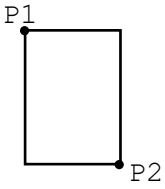
q2

در این بخش یک کلاس پیاده سازی شده که ابتدا توضیحشان میدهیم.

کلاس `Box` :



ویژگی ها:
1) مختصات راس بالا سمت چپ
2) مختصات راس پایین سمت راست

استفاده: برای کشیدن قاب های دور موارد پیدا شده در تصویر به مختصات آنها نیاز داریم که برای سهولت هر قاب را بعنوان یک شئ از کلاس `Box` نگه میداریم. یک تابع نیز در این کلاس بنام `resize` تعریف شده است که با صدا زدن آن برای هر شئ با ورودی `a`، مختصات نقاط این شئ `a` برابر میشود. جلوتر به کاربرد آن خواهیم پرداخت. در ادامه‌ی توضیحات منظور از **p1** و **p2** برای هر قاب همین نقاط مشخص شده در شکل خواهند بود.

در این بخش سه تابع پیاده سازی شده که ابتدا توضیحشان میدهیم.

تابع `drawBoxes` :

ورودی ها:
1) یک تصویر
2) یک لیست از قاب ها که اعضای آن شئ‌ای از جنس `Box` اند

خروجی ها:
1) تصویری که تمام قاب های داده شده بر روی تصویر اصلی رسم شده اند

عملکرد: برای این کار به سادگی روی تمام اعضای لیست قاب ها `for` زده شده و هر قاب با استفاده از تابع آماده‌ی `cv2.rectangle` بر روی تصویر داده شده رسم شده و تصویر حاصل بعنوان خروجی بازگشت داده شده.

تابع `matchTemplate` :

ورودی ها:
1) یک تصویر بزرگ
2) یک تصویر کوچک بعنوان patch
3) یک لیست از قاب ها که اعضای آن شئ‌ای از جنس `Box` اند

خروجی ها:
1) یک لیست از قاب ها که اعضای آن شئ‌ای از جنس `Box` اند و نمونه های یافت شده‌ی جدید به آن اضافه شده اند

عملکرد: ابتدا تصویر patch را در متغیر g نگه میداریم و میانگین درایه های آن را محاسبه کرده و در متغیر gm نگه میداریم. اکنون میخواهیم patch داده شده را روی تصویر اصلی حرکت دهیم و مختصات قابی که در آن، همبستگی یا correlation بین patch و تصویر بیش از 0.78 بود را در لیست قاب ها نگه داریم. برای این کار از دو for تو دو تو استفاده کرده تا patch بتواند در هر دو راستای عمودی و افقی روی تصویر حرکت کند و در هر نقطه، قابی به ابعاد patch از تصویر اصلی را برش میدهیم و آن را در متغیر f نگه میداریم. سپس میانگین آن را محاسبه کرده و در متغیر fm نگه میداریم. سپس با استفاده از فرمول زیر همبستگی بین patch و برشی از تصویر اصلی را در این نقطه محاسبه میکنیم.

$$corr = \frac{\sum(g - gm)(f - fm)}{\sqrt{\sum(g - gm)^2 \sum(f - fm)^2}}$$

و اگر این مقدار بیشتر از 0.78 بود، یک قاب با مختصات برش یافته از تصویر اصلی ساخته و آن را به لیست قاب ها اضافه کنیم. در انتها یک لیست از قاب هایی داریم که تعدادی قاب جدید به آن اضافه شده. که مختصات هر قاب نشان دهنده این است که آن مختصات یک مورد بسیار مشابه به patch یافت شده است. سپس این لیست قاب ها را بعنوان خروجی بازگشت میدهیم.

تابع removeExtraBoxes :

ورودی ها: 1) یک لیست از قاب ها که اعضای آن شئ‌ای از جنس Box اند

خروجی ها: 1) یک لیست از قاب ها که اعضای آن شئ‌ای از جنس Box اند و نمونه های تکراری آن حذف شده اند.

