک لیست تک بعدی با همان اندازه تعداد داده ها خروجی میدهد که مقدار هر کدام، شماره خوشه را مشخص میکند. روش کار هم به این صورت است که ابتدا k نقطه تصادفی مشخص میشوند و وارد startpoints میشوند. سپس بر روی کل داده ها حلقه میزنیم و فاصله ی هرکدام را تا نقاط startpoints پیدا میکنیم. startpoints ای که کمترین فاصله را با نقطه مورد نظر دارد را پیدا کرده و نقطه را درون cluster مناسب همان نقطه قرار میدهیم. پس از پایان پیمایش روی خوشه ها، میانه ی داده های هر خوشه را پیدا میکنیم و بررسی میکنیم که آیا این نقاط همان نقاط قبلی هستند، یعنی مجموع فاصله هایشان با نقاط قبلی کمتر از 0.001 هست یا خیر.

اگر کمتر بود یا این نقاط برابر با نقاط دو مرحله پیش بودند(حرکت نوسانی داشت) از حلقه خارج شده و خوشه های نهایی به تابع $q1$ برمیگردد. در غیر اینصورت مقادیر startpoints برابر با میانه های جدید قرار میگیرند و برنامه ادامه پیدا میکند تا در نهایت به یک سری خوشه مناسب برسیم. سپس در تابع اصلی لیستی که برمیگردد را میگیریم و بر اساس اون k خوشه متشکل از داده ها میسازیم. و در نهایتا با استفاده از تابع drawplot و تابع plt.scatter آنهارا با رنگ های مختلف رسم میکنیم و ذخیره میکنیم. هر بار نتیجه ی نسبتا متفاوت (و البته نادرست) میدهد. دو نمونه از این اجرا ها در نتایج ذخیره شده اند.

پس از چند اجرا، کاملا مشخص میشود این روش برای این داده ها که بصورت دایره ای تو در تو هستند اصلا مناسب نیست، چرا که دو توده نداریم که کمی از هم فاصله داشته باشند و باید این دو خوشه تو در تو نقاط مرکز یکسان باشند که این بر خلاف الگوریتم کارکرد kmean میباشد (وقتی میانه از نقاط میگیریم از آنجایی که فاصله ی نقاط خارجی از نقطه ۱ مثل نقطه ۲ هست، جابجایی و خوشه بندی جدید به درستی اتفاق نمی‌افتد. بنابر این هیچگاه مرکز این دو خوشه نمیتوانند کنار هم در مرکز دو دایره از داده ها قرار بگیرد) . پس ما به یک دستگاه دیگر نیاز داریم، از آنجا که فاصله نقاط از مرکز صفحه حدودا دو بازه ی کاملا جدا دارد، پس میتوانیم محتصاتی را انتخای کنیم که یک عامل r داشته باشه که r فاصله از مرکز است. در اینجا ما مختصات (r, θ) را انتخاب کرده و تغییر مختصات میدهیم ، یعنی مختصات را از دکارتی به قطبی تغییر میدهیم. همان داده های اولیه بصورت (شکل ۱) در می‌آیند.

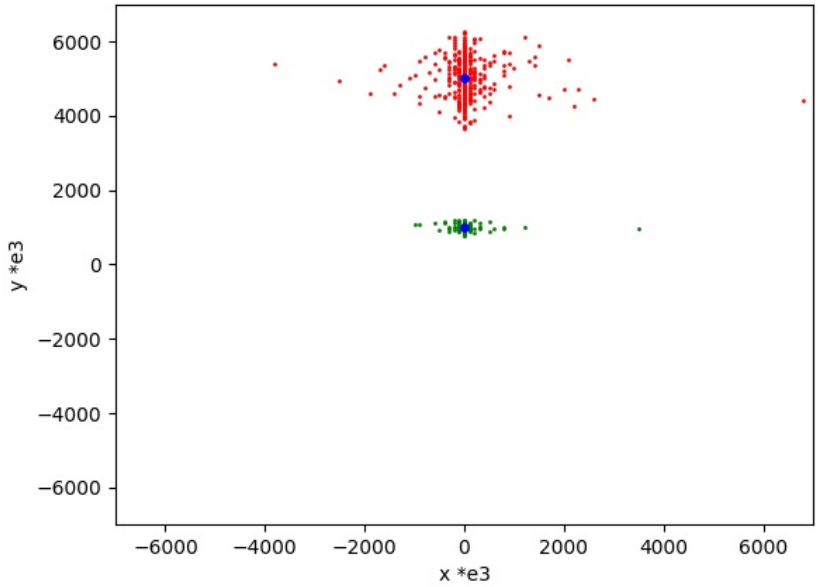


شکل ۱: تصویر اولیه در مختصات قطبی

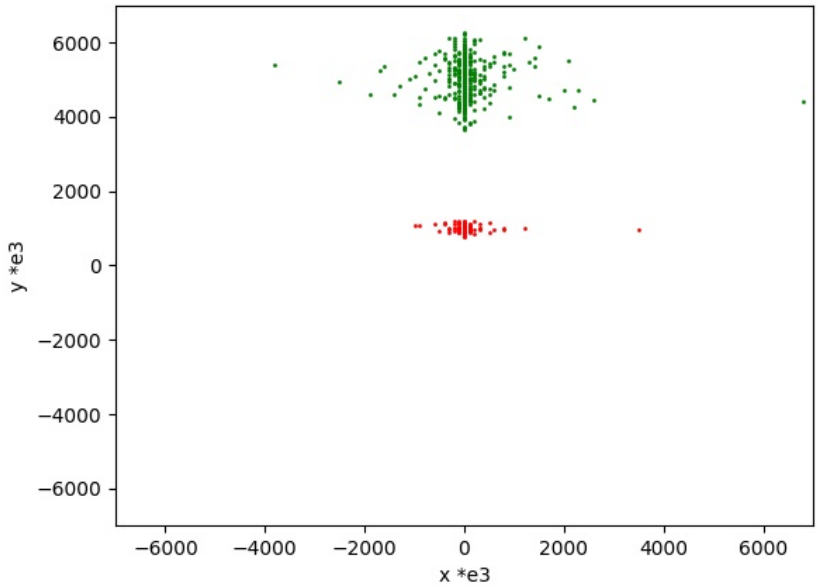
حال اگر همین داده ها را عینا به تابع kmean بدهیم، در پایان پروسه نتایج (شکل ۲) حاصل میشوند. (نقاط آبی پررنگ مرکز دو دسته هستند)

که با حذف مراکز به نتیجه (شکل ۳) میرسیم.

حال به راحتی با استفاده از همان sin و cos که مختصات را تبدیل به مختصات دکارتی میکنیم اما دست به لیست خوشه ها نمیزنیم، سپس این خوشه بندی جدید را در مختصات دکارتی نشان داده و آن را ذخیره میکنیم. (در بخش نتایج، این نتیجه موجود است)



شکل ۲: تصویر خوشه بندی شده در مختصات قطبی به همراه مراکز خوشه ها



شکل ۳: تصویر خوشه بندی شده در مختصات قطبی

پرسش ۲) Mean-Shift

برای سوال دو همانطور که گفته شده بود، از روش mean-shift استفاده شد. کارکرد این روش به این صورت است که در ابتدا تصویر دریافت شده و در بخش اصلی کد، به فضای رنگی YCbCr میرود. این تصویر هم به تابع $q2$ به عنوان آرگومان ورودی داده میشود ورودی دیگر شعاع همسایگی است که بعدا توضیح داده میشود. خروجی این تابع نیز به قضای رنگی BGR بازگشته و نمایش داده میشود.

درون تابع در ابتدا یک ماتریس ۳ بعدی به نام histogram تعریف میشود. کار این مارتیس انباشت کردن رنگ تمام پیکسل های تصویر است. یک حلقه ی تودرتو تعریف میکنیم. در این حلقه تک تک پیکسل های تصویر را مورد بررسی قرار میدهیم و مقدار هر ۳ لایه رنگی آن را به عنوان ورودی به ماتریس ۳ بعدی histogram میدهیم. از آنجا که حداکثر این مقادیر میتوانند ۲۵۵ باشند، پس این ماتریس یک ماتریس $256 \times 256 \times 256$ میباشد. در ادامه یک ماتریس $r \times r \times r$ تعریف میکنیم. این یک مکعب است که تمام درایه هایی از آن که از نصف طول مکعب به آن نزدیک تر هستند ۱ و مابقی ۰ میباشند. r) هم یکی از آرگومان های تابع $q2$ است)بدین ترتیب انگار یک کره با مقادیر ۱ در اختیار داریم.

در اینجا ماتریس dual را دقیقا با ابعاد histogram میسازیم و مقادیر آن را ۰ میگذاریم. در بخش بعد با استفاده از ۳ حلقه ای که درون هم قرار دارند، تمام مقادیری از histogram که مقدار بین ۰ تا r یا ۲۵۵-r تا ۲۵۵ دارند را به لب مرکز معکب داخلی histogram می‌آوریم که اولا تمام توسط کره ای که قرار است این مکعب را پیمایش کند پوشش داده شود، و دوما به دلیل اینکه برای پیدا کردن مرکز خوشه ها به دیواره های معکب histogram برخورد نکنیم. بعد از این بخش، وارد یک حلقه ۳ تایی میشویم. این حلقه در هر بعد از r تا $256 - r$ پیمایش میکند. بعد از بخش هایی که برای افزایش سرعت اجرا نوشتیم، در بخش اصلی این حلقه یک حلقه دیگر تعریف میکنیم که هرگاه درون histogram به درایه ای رسیدیم که برابر ۰ نبود وارد این حلقه بشویم. در ابتدای حلقه ی بدون شرط جدید، مختصات نقطه ای که در حال بررسی آن هستیم را ذخیره میکنیم. سپس بلافاصله دو عدد ماتریس میسازیم. یکی ضرب همسایگی تعریف شده برای نقطه ای histogram که بر روی آن پیمایش میکنیم در ماتریس کروی ساخته شده، و دیگری نیز همین همسایگی برای نقطه مشابه در dual . به اینصورت همسایگی کروی را حول این دو نقطه را بدست می آوریم و ذخیره میکنیم.

حال با استفاده از ۳ حلقه، از ۳ جهت میانگین وزن داری برای کره ی بدست آمده پیدا میکنیم. (وزن هر درایه ی کره برابر با مقدارش است و این میتواند هر عددی باشد) . بعد از رند کردن و گسسته سازی این مقادیر، چند تا شرط را بررسی میکنیم.

۱. مقدار بدست آمده اگر خارج از فضای جستجوی ما قرار داشت، متوقف شود. در این موقعیت ۲ اتفاق امکان دارد بیوفتد، اگر در همسایگی dual ما مقداری منفی پیدا شد، یعنی در آن همسایگی یک پیکسل هست که در خوشه قرار دارد، پس هم پیکسل آخر هم همه آنهایی که در این راه ذخیره شده بودند برابر با مقدار خوشه ی همسایگی قرار میگیرند. اگر هم هیچ خوشه ای در آن همسایگی نبود، یک خوشه جدید تشکیل میدهیم و کل پیکسل های این راه را در dual برابر خوشه جدید قرار میدهیم. (منفی بودن یک درایه dual ، یعنی آن درایه در ۱ خوشه هست با شماره خوشه ی قدر مطلق آن مقدار)

۲. اگر مقدار جدید برابر با مقدار قبلی بود، همین کار های قبل تکرار شود و یا مقادیر بررسی شده به خوشه جدید ریخته بشوند یا خوشه ی موجود در همسایگی.

۳. اگر هیچکدام صدق نکرد، ببیند خود پیکسلی که در آن قرار گرفته ایم در خوشه ای نیست؟ اگر که هست مقادیر قبلی را در آن بریزیم.

اگر هرکدام از شروط بالا به نحوی برقرار بود، از حلقه خارج میشویم و به سر درایه بعدی میرویم. ولی اگر هیچکدام برقرار نبود، همین کار ها را عینا با مقادیر جدید میکنیم و آنقدر پیش میرویم که یکی از این حالا رخ بدهد.

بعد از اینکه ماتریس dual ساخته شد، یک حلقه ۳ تایی تو در تو داریم که در اینجا تک تک مقادیر dual بر حسب مکانی که هستند بصورت ضرب در مقداری که در ابتدا از histogram داشتند به درون یک آرایه ی دو بعدی از مقادیر هر خوشه قرار میگیرد. مقادیر اصلی (وزن های هرکدام) نیز درون آرایه ای مشابه همان آرایه خوشه ها ریخته میشود. سپس با

شکل ۴: تصویر با r=۷ و بدون فیلتر

استفاده از دو عدد حلقه، در روی خوشه ها و مقادیرشان پیمایش میکنیم و همه ی خوشه ها را جدا جدا با هم جمع کرده و بر وزنی که قبلا بدست آوریم تقسیم میکنیم و به این صورت، برای هر خوشه یک آرایه ۳ تایی که رنگ میانگین آن خوشه را نشان میدهد خواهیم داشت. این مقادیر در claverages ذخیره میگردند.

در مرحله ی آخر هم بر روی تصویر پیمایش کرده و به هر پیکسلی که میرسیم از طریق dual شماره خوشه و از طریق claverages میانگین رنگ آن خوشه را بدست میاوریم و آن را در تصویر نهایی جایگزین میکنیم. و خروجی میگیریم. برای بهتر کردن نتیجه، نیاز به آزمون و خطاهای مختلف بود. تصویر اصلی به دلیل ماهیتش، مقدار زیادی نویز دارد (شکل ۴) و اگر فیلتر median قبل از شروع اعمال شود بسیاری از جزئیات را از بین میبرد. حتی اگر بعد از میانگین گیری هم اعمال بشود، برای r های نامناسب نتایج خوبی ندارد. (شکل ۵) همانطور که در تصویر ۵ مشاهده میشود، رنگ ها بسیار کم و کاملا رنگ بندی غیر طبیعی است. این به دلیل کم بودن تعداد خوشه هاست. (شکل ۶) در تصویر ۶ از آنجا که خوشه ها در همسایگی ۵ بسیار زیاد است، فیلتر mrdian باعث کاهش نویز نمیشود، بلکه باعث کاهش کیفیت و بلوری شدن تصویر میشود. تصویر زیر بدون فیلتر ۵ است و میبینیم که رنگ ها خیلی کامل هستند و از دور تفاوت زیادی هم با تصویر اصلی ندارند. (شکل ۷) نیز نتیجه مطلوب ما نیست. پس از آزمون و خطا تصویر $r = 7$ با فیلتر median در انتها و اندازه ۱۵ ، به عنوان بهترین تصویر انتخا شد و بعنوان خروجی قرار داده شد.

شکل ۵: تصویر با r=۹ و با فیلتر

شکل ۶: تصویر با r=۷ و با فیلتر

شکل ۷: تصویر با r=۷ و بدون فیلتر

پرسش ۳ (SLIC

حل سوال ۳ بوسیله ی مقاله ی SLIC و نکات و الگوریتم هایش صورت گرفت. در ابتدا مانند قبل در کد اصلی تصویر خوانده شده و به همراه ۲ آرگومان تعداد بخش ها که از کاربر میگیرد و تعداد ایترشن ها به تابع $q3$ داده میشود. خروجی نیز تصویر قطعه بندی شده نهایی میباشد که در انتهای کد ذخیره میشود.

پس از ورود به تابع، ابتدا گرادیان تصویر با استفاده از تابع لاپلاسین گرفته شده و درون laplacian ریخته میشود. سپس مساحت تصویر بعنوان N بدست می‌آید و از تقسیم این عدد تقسیم بر تعداد قطعات و گرفتن رادیکال آن، طول و عرض حدودی هر قطعه بدست می آید، سپس به بالا رند شده و گسسته شده آن را در s ذخیره کرده و از تقسیم بر طول و عرض تصویر و رند کردن به بالا، همچنین تبدیل آن به عدد صحیح، w و h را بدست می آوریم که به ترتیب قطعه های افقی و عمودی درون تصویر خواهند بود.

سپس یک آرایه برای ذخیره مرکز ها و یک ماتریس dual برای ذخیره لیبل پیکسل ها (که برابر با شماره خوشه هستند) تعریف شده و تصویر را نیز در فضای رنگی Lab بدست می آوریم. با استفاده از یک حلقه محل x و y تمام نقاط مراکز دسته را بدست می‌آوریم، همسایگی ۵ واحدی اطرافش را در تصویر laplacian میگردیم و کمترین مقدار آن در اینن ۲۵ خانه را به عنوان مرکز در آرایه centers قرار میدهیم. سپس برای هم مرکزی که بدست می آوریم، یک همسایگی به ابعاد s در s در نظر گرفته و مقادیر تمام درایه های dual در آن محدوده را برابر با آن خوشه قرار میدهیم.

سپس یک ماتریس tmp با ابعاد مشابه تصویر با مقادیر - ۱ برای ذخیره ی مقادیر آتی میسازیم. بعد از آن حلقه شروع میشود. در ابتدا یک حلقه دوتایی تودرتو داریم که از ۰ تا $h+1$ و نیز $w+1$ پیمایش میکنند. در این روش، بین هر ۴ تا مرکز کنار هم پیمایش میکنیم، بنابراین $(w+1)\times(h+1)$ ناحیه خواهیم داشت. در ابتدا این ۴ نقطه مشخص میشود، گوشه بالا سمت چپ ناحیه برابر مختصات مرکز بالا سمت راست و گوشه ی پایین سمت چپ نیز مرکز ناحیه ی مربوط است. مقدار label هر کدام ازین مراکز در dual در coords ذخیره میشود و هرکجا که در ابتدا و انتهای ناحیه های در حال پیمایش بودیم، بجای مراکزی که وجود ندارد، مقدار انتها یا ابتدای تصویر را قرار میدهیم و به اینصورت نتیجه ی مناسبی میتوانیم بدست بیاوریم. سپس در ادامه روی کل نقاط این نواحی پیمایش میکنیم و مقدار زیر را برای هر ۴ مرکز اطراف این ناحیه بدست می آوریم. نتایج با مقدار $\alpha = \frac{6}{S}$ بدست آمده اند.

$$d_{lab} = (L_k - L_n)^2 + (a_k - a_n)^2 + (b_k - b_n)^2$$

$$d_{xy} = (X_k - X_n)^2 + (Y_k - Y_n)^2$$
$$D = d_{lab} + \alpha d_{xy}$$

سپس کمترین این مقدار برای ۴ نقطه اطراف را پیدا کرده و مقدار آن را برابر با پیکسلی که روی آن درحال پیمایش هستیم قرار میدهیم. آنقدر این کار هارا ادامه میدهیم که کل پیکسل های dual مقدارشان بروز شود. در پایان، این بخش، بر بروی Dual یک فیلتر median از تابع scipy با اندازه کرنل $2S^{2.5}$ میزنیم. با اینکار حفره ها و نویز های تو در توی بوجود آمده بخصوص در دریا، بسیار کمتر شده و خوشه های بدست آمده بسیار یکدست تر میشوند. حال در اینجا اگر ایتریشن هایمان تمام شده بود برنامه توقف میشود و در غیر اینصورت به یک حلقه تودرتو وارد میشویم.سپس در هر حلقه یک عدد از ۰ تا k داریم که این برابر شماره خوشه میباشد. سپس یک آرایه mod با ابعاد dual ساخته و آن مقداری که اشاره کردیم را در این آرایه برابر ۱ و بقیه درایه ها تماما ۰ میشوند. سپس از طریق روابط ماتریسی و ضرب داخلی ماتریس در بردار وزن، میانه ی هر خوشه را پیدا میکنیم و نگه میداریم. سپس نقاط مرکز مرتبط با آن شماره خوشه را نگه میداریم و فاصله ی مراکز جدید و قدیم را مقایسه میکنیم. سپس مقادیر قدیمی مرکز خوشه را به مقدار جدید بروز کرده و شرط میگذاریم که اگر فاصله ی مرکز قدیمی خوشه با مرکز جدید از مقدار threshold کمتر بود، یکی به errcount افزوده شود، ($threshold = \frac{S}{8}$)

در پایان حلقه هرگاه errcount از مقدار 0.2 تعداد کل خوشه های پیمایش شده کمتر بود، یعنی به مقدار خوبی از دقت رسیدیم که میتوانیم با استفاده از break از حلقه خارج بشویم. پس به دو روش مختلف میتوانیم از حلقه خارج بشویم. بعد از حلقه ، dual را به $uint8$ تدبیل میکنیم، از آنجا که در اینجا مقدار dual برایمان اهمیتی ندارد و بیشتر تغییرات مقادیر مهم است، کافی است با استفاده از فیلتر Canny ، در ابتدا این فیلتر را روی dual بزنیم و با یک فیلتر ،blur کمی پهنای خطوط را بیشتر کنیم. در ادامه با ضرب dual در حداکثر مقدار ۲۵۶ و سپس اضافه کردن آن به تصویر و برابر با ۰ کردن هر پیکسلی از تصویر که مقدار بیشتر از ۲۵۵ دارد، به سادگی میتوانیم خطوط تغییرات را در تصویر اصلی رسم کنیم. در پایان نیز تصویر بدست آمده را برمیگردانیم و در بخش اصلی کد آن را ذخیره میکنیم.

$$\alpha = \frac{6}{S}$$

پرسش ۴) Segmentation

برای حل این سوال، روند کلی به این صورت است که تصویر به کاربرد داده میشود، کاربر با استفاده از رابط کاربری چند نقطه از پس زمینه و شی را انتخاب میکند و این تصویر به عنوان ماسک به تابع grabcut از opencv وارد میشود و یک ماسک نهایی از اشیا میدهد که ما ماسک را روی تصویر کشیده و آن را نمایش میدهیم. به سراغ توضیح کد میرویم. (این روش بطور کامل با الگوریتم های مربوط در اسلاید شماره ۱۹ به عنوان یک روش گراف بیس توضیح داده شده بود) در آغاز چند تا مقدار ثابت و ماتریس ها با رنگ ها و اندازه های متفاوت برای brush کردن روی تصویر تعریف شده و سپس تصویر دریافت میشود و مقیاس مناسبی از آن که یک چهارم است در یک آرایه ذخیره شده و آنرا در یک پنجره نمایش میدهیم. برای این پنجره یک تابع برای کار با موس روی آن بنازم operate میسازیم و سپس وارد یک حلقه بی پایان برای دریافت کارهای موس میشویم و نیز شرط پایان حلقه صدا خوردن کلید esc کیبورد میباشد.

لازم به ذکر است یک ماسک تعریف میکنیم کهدر ابتدا تماما مقادیر -۱ دارد و با انتخاب کردن کاربر بعضی مقادیر ۰ و ۱ نیز خواهد گرفت که در ادامه -۱ ها تبدیل به ۲ و اعداد بزرگتر از ۲ برابر با ۱ میشوند. (به دلیل فراهم کردن ماسک مورد نیاز (grabcut در این تابع ۴ تا ایونت مختلف داریم.

۱. کلیک چپ ماوس، برای حالتی که روی حالت انتخاب شی هستیم، با هر کلیک سمت چپ یک همسایگی ۳ در ۳ از محل کلیک را به عنوان شی در نظر میگیریم. به اینصورت که مقدار ۱ به ماسک خواهیم داد و یک مربع آبی نیز به تصویر که باعث آبی شدن آن بخش از تصویر میشود.

۲. کلیک راست ماوس، برای تغییر حالت از پس زمینه به شی مورد استفاده قرار میگیرد.

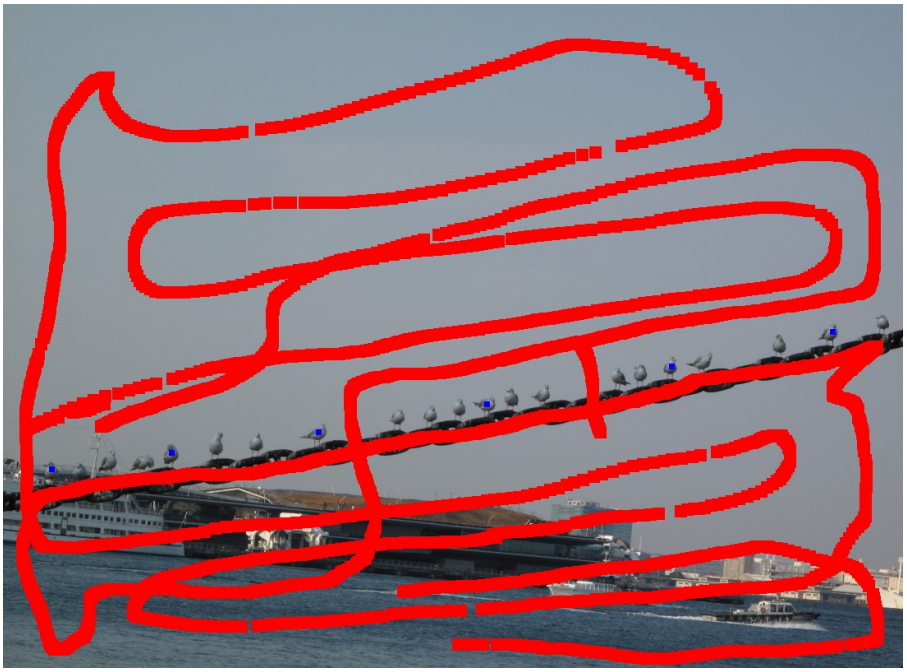
۳. حرکت ماوس، برای وقتی که در حالت پس زمینه قرار داریم، در این حالت با جابجا کردن ماوس، مقدار ۰ به ماسک خواهیم داد و یک مربع قرمز نیز به تصویر که باعث قرمز شدن آن بخش از تصویر میشود.

۴. دابل کلیک چپ، که باعث دادن تصویر و ماسک به تابع و شروع الگوریتم میشود.

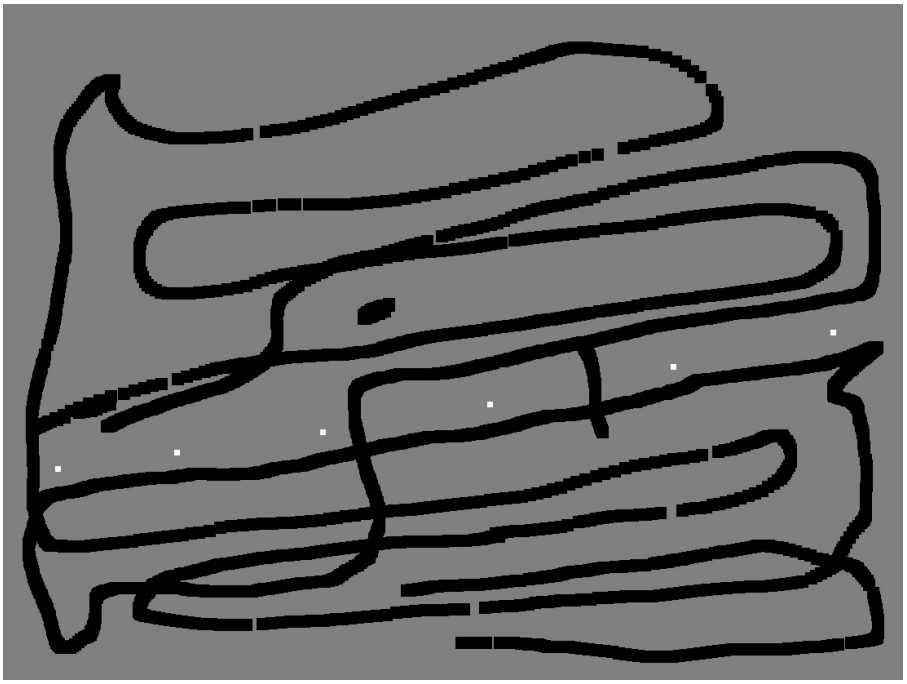
کاربر وقتی برنامه را شروع میکند، در ابتدا در حالت انتخاب شی قرار دارد و میتواند با کلیک چپ یک شی را انتخاب بکند یا با کلیک راست حالت را به انتخاب پس زمینه تغییر بدهد. لازم به ذکر است در هر چرخه تصویر بروز میشود و مقادیر بیشتر از ۲۵۵ برابر با ۲۵۵ میشود. در ادامه اگر کاربر دابل کلیک بکند، همانطور که واضح است، ماسک به صورت مناسب با ابعاد یک دوم تصویر اصلی در می آید. سپس با یه دوم شده ی تصویر آنرا به تابع grabCut میدهیم و یک خروجی به نام mask از آن میگیریم. در ادامه هرجا که مقادیر آن ۲ یا ۰ هستند را برابر ۱ کرده و اندازه اش را نیز ۲ برابر میکنیم. سپس با استفاده از یک حلقه، هرجا که ماسک خروجی برابر ۱ بود، طیف سبز آن پیکسل را زیاد میکنیم که ماسک روی تصویر مشخص شود. در پایان نیز از تصویر بدست آمده خروجی میگیریم و آنرا ذخیره میکنیم و با دستور sys.exit برنامه را میبندیم.

در ادامه تصویر موقع انتخاب قسمت های شی و پس زمینه (شکل ۸)، ماسک انتخاب شده توسط کاربر که نرمالایز شده ورودی تابع است (شکل ۹) و در آخر نمونه شکلی را میبینیم که حاصل ضرب تصویر در ماسک اولیه است (تصویر ۱۰).

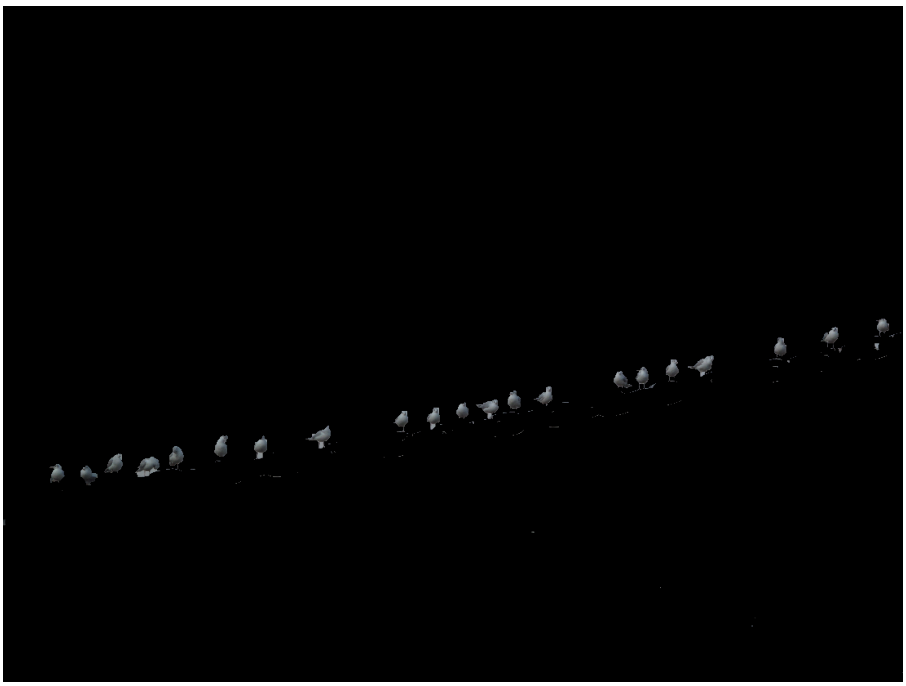
(عیب این روش همانطور که در کلاس درس تدیس شد، این است که برای نقاط گسسته عملکرد کاملا مطلوبی ندارد، چرا که برای گسترش گراف ها، نیاز است که یک نقطه حداقل وجود داشته باشد. در اینجا اگر روی تعدادی پرنده حداقل یکبار کلیک نکنیم، این احتمال وجود دارد که پرنده شناسایی نشود، زیرا اصلا گراف شی در آن نقطه ایجاد نخواهد شد.)



شکل ۸: انتخاب نواحی پس زمینه و اشیا توسط کاربر



شکل ۹: ماسک ورودی تابع



شکل ۱۰: تصویر حاصل شده پس از ضرب تصویر در ماسک خروجی

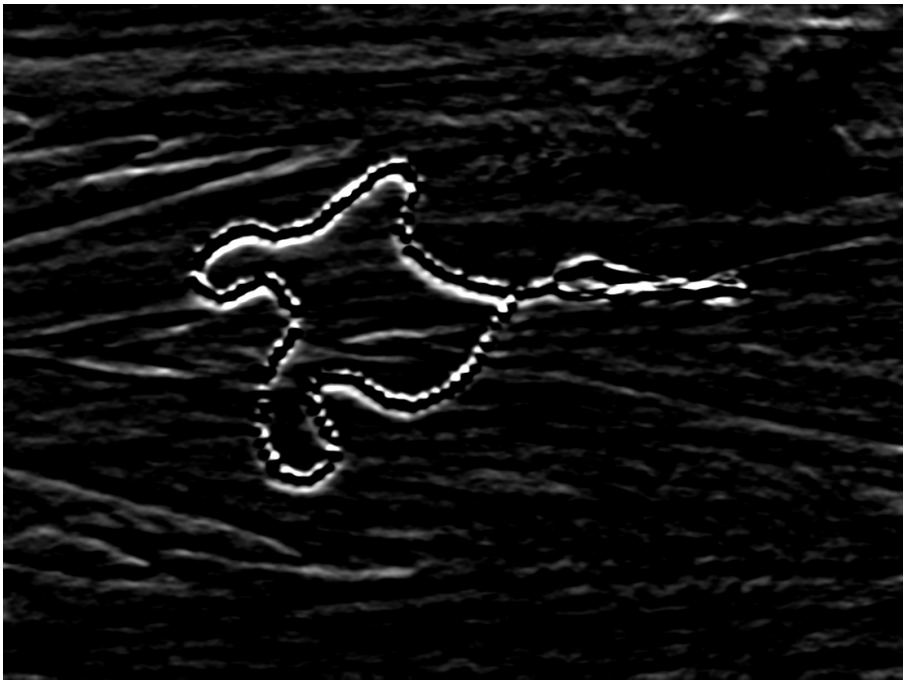
پرسش ۵) Active Contours

برای حل این سوال، عمدتا از روش های تدریس شده با کمی خلاقیت استفاده شد. در ابتدا پس از اینکه تصویر در بخش اصلی کد دریافت شد در ابتدا رویش فیلتر مدین با سایز ۱۳ میخورد که بریدگی های نامربوط را کم کند. sobelX و sobelY با مقدار مشتق ۱ و ابعاد ۵ که گرادیان های افقی و عمودی ما هستند تعریف شده و افکت بلور گاوسی با سیگمای ۲ برروی آنها میدهیم که کمی اطرافشان هم روشن شوند. این دو تصویر را در شکل های ۱۱ و ۱۲ میبینید.

در ادامه دقیقا مانند سوال قبل، یک پنجره با تابع operate برای دستورات آن باز میشود. در اینجا به روشی ساده در فشردن کلیک مرکز یک دایره را میگیریم (همزمان دایره قرمز کوچی در کرمز رسم میشود.) و هرجا آن را رها کنیم، به معنای این است که شعاع ما تا آنجاست. در همان لحطه دایره مورد نظر رسم میشود و سپس مختصات کلیک و رها کردن را بصورت دو آرایه به تابع $q5$ میدهیم. در ابتدا یک ویدیو برای ذخیره فریم ها در آن باز میکنیم. سپس شعاع دایره را بدست میاوریم و در raduis قرار میدهیم. بعد از آن درون یک حلقه داخل یک آرایه، طول و عرض هر کانتور را با توجه به ابعاد دایره و تعداد کانتور پیدا کرده و در یه آرایه قرار میدهیم. dd را که بیانگیر شروع شدن فاز ۲ پیشروی است را False کرده و وارد یک حلقه میشویم که ما برای مقدار ۳۵۰ را حد در نظر گرفته ایم، اما به روش های دیگری میتواند حرکت تمام بشود.

در هنگام شروع این حلقه dbar به عنوان میانگین بتوان ۲ فاصله ی همه ی کانتور ها با کناری هایشان تعریف میکنیم و آنرا بر 1.6 تقسیم میکنیم. سپس در صورتی که فاز دوم شروع شده بود، برحسب شماره ایتریشنی که در آن قرار داریم، برای a، b ، g و همچنین m مقادیر مناسبی قرار میدهیم که کاربرد هایشان در ادامه ذکر میشوند. مرکز ثقل کل کانتور ها نیز با استفاده از میانه ی تمام مولفه های اول کانتور ها و مولفه ی دومشان تعیین میگردد. در اینجا تصویر اصلی را درون یک تصویر موقت کپی میکنیم و کل دایره ها و خط های مربوط به کانتور ها را روی آن رسم میکنیم. در ادامه وارد تابع viterbi میشویم، به این تابع تعداد کانتور ها، لیست آنها، dbar ، مرکز ثقل آنها و پارامتر های مربوط دیگر را که در ادامه توضیح میدهیم، داده میشود و لیست کانتور های جدید به عنوان خروجی از این تابع گرفته میشود.

کارکرد این تابع کاملا مشابه توضیح داخل کلاس هست، تنها درون پارامتر های انرژیشان تغییراتی اعمال شده است. کار ما مانند این است روی یک حلقه بسته به دنبال پیدا کردن کوتاه ترین مسیر میگردیم. در ابتدا یک ماتریس هزینه ۳ در ۳ در تعداد کانتور ها با مقادیر نامحدود داریم. ۹ ایتریشن روی درایه ی آخر این ماتریس (که به اولی وصل میشود) انجام میدهیم و در هر مرحله کمترین هزینه رفتن به تک تک خانه های لایه ی بعدی ماتریس بررسی کرده و کمترین را به آن اضافه



شکل ۱۱: SobelY

شکل ۱۲: SobelX

کرده و مسیری که رفتیم نیز در path ذخیره میگردد. در پایان نیز به همان خانه ای که شروع کردیم میرویم و این هزینه نهاییمان است. در انتهای حلقه اصلی، هزینه نهایی با هزینه ی نهایی حرکت قبلی (که نقطه شروع و پایان متفاوت داشت) مقایسه میشود. اگر این مقدار کمتر بود، ماتریس هزینه و مسیر در ماتریس هزینه و مسیر نهایی ریخته میشوند. بعد از اینکه مسیر نهایی پیدا شد، از نقطه آخر ماتریس اصلی شروع میکنیم و کمترین مقدار را بدست می آوریم، این حرکت بعدی کانتور آخر ما خواهد بود. در ادامه ۱۰۰ بار، نگاه به نقطه ای که در آن هستیم میکنیم و با توجه به ماتریس ،path مسیر بعدی را بدست می آوریم و کانتور را بر همان اساس آپدیت میکنیم. آنقدر این کار را تکرار میکنیم که برای کل کانتور ها موقعیت جدید بدست بیاید.

کانتور های جدید را به عنوان خروجی تابع برگردانده و بررسی میکنیم که آیا نسبت به قبل ثابت مانده است یا خیر. اگر نسبت به قبل مانده بود در ۵ ایتریشن هم ثابت ماند، مرحله دوم پیمایش شروع میشود، اگر در مرحله ی دوم حتی یک بار این برابری رخ داد، برنامه پایان پیدا میکند. در غیر اینصورت، یک فریم به ویدیو اضافه شده و تصویر درون پنجره تصویر هم بروز میشود. وقتی هم که برنامه تمام میشود، تصویر موقتی که گفته بودیم، به عنوان تصویر نهایی ذخیره شده و ویدیو ذخیره میگردد.

حال به سراغ توضیح انرژی ها و ضریب های استفاده شده در آن برای الگوریتم viterbi میرویم. دقت کنید p کسر پیشرفتمان در ایتریشن هاست که از ۰ تا ۱ متغیر است.

۱. انرژی درونی ، گرادیان : گرادیان به همان صورتی که در اسلاید ها گفته شده بود محاسبه میشود.

$$e_1 = -g(G_x(x,y)^2 + G_y(x,y)^2)$$

در این تابع هرچقدر به یک مرکز با گرادیان بالاتر نزدیک تر باشیم، انرژیمان منفی تر و کمتر خواهد بود. (برای گرادیان ها از همان sobel ها استفاده میشود.) ضریب g در مرحله اول ۲۰ و در مرحله دوم $50 + 150p^2.7$ میباشد.

۲. انرژی بیرونی اول: فاصله بین دو نقطه متوالی

$$e_2 = a((x_1 - x_2)^2 + (y_1 - y_2)^2 - dbar)^2$$

که کانتور ها مانند یک کش عمل کنند. مقدار a در مرحله اول برابر ۲۰ و در مرحله دوم مساوی است با $60 - 50p$

۳. انرژی بیرونی دوم:

$$e_3 = b((x - r_x)^2 + (y - r_y)^2)$$

که r مرکز تمام کانتور هاست. باعث میشود یک گرایشی تمام کانتور ها به مرکز داشته باشند. در مرحله اول b برابر ۴ و در مرحله بعدی برابر $70p^4$ است.

۴. انرژی درونی سوم: گرانش بین نقاط در اینجا یک تابع گرانش بین نقاطی که اختلاف شماره هایشان بین ۰ تا ۱۲ است بصورت زیر تعیین میشود:

$$e_4 = -m(\sum_{j=i-22}^{i-12} |j-i|/dist(v_j, v_i) + \sum_{j=i+12}^{i+22} |j-i|/dist(v_j, v_i))$$

که dist تابع فاصله بین دو نقطه مذکور است. در مرحله اول m برابر ۰ بوده و برابر شماره ایتریشن ضربدر 3×10^8 میباشد.

همه ی مقادیری که در بالا اشاره شد، برای بهینه تر شدن در سرعت و عملکرد انتخاب شده اند، این مقادیر (و مقادیر نسبتا نزدیک) احتمالا برای نمونه های دیگر نیز کارآمد باشند.