اصول پردازش تصاویر

نیم‌سال اول ۱۴۰۰-۱۴۰۱

مدرس: دکتر مصطفی کمالی تبریزی

دانشکده علوم ریاضی

گزارش تمرین سری پنجم

نام و نام‌خانوادگی: -

شماره دانشجویی: -

کد های پاسخ درون فایل ZIP پاسخ به عنوان $q1.py$ تا $q3.py$ و نیز نتایج حاصل شده از آنها از $res01.jpg$ تا $res10.jpg$ ذخیره شده‌اند. در این سری از تمرین ها از محمدعلی علما در سوال های ۱ و ۲، و نیز از سروش آتشی در سوال ۳، مشورت های جزئی در مورد راهبرد کلی سوال و مواردی مانند ماتریس sparse گرفته شد.

پرسش ۱ - Morphing

روند حل کلی این سوال بصورت زیر است.

۱. بدست آورن نقاط متناظر دو صورت و قرار دادن آنها به صورت متناظر در فایل های $landmarks1.dat$ و $landmarks2.dat$

۲. تحویل یک سری از نقاط به تابع Delaunay کتابخانه scipy و دریافت شماره نقاطی که به عنوان مثلث لحاظ شده اند و استفاده از این شماره نقاط برای پیدا کردن مثلث های متناظر در دو تصویر.

۳. بدست آوردن n نگاشت خطی از n مثلث تصویر اول به مثلث متناظرش در تصویر دوم و ذخیره آنها (و همچنین بدست آوردن n نگاشت دیگر از مثلث های تصویر دوم به اول [که معکوس n تای قبلی میشوند] و خیره آنها)

۴. پیمایش روی کل پیکسل های تصویر و - از طریق رابطه ی مساحت مثلث ها - فهمیدن اینکه هر یک در کدام مثلث قرار دارند (و بطور مشابه در تصویر دوم)

۵. ضرب ماتریسی مختصات پیکسل در حال پیمایش در نگاشت affine متناظر خودش و بدست آوردن جایگاه متناظرش در تصویر دوم (و بطور مشابه در تصویر دوم)

۶. ساخت ۴۳ تصویر دیگر (علاوه بر تصویر اول و آخر) با استفاده از درونیابی نقاط اولیه و نهایی، این تصاویر هم اندازه با تصویر اصلی اند و هر کدام از جایگاه هایش مختصات پیکسل متناظرش در تصویر اول را در فریم n ام نشان میدهد که

$n < 45$ (و بطور مشابه در تصویر دوم)

۷. استفاده از حلقه ۴۵ تایی و ایجاد تناظر بین نقاط مبدا و مقصد در هر فریم (و در هر دو تصویر). بعد از اینکه فهمیدم در مرحله i ام در هر پیکسل تصویر جدید باید از چه مختصاتی برای نگاشت آن در پیکسل پیش رو در تصویر جدید استفاده کنیم، (از آنجایی که عموما این مختصات اعشاری است) میتوانیم با استفاده از درونیابی با استفاده از معکوس مساحت محیط شده بین نقطه اعشاری و پیکسلی که در مختصات صحیح دارد استفاده کنیم. (این روش را عینا در سوال ۳ تمرین سری ۲ استفاده کردیم)

۸. در پایان هر تصویر را در یک لیست میریزیم و با یک حلقه یکبار از اول به آخر و یکبار از آخر به اول تصاویر این لیست را به عنوان فریم به ویدیوی در حال ساخت میدهیم و به این صورت فایل $mp4$. تشکیل میشود و با استفاده از نرم افزار های ساده تبدیل فرمت آن را به gif تغییر میدهیم.

حال به طول مفصل تر به توضیح کارکرد برنامه میپردازیم. درون فولدر $q1 - ext$ درون فایل زیپ برنامه، علاوه بر داده هایی که شامل مختصات نقاط مشخص شده هستند، یک کد پایتون به نام $landmark_maker.py$ قرار دارد که در ابتدا

دو تصویر ورودی را دریافت کرده و با استفاده از کتابخانه dlib و دیتای Train شده از طریق ماشین لرنینگ، ۸۱ نقطه مهم صورت را بدست می آورد، سپس دو تصویر باز میشوند کاربر با کلیک کردن یکنواخت و همسان بین نقاط متناظر بین نقاط صورت، گوش ها، گردن و لباس، بقیه ی نقاط منتاظر را مشخص میکند و در نهایت با فشردن کلید Enter کیبرد، مختصات ۴ گوشه ی تصویر و تعداد نقاط به داده ها اضافه شده و داده ها به عنوان خروجی ذخیره میگردند. (لینک دیتای Train شده)

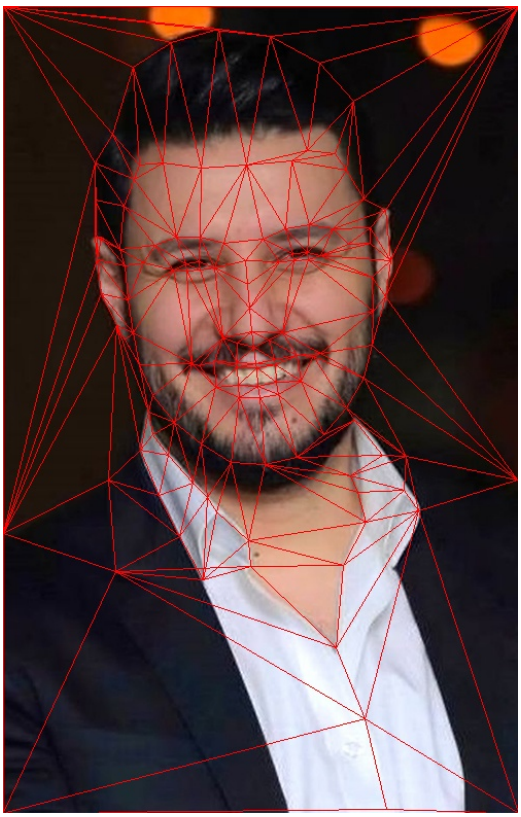
در ادامه داده هایی که بدست آورده این را به کد $q1.py$ میدهیم. این کد در بخش اصلی اجرایی خود، در ابتدا با استفاده از تابع getdata (که مشابه آن در سوال ۱ تمرین سری ۴ پیاده سازی شد)، با استفاده از regex داده ها را میخواند و آن ها را به عنوان یک لیست دو بعدی برمیگرداند. این تابع را دو بار برای دو فایل $landmarks1.dat$ و $landmarks2.dat$ تکرار میکنیم که نقاط مشخص شده بدست بیایند.

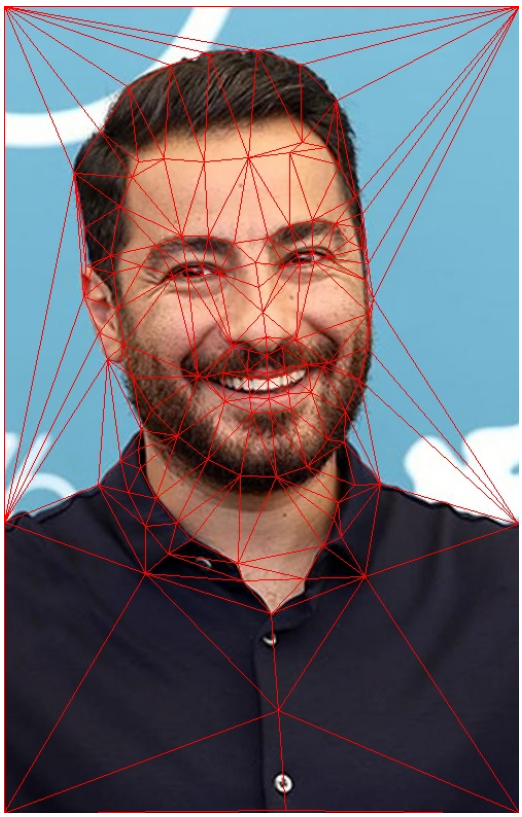
شکل ۱: تصویر نقطه گذاری شده جواد عزتی

شکل ۲: تصویر نقطه گذاری شده نوید محمدزاده

وقتی این لیست نقاط را به تابع Delaunay کتابخانه scipy میدهیم، به ما لیستی از نقاط با دسته های ۳ تایی که در هر دسته ۳ نقطه قرار دارد میدهد. حال پس از اینکه مقایسه که مثلث های متناظر از حاصل از مثلثی کردن لیست اول بیشتر به تصویر دوم میخورد یا بالعکس، لیست شماره های مورد نظر را به هر دو لیست نقاط میدهیم و به عنوان خروجی لیستی از مثلث های متناظر هر دو تصویر بدست خواهیم آورد.
شکل شماره ۳، به صورت بهینه مثلث بندی شده است اما شکل ۴ که از شماره نقاط شکل ۳ استفاده کرده است، ترکیب بعضی از مثلث ها به هم ریخته است. همچنین بعضی از مثلث ها هم را قطع کرده اند. پس این مثلث بندی مناسبی نیست. اما در شکل ۵ که از شماره های نقاط مثلثی شده شکل ۶ استفاده شده است، برعکس شکل ۴ در هم ریختگی مثلث ها و قطع ضلع ها را نداریم، هرچند به نحو بهینه مثلث بندی نشده اند. پس انتخاب ما، دو تصویر دوم (شکل ۵ و ۶) خواهند بود.

سپس دو تصویر و دو آرایه ۳ بعدی از مثلث ها و همچنین متغیر rate که ابعاد تصویر را مشخص میکند (در حالت عادی برابر ۱ است) را به تابع $q1$ میدهیم و ادامه ی روند اجرای کد در این تابع از سر گرفته میشود. حال از آنجا که دو آرایه از رئوس مثلث ها داریم، میتوانیم از متناظر بودن دو به دوی مثلث ها در این دو لیست بهره بگیریم و با پیمایش روی





شکل ۳: در تصویر جواد عزتی مثلث بندی توسط نقاط مثلثی شده گرفته شده از تصویر نوید محمد زاده انجام شده است

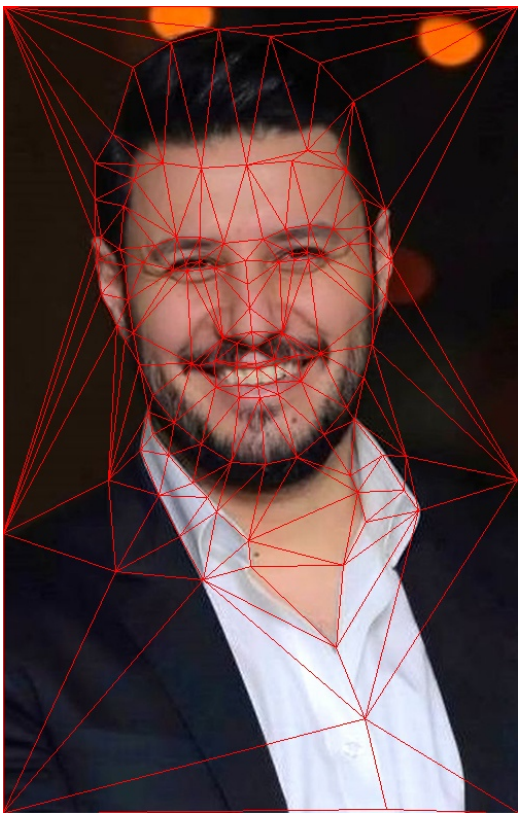
شکل ۴: در تصویر نوید محمدزاده مثلث بندی توسط نقاط تصویر نوید محمد زاده انجام شده است

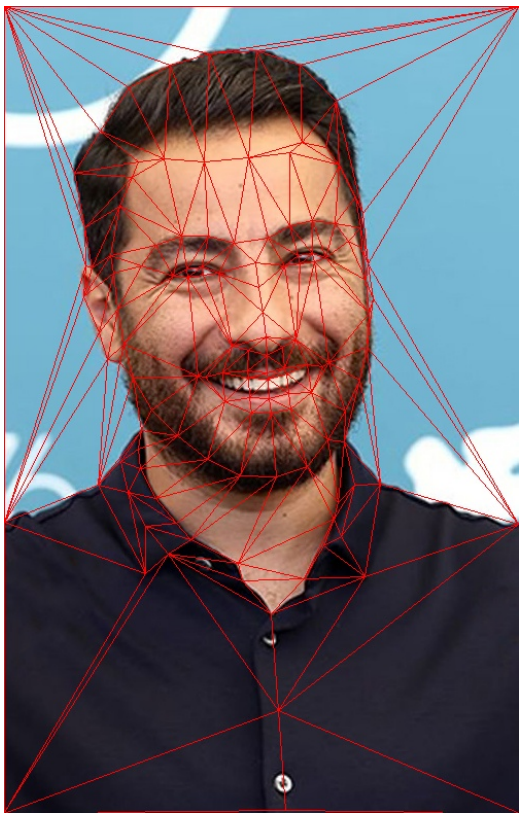
کل مثلث ها، با دادن سه نقطه ی مثلث اول و دوم به تابع getAffineTransform در opencv، نگاشت affine مربوط به همان مثلث را بدست بیاوریم. نگاشت حاصل را به در یک لیست میریزیم و سپس همین کار را نیز برای مثلث های دوم نسبت به اولی میکنیم و درحقیقت معکوس این نگاشت ها را در لیستی دیگر میریزیم.

حال دو ماتریس map۱ و map۲ را میسازیم. این دو ماتریس هم اندازه ی تصویر اصلی هستند، اما در هر درایه اش یک دوتایی قرار دارد. برای map۱ ، تک به تک روی همه ی پیکسل ها پیمایش میکنیم و با استفاده از تابع isInside، مثلثی که پیکسل در آن قرار دارد را پیدا کرده و با توجه به شماره ی مثلث، نگاشت مورد نظر را از ماتریس نگاشت ها (که در مرحله قبل بدست آوردیم) را بدست آورده و با ضرب در مختصات پیکسل در حال پیمایش، مکان پیکسل مقصد بدست می آید. این مختصات را در map۱ قرار میدهیم. سپس برای map۲ عینا همین کار را با لیست مثلث های دوم و لیست نگاشت های دوم انجام میدهیم و پیکسل های مقصد برای تصویر دوم نیز بدست می آید.

در مرحله بعد ابتدا یک آرایه سه بعدی به ابعاد تصویر که هر درایه آن مختصات آن درایه را نشان میدهد میسازیم (آرایه ی مختصات آغازین) ، سپس یک آرایه ۴ بعدی که در حقیقت ۴۵ تا ماتریس از نوع map۱ و map۲ را کنار هم گذاشته ایم میسازیم و با یک حلقه ساده با ۴۵ دور، در هر دور با توجه به شماره حلقه تقسیم بر ۴۴ که میتواند عددی بین ۰ تا ۱ باشد، لایه ی چهارم آرایه coefs۱ را میسازیم. به این منظور از یک درونیابی کلی استفاده میکنیم که ماتریس اولیه را با توجه به مقدار fr که مقداری از ۰ تا ۱ دارد، به آن وزن ۰ تا ۱ داده و ماتریس نهایی را به آن وزن یک منهای وزن قبلی میدهیم. به اینصورت برای لایه ی آخر همان مختصات عادی عکس اول را داریم و برای لایه ی اول مختصات هر پیکسل در مقصد. برای coefs۲ عکس این کار را انجام میدهیم و در نتیجه برای لایه آخر coefs۲ مختصات هر پیکسل در تصویر مقصد و برای لایه ی اول مختصات هر پیکسل در تصویر دوم را خواهیم داشت. در پایان این بخش یک ویدیو با سرعت و اندازه ی مورد نظر تعریف میکنیم که برای درست کردن گیف مورد خواسته ی سوال کاربرد دارد.

بعد از این بخش با توجه به مقادیری که ذخیره کرده ایم، وارد حلقه اصلی با ۴۵ دور میشویم که در انتهای این حلقه





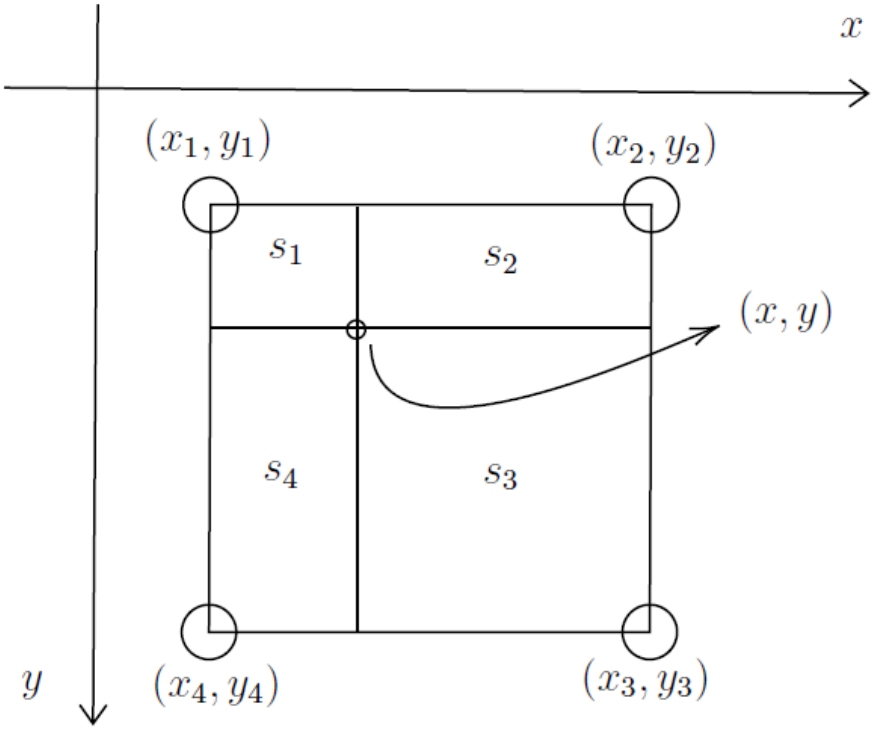
شکل ۵: در تصویر جواد عزتی مثلث بندی توسط نقاط تصویر جواد عزتی انجام شده است

شکل ۶: در تصویر نوید محمد زاده مثلث بندی توسط نقاط مثلثی شده گرفته شده از تصویر جواد عزتی انجام شده است

هر فریم تصویر بدست می آید. در اینجا ابتدا دو کسر به نام fr۱ و fr۲ میسازیم که هر کدام یک عدد اعشاری بین ۰ تا ۱ هستند. عدد fr۱ در دور اول ۱ و در دور آخر ۰ است و عدد fr۲ نیز در دور اول ۰ و در دور آخر ۱ است. سپس image۱ و image۲ را درون res ۱ و res۲ کپی میکنیم سپس هر هر ایتریشن به اندازه ی کل پیکسل ها پیمایش انجام میدهیم. در هر ایتریشن شماره ی آن را به بعد شما صفر coefs۱ داده و i و j پیکسل در حال پیمایش را به بعد ۱ و ۲ و سپس بر اساس ۰ یا ۱ بودن بعد بعد ۳ آن، آن را در متغیر موقتی ch یا cw ذخیره میکنیم. این شماره، به ما میگوید مقدار پیکسل در حال پیمایش را از کدام پیکسل تصویر اول باید بگیریم.

در ادامه با استفاده از ch و cw که میتوانند مقدار اعشاری داشته باشند، رنگی که قرار است به پیکسل در حال پیمایش انتقال بدهیم را با درونیابی بدست میاوریم. اول ۴ نقطه صحیح اطراف ch و cw را بدست میاوریم و استثنا ها را از بین میبریم. سپس مساحت هر قطعه از اطراف پیکسل را بدست میاوریم. حال رنگ هر قسمت را در مساحت مقابلش ضرب کرده و با هم جمع کرده و حاصل را گسسته کرده و در ماتریس res۱ قرار میدهیم که ماتریس نتایج موقتی عکس اول میباشد. (از آنجا که مجموع مساحت قطعات برابر ۱ میشود نیازی به نرمال کردن آن نخواهیم داشت)
در ادامه دقیقا همین کار را برای تصویر دوم با ماتریس ها و ضرایب مربوط به خودش انجام میدهیم و res۲ را نیز به همین ترتیب بدست میاوریم. حال res۱ را در fr۱ (که از ۱ تا ۰ کم میشد) و res۲ را در fr۲ (که از ۰ تا ۱ زیاد میشد)ضرب کرده و حاصل را با هم جمع کرده و بین ۰ تا ۲۵۵ کلیپ میکنیم. در ادامه خروجی های لازم را میگیریم و تصویر بدست آمده را در یک پشته (که لیست است) ذخیره میکنیم و بعد از تمام شدن این حلقه، ۲ حلقه تشکیل میدهیم که یکبار پشته را از شماره ۰ تا ۴۴ را به عنوان فریم ۱ تا ۴۵ درون ویدیو ذخیره کند و در حلقه دیگر پشته را از شماره ۴۵ تا ۸۹ به عنوان فریم های ۴۶ تا ۹۰ ذخیره کند و در پایان آنرا با فرمت mp۴ خروجی بدهد.

تصاویر فریم های ۱۵ و ۳۰ که در درو ۱۴ و ۲۹ ساخته میشوند و فایل gif مورد نظر به نام $morph.gif$ در پوشه ی نتایج قرار دارد. درون فولدر $q1 - ext$ که داخل فایل زیپ قرار دارد، کد انتخابگر نقاط، مختصات نقاط در تصویر های ۱ و ۲ ، و همچنین فایل با فرمت $mp4$ از gif مورد نظر که مستقیما توسط برنامه ساخته شده است قرار گرفته است.



شکل ۷: روش بدست آوردن وزن هر پیکسل بر اساسا مساحت قطعه ی روبرویش در این روش داریم:

$$g(x3, y3) = f(x3, y3).s1$$

$$g(x4, y4) = f(x4, y4).s2$$

$$g(x1, y1) = f(x1, y1).s3$$

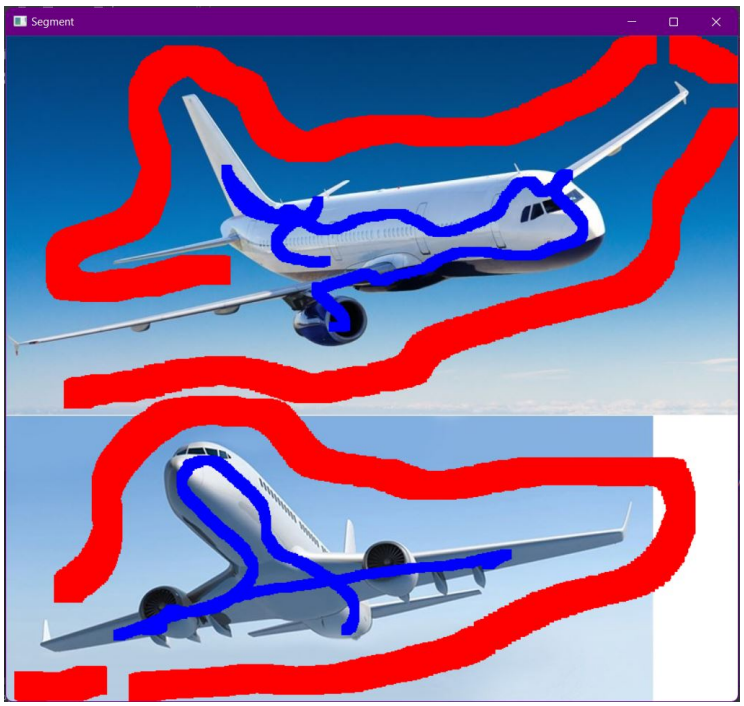
$$g(x2, y2) = f(x2, y2).s4$$

و مقدار نهایی برای هر کانال رنگی برابر میشود با: $f(x, y) = \sum_1^4 g(xi, yi)$

پرسش ۲ - Poisson Blending

در این سوال به عنوان تابع اصلی، تابع $q2$ را داریم. این تابع به عنوان ورودی منبع (یا فورگراند)، مقصد(یا بکگراند) ، ماسک، طول و عرض قرار گیری ، نسبت تصویر و سیگمای تابع گاوسی را دریافت میکند. از اینجا که در این سوال ۲ نمونه ماسک و بک گراند داشتیم، آن را با هم در یک تصویر ادغام کرده و در ورودی های تصویر آن ها را از هم تفکیک کردیم. بار اول تصویر یک هواپیما و ماسک و پس زمینه آن را به تابع سوال داده و در ادامه نتیجه ی مرحله اول را به عنوان پس زمینه ی مرحله دوم با هواپیما و ماسک متفاوت به تابع میدهیم.

برای بدست آوردن ماسک میتوان از کد درون فولدر $q2 - ext$ استفاده کرد که به نحوی اصلاح شده ی کد تمرین سری ۴ سوال ۴ است و از تابع graphcut استفاده میکند و mask را خروجی میدهد. یا هم از فایل $mask.jpg$ از پوشه ی $q2 - ext$ بطور مستقیم استفاده کرد که نتیجه ی خوبی را حاصل میکند. یک نمونه از سگمنت درست توسط کد فایل $Segment.py$ بصورت شکل زیر است.



روش کار این کد هم بصورت مختصر در ادامه آمده است:

۱. برای حالتی که روی حالت انتخاب شی هستیم، با حرکت دادن ماوس یک همسایگی ۵ در ۵ از محل حرکت 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 غیر فعال از حالت غیر فعال به شی مورد استفاده قرار میگیرد.

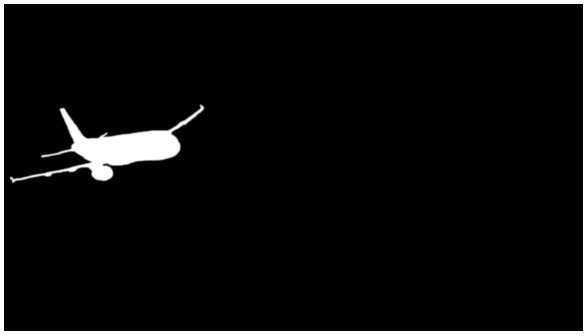
۳. حرکت ماوس، برای وقتی که در حالت پس زمینه قرار داریم، در این حالت با جابجا کردن ماوس، به همسایگی ۱۵ در ۱۵ محل حرکت ماوس ،مقدار ۰ به ماسک میدهد و یک مربع قرمز نیز به تصویر که باعث قرمز شدن آن بخش از تصویر میشود.

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

در تابع سوال، ابتدا ماسک دریافت شده را با نسبت sigma بصورت گاوسی بلر میکنیم. این کار باعث یکنواخت تر شدن حرکت از محیط به سمت بدنه ی هواپیما ها میشود. در ادامه نیز ماسک ها را با استفاده از تابع resize به نسبت rate تغییر اندازه میدهیم تا اندازه ی مورد نظر ما بدست بیاید. ماسک های جدید حاصل شده بعد از این فیلتر را در شکل ۸ و ۹ مشاهده میکنید.


شکل ۸: تصویر حاصل شده بعد از اعمال فیلتر گاوس در تصویر اول

شکل ۹: تصویر حاصل شده بعد از اعمال فیلتر گاوس در تصویر دوم




شکل ۱۰: تصویر حاصل بعد از شیفت دادن ماسک اول

شکل ۱۱: تصویر حاصل بعد از شیفت دادن ماسک دوم

سپس با توجه به مختصات دریافت شده در تابع برای قرار گیری تصویر مبدا، ماسک ها و تصاویر مبدا را شیفت میدهیم و آنها را در جای درست خودشان قرار میدهیم.
حال این از طریق این دو ماسک، ماسک مخالف آن ها را ایجاد میکنیم.


شکل ۱۲: ماسک مخالف برای ماسک تصویر اول

شکل ۱۳: ماسک مخالف برای ماسک تصویر دوم

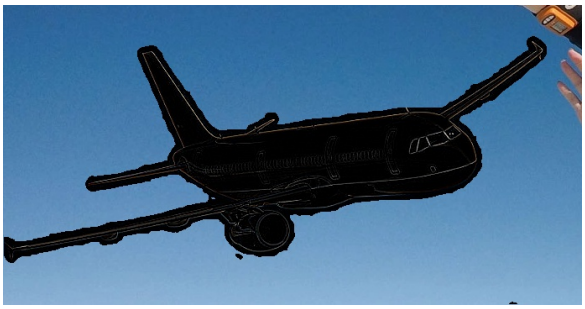
حال هم ماسک بکگراند هم ماسک فور گراند را بر ۲۵۵ تقسیم میکنیم که مقادیر آن بین ۰ تا ۲۵۵ قرار گیرد و مناسب استفاده باشد. حال تصویر مبدا را نیز به اندازه rate تغییر اندازه میدهیم که مناسب با ماسک باشد. سپس به اندازه ی تصویر مبدا از تصویر مقصد جدا کرده و هم ۳ کانال رنگی این تصویر جدا شده و هم تصویر سورس را با استفاده از تابع split از هم جدا میکنیم.

در این مرحله، بنابر الگوریتم، نیاز است ماتریس ضرایب را با استفاده از اعمال لاپلاسین بسازیم. به این منظور روی ماسک پیمایش میکنیم و هرجا مقدار غیر صفر بود، روی ۳ تا کانال قطعه ی جدا شده از تصویر مقصد، تابع لاپلاسینی که

کرنل زیر را دارد، روی همان پیکسل از تصویر مبدا (در پیکسلی که در مکان متناظر پیکسل غیر ۰ ماسک) اعمال کرده و نتیجه را برابر با پیکسل متناظر از پس زمینه اعمال میکنیم.

$$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix}$$

پس از این فرایند، ۳ کانال رنگی که امکان دارد مقادیر بسیار بزرگ یا حتی منفی داشته باشند، را ماتریس ضرایب نامیده و آن را برای استفاده ی بعدی نگه میداریم. این ماتریس های ساخته شده بصورت کلیپ شده، بصورت شکل های ۱۴ و ۱۵ هستند.



شکل ۱۴: ماتریس ضرایب کلیپ شده تصویر اول

شکل ۱۵: ماتریس ضرایب کلیپ شده تصویر اول

در ادامه با توجه به اسلاید جلسه ۲۵، ماتریس مورد نظر را با استفاده از قطر ها میسازیم. (فرض میکنیم تئوری ساخت این ماتریس را میدانیم.) برای ساخت قطر اصلی روی کل پیکسل ها در ماسک تغییر اندازه در جهت بالا به پایین پیمایش کرده و هرجا پیکسل ۰ داشتیم مقدار قطر اصلی را ۱ و هرجا مقدار غیر ۰ داشتیم قطر اصلی را ۴ قرار میدهیم. در عین حال که داریم مقادیر ۱ و ۴ قرار میدهیم، در آرایه تک بعدی قطر فرعی نیز عینا همین کار را با مقادیر ۰ و ۱- انجام میدهیم. به اینصورت که هرجا مقدار ماسک ۰ بود مقدار قطر فرعی را ۰ قرار داده و هرجا غیر صفر بود، این مقدار را ۱- قرار میدهیم. در ادامه نیز با توجه به ساختار قطر ها و شماره گذاری آنها، قطر فرعی بدست آمده را ۳ بار درون ۳ قطر فرعی دیگر کپی میکنیم و به ترتیب این قطر هارا یک شیفت منطقی به سمت چپ، یک شیفت منطقی به سمت راست، h (ارتفاع عکس) واحد شیفت منطقی به سمت چپ و h شیفت منطقی به سمت راست میدهیم. با به هم چسباندن این ۵ قطر در یک راستای جدید، یک ماتریس ۲ بعدی خواهیم داشت که مقادیرشان ۰ و ۱ و ۴ و ۱- هستند. حال یک ماتریس تک بعدی ۵ تایی نیز شامل h- و ۱- و ۰ و ۱ و h میسازیم. این ۵ عدد بنابر ترتیب قرارگیری قطر ها در ماتریس ۲ بعدی که قبلا ساخته ایم مرتب میشوند.

حال با دادن این دو ماتریس و تعداد پیکسل ها به تابع sparse.spdiags ، یک ماتریس sparse بسیار بزرگ با طول و عرض تعداد پیکسل ها تولید میکنیم. در مرحله بعدی این sparse و ماتریس ضرایب که پیشتر ساخته ایم را تبدیل به ماتریس csr میکنیم. پس از آنکه ماتریس ضرایب را با اوردر 'F' (یعنی پیشروی ایندکس گذاری آن بصورت عمودی است) به عرض ۱ و ارتفاع تعداد پیکسل ها تغییر شکل میدهیم، میتوانیم با دادن ماتریس ساخته شده به طول و عرض تعداد پیکسل های ناحیه و ماتریس ضرایب تغییر شکل یافته به تابع linalg.spsolve و تغییر شکل آن با اوردر'F و همچنین به طول و عرض ناحیه، ماتریس پاسخ را بدست بیاوریم. حال همین کار را برای ۳ کانال رنگی میکنیم و ۳ کانال رنگی پاسخ را بدست میاوریم. با ادغام این ۳ کانال رنگی با هم و تولید یک تصویر rgb ، تصاویر شکل ۱۶ و ۱۷ حاصل میشوند.

حال اگر کمی با دقت به شکل ۱۵ نگاه کنیم، میبینیم که یک هاله ی مات دور بدنه وجود دارد که این خاصیت خود این روش است. در اینجا ماسک موجود در شکل ۱۱ و ۱۲ به کارمان میاید. به این صورت که این ماسک اطرافش کمی محو تر و جمع تر هست. اگه ما این ماسک را روی تصویر اعمال کیم، نتیجه ی خوبی خواهیم گرفت، زیاد که به آرامی از داخل تر از الان وارد تصویر میشویم و در عمل متوجه مات شدگی اطراف تصویر نمیشویم، چرا که تصویر پس زمینه اصلی همپوشانی کاملی با تصویر مات نیز دارد و در آن ناحیه که ضریبی از هر دو حضور دارد، اصلا متوجه نمیشیم که دور تصویر مات شده است پس این تصاویر را در ماسک های بین ۰ تا ۱ که ساخته بودیم ضرب میکنیم. سپس با ساخت تصاویر با سایز اصلی و قرار دادن این تصاویری که ماسک هم شده اند، شکل ۱۸ و ۱۹ را خواهیم داشت.

و از تصویر مقصدی که بصورت مناسب ماسک شده است هم شکل ۲۰ و ۲۱ را خواهیم داشت.

شکل ۱۶: نتیجه بدست آمده برای تصویر اول

شکل ۱۷: نتیجه بدست آمده برای تصویر دوم

شکل ۱۸: نتیجه برای تصویر ماسک شده از مبدا اول

شکل ۱۹: نتیجه برای تصویر ماسک شده از مبدا دوم

شکل ۲۰: نتیجه برای تصویر ماسک شده از مقصد برای مبدا اول

شکل ۲۱: نتیجه برای تصویر ماسک شده از مقصد برای مبدا دوم

دلیل اینکه در تصویر دوم هواپیمای اول وجود دارد، این است که همانطور که گفته شد تصویر نهایی حاصله از مرحله ی اول (که در آن هواپیمای اول را میگذاشتیم) را به عنوان مقصد مرحله دوم داده است.

با ترکیب تصاویر ماسک بصورت مناسب در شکل ۱۹ و ۲۱، تصویر نهایی موجود در نتایج بدست می آید.

شکل ۲۲: ماسک اعمال شده در اولین پله از لاپلاسین استک

پرسش ۳ - Multiresolution Blending and Feathering

برای نمونه ی این سوال، ترکیب یک آناناس و دوریان استفاده شد. [فایل $mask.jpg$ هم که برای اجرای برنامه لازم است، در فولدر $q3 - ext$ قرار دارد.] تابع اصلی این سوال، تابع $q3$ است. پارامتر هایی که این تابع میگیرد تصویر اول، تصویر دوم، ماسک، تعداد ایتریشن ها، a و b و c هستند که در ادامه در مورد آنها توضیح خواهد داده شد. پس ازینکه تصویر اول و تصویر دوم و ماسک را وارد برنامه کردیم، این تصاویر و اعداد را به تابع فوق میدهیم. درون تابع در ابتدا یک لیست به نام استک پلپلاسین تعریف میکنیم و از این به بعد آن را به نام استک میشناسیم. حال مراحل زیر را قبل از شروع حلقه انجام میدهیم.

در ابتدا دو تصویر ورودی را تبدیل به int میکنیم. سپس هر دو تصویر را در im۱_optimized و im۲_optimized کپی میکنیم. حال روی ماسک اولیه یک فیلتر گاوسی اعمال میکنیم که این شیب اولیه ی تلفیق شدن دو تصویر را تعیین میکند، زیرا برای ترکیب لاپلاسین تصویر اولی مورد استفاده قرار میگیرد. اگر این مقدار خیلی کوچک باشد مرز دو تصویر خیلی معلوم میشود. در حالت پیشرفض این مقدار ۳ است، ما این ماسک را در mask۱_optimized ذخیره میکنیم. میتوانید ماسک حاصل از این blur را در شکل ۲۲ ببینید.

در ادامه از کم کردن این ماسک از ۲۵۵، ماسک وارون این ماسک بدست می آید. و آن را در mask۲_optimized ذخیره میکنیم. سپس با ادغام این دو ماسک در ۳ کانال رنگی توسط تابع merge در opencv، آن ها را برای استفاده های بعدی آماده میکنیم. سپس با تقسیم این دو ماسک سه کاناله بر ۲۵۵ omask۱_binary و omask۲_binary را میسازیم. حال دو ماتریس به نام level_last داریم که تصویر مرحله قبل را ذخیره میکند. از آنجا که هنوز وارد مرحله اول نشدیم مقدار این دو آرایه را برابر با با تصاویر اولیه قرار میدهیم. در ادامه وارد یک حلقه میشویم که تعداد دور هایش برابر با همان مقدار levels از ورودی تابع است.

در ابتدای حلقه یک متغیر به نام gauss با رابطه زیر تعریف میکنیم.

$$gauss = (layer + 1) \times a$$

در اینجا layer شماره ی دوری است که در آن قرار داریم و a ورودی تابع اصلی برنامه بود. (اندازه کل کرنل های گاوسی ما برابر (۲۱۵، ۲۱۵) میبشد.) در ایمجا با مقدار سیگمای gauss روی تصویر های اول و دوم فیلتر گاوسی میزنیم و نتیجه را در im_optimized ها قرار میدهیم. سپس از کم کردن این تصویر از تصویر مرحله قبل، لاپلاسین این تصویر را بدست آورده

شکل ۲۳: قالب تصویر (تصاویر ترکیب شده در آخرین مرحله)

و آن را در $l1$ و $l2$ قرار میدهیم. حال با ضرب این لاپلاسین ها در همان omask_binary هایی که قبل از شروع شدن حلقه هم بدست آورده بودیم، تصویر آماده ی ترکیب شدن را بدست میاوریم. مجموع این دو تصویر را در ابتدای استک میریزیم و دقیقا همان کار هایی که قبل از شروع شدن حلقه کرده بودیم را تکرار میکنیم با یک تفاوت کوچک، اینبار بجای اعمال فیلتر گاوس روی ماسک اصلی با سیگمای c، آن را با سیگمای $gauss \times b$ اعمال میکنیم. در ادامه ماسک ۲ را میسازیم و آنهارا ۳ کاناله کرده و omask_binary های جدید را بدست میاوریم. در پایان نیز تصاویری که در این مرحله رویشان فیلتر گاوس اعمال شده بود یعنی im۱_optimized و im۲_optimized را به ترتیب در level۱_last و level۲_last کپی میکنیم. (دلیل اینکه در ابتدای حلقه این کار هارا نکردیم، کاری است که ما با ماسک ها و تصاویر مرحله ی آخر در پایان حلقه داریم.)

وقتی حلقه به تعداد levels انجام شد و تمام شد، تصاویر گاوسی شده مرحله آخر را یعنی im۱_optimized و im۲_optimized که در انتهای آخرین حلقه بدست آمده اند و نیز omask۱_binary و omask۲_binary متناظر را به ترتیب در هم ضرب میکنیم. (im۱۱ در omask۱ و im۲ در omask۲) حال با ترکیب این دو نتیجه باهم، قالب ما تشکیل میشود. قالب تصویر نتیجه را در شکل ۲۳ مشاهده میکنید.
پس از آن به اندازه مراحلی که طی کردیم، تصویر لاپلاسین را از انتهای پشته pop میکنیم (از انتها به ابتدا میاییم) وبا قالب جمع میکنیم. در پایان نیز نتیجه را بین ۰ تا ۲۵۵ کلیپ کرده و به عنوان خروجی تابع برگردانده و در بخش اصلی کد نتیجه را ذخیره میکنیم.

نکته مهمی که وجود دارد، این است که a و b و c را مناسب بگیریم، زیرا اگر این سه را کوچکتر از حد بگیریم، مرز بین دو تصویر واضح میماند، رنگ دو تصویر با هم ترکیب نمیشوند و مواردی از این دسته. اگر این سه مقدار را نیز خیلی بزرگ بگیریم، مرز خیلی به آرامی حرکت میکند و نمای زیبایی ندارد، و علیرغم اینکه رنگ آن رو به خوبی با هم ترکیب میشوند، رنگ ها از سوژه امکان دارد خارج بشوند. در شکل های ۲۴ و ۲۵، نمونه هایی از کوچک و بزرگ قرار دادن a و b و c را مشاهده میکنیم.
نتیجه ی مناسب (موجود در کیفیت بالا در فایل پاسخ) هم در ادامه قرار میگیرد که تفاوت آن با این دو مورد به خوبی قابل مشاهده باشد.

شکل ۲۴: نتیجه در صورتی که a و b و c کم باشند

شکل ۲۵: نتیجه در صورتی که a و b و c زیاد باشند