عملکرد: همانطور که میدانیم در این روش template matching ممکن است تعداد زیادی قاب دور تنها یک مورد یافت شده رسم میشود. پس هدف این است که از تکرار قاب هایی که هردو نشان دهنده‌ی یک مورد مشابه یافت شده اند جلوگیری کنیم و لیست قاب ها را پاک سازی کنیم تا به ازای هر نمونه، تنها یک قاب دور آن موجود باشد. برای این کار از اجتماع گیری استفاده میکنیم و هر قاب از لیست داده شده را با قاب های دیگر مقایسه میکنیم. اگر دو شرطی که در پایین ذکر شده اند را دارا بودند، یکی از آن دو قاب اضافی بوده و آنگاه کوچک ترین قاب ممکن که هر دو قاب را بر میگیرد را رسم میکنیم. دو شرط لازم این گونه اند که اولا اگر دو قاب داده شده کاملا زیر هم و نزدیک به هم بودند، دوما اگر قابی درون قاب دیگر بود، قاب جدید دور آنها رسم میکنیم و یکی از آن دو قاب را از لیست حذف میکنیم تا

محاسبات کمتر شود. تشخیص دو شرط ذکر شده با استفاده از مختصات همان دو نقطه ای است که در کلاس Box بالاتر توضیح داده شد. یعنی نقطه‌ی بالا سمت چپ قاب و نقطه پایین سمت راست قاب.

شرط اول معادل با این است که مولفه‌ی راستای عمودی دو نقطه‌ی p1 و p2 در هر دو قاب برابر باشند.

شرط دوم معادل با این است که هر دو مولفه‌ی نقطه‌ی p1 قاب اول، از هر دو مولفه‌ی نقطه‌ی p1 قاب دوم کمتر باشد و هر دو مولفه‌ی نقطه‌ی p2 قاب اول، از هر دو مولفه‌ی نقطه‌ی p2 قاب دوم بیشتر باشد.

دو شرط بالا برای هر دو قاب، دو به دو چک میشود پس از دو for تو در تو استفاده میکنیم. ولی این کار زمان بر نیست، زیرا در هر مرحله هر قابی که بعنوان قاب اضافی تشخیص داده میشود از آرایه حذف شده و دیگر آن را چک نمیکنیم.

ساختن قابی که دو قاب را در بر بگیرد، در هر یک از دو شرط بالا واضح است. در هر دو شرط مولفه های نقطه‌ی p1 قاب جدید از مینیمم مولفه های نقطه‌ی p1 دو قاب گرفته میشوند و مولفه های نقطه‌ی p2 قاب جدید از ماکزیمم مولفه های نقطه‌ی p2 دو قاب گرفته میشوند. حال با داشتن مولفه های نقاط p1 و p2 قاب جدیدی ساخته و آن را به آرایه‌ی جدیدی از قاب ها بنام optimalBoxes اضافه میکنیم که در این آرایه جدید هیچ دو قابی دو شرط بالا را ندارند. پس این آرایه‌ی جدید را بعنوان خروجی بازگشت میدهیم.

حال پس از توضیحات مربوط به توابع پیاده سازی شده به توضیحات روند استفاده از آنها و رسیدن به خروجی مطلوب میپردازیم.

ابتدا کمی از اطراف patch داده شده را برش میدهیم تا دریا، بخش غالب patch نباشد تا دچار خطا نشویم. ابعاد هر دو تصویر اصلی و تصویر patch را تقسیم بر a=8 کرده تا ابعاد آن کاهش یابد زمان اجرای برنامه تسریع شود. برای آنکه در چند سایز مختلف در تصویر اصلی چک شود از هرمی سازی استفاده میکنیم به طوری که در هر گام ابعاد patch داده شده را h=7/8 برابر ابعاد مرحله‌ی قبلش میکنیم. و این کار را در چهار مرحله انجام میدهیم. یعنی هرم ما چهار مرحله ای است. در ابتدا آرایه ای به نام boxes از قاب ها که در نهایت باید آنها را رسم کنیم، میسازیم. در هر مرحله از هرم تصویر کشتی و تصویر میله و آرایه‌ی boxes را به تابع matchTemplate که بالاتر تعریف شده، میدهیم تا قاب های یافت شده‌ی جدید به آرایه‌ی boxes اضافه شوند. قاب های جدید به معنای پیدا شدن موارد جدید اند که شبیه به میله میباشند. پس از اجرام هر چهار مرحله‌ی هرم، ملاحظه میکنیم

که چند قاب مجزا برای یک میله‌ی یکسان یافت شده که اضافی هستند. پس آرایه‌ی boxes را به تابع removeExtraBoxes میدهیم تا قاب های اضافی از آرایه حذف گردند. در نهایت عکس را به ابعاد اصلی خودش برمیگردانیم و تمام قاب های باقی مانده را باید در ابعاد و کیفیت اصلی عکس رسم کنیم. از آنجا که ابعاد عکس را در ابتدا 1/8 کردیم و مختصات قاب ها را در آن ابعاد بدست آوردیم، پس نیاز است تا مختصات قاب ها را ضرب در a=8 کنیم تا پس از رسم آن ها در عکس اصلی در جای درستی قرار گیرند. 8 برابر کردن ابعاد قاب ها با استفاده از تابع resize که در کلاس Box در بالا تعریف شد، انجام میشود. در نهایت با استفاده از تابع drawBoxes تمام قاب های باقی مانده در آرایه‌ی boxes را بر روی تصویر رسم میکنیم. از آنجا که اندازه‌ی قاب های رسم شده متفاوت اند، نشان میدهد که هرمی سازی به پیدا کردن میله های کوچک نیز کمک کرده است.

تمام عدد ها که در این سوال در نظر گرفته شده (از قبیل تعداد گام های هرم، آستانه‌ی همبستگی 0.78 و برش ابعاد patch) به یکدیگر وابسته هستند و به صورت تجربی با هم هماهنگ شده اند.

q3

در این بخش چهار تابع پیاده سازی شده که ابتدا توضیحشان میدهیم.

تابع shape :

ورودی ها: 1) چهار نقطه که راس های یک چهار ضلعی اند

خروجی ها: 1) طول و عرض یک مستطیل که چهار ضلعی مورد نظر در آن قرار میگیرد

عملکرد: هدف این است که طول و عرض کتاب را به کمک مختصات چهار راس انتخاب شده از آن پیدا کنیم. برای این کار فاصله‌ی یکی از این راس‌ها را تا سه راس دیگر محاسبه میکنیم و در آرایه‌ی dists نگه میداریم. آن فاصله‌ای که از همه بیشتر است طول قطر چهار ضلعی را به ما میدهد که مورد نیاز ما نیست پس آن را از آرایه حذف میکنیم. اکنون دو عدد در آرایه باقی مانده است که عدد کوچکتر اندازه‌ی عرض و عدد بزرگتر اندازه‌ی طول کتاب را به ما میدهد که آن‌ها را بعنوان خروجی بازگشت میدهیم.

تابع findHomography :

ورودی ها: 1) چهار نقطه که راس های یک چهار ضلعی اند

خروجی ها: 1) ماتریسی هموگرافی‌ای که تحت آن، کتاب نگاشت پیدا میکند و از رو به رو به صورت صاف در گوشه بالا سمت چپ تصویر قرار میگیرد

عملکرد: هدف این است که کتاب انتخاب شده، از یک مختلف‌الاضلاع به یک مستطیل نگاشت پیدا کند بطوری که کتاب با این نگاشت، از رو به رو و به صورت صاف قرار گیرد. برای این کار ابتدا با استفاده از مختصات رئوس داده شده طول و عرض کتاب را به کمک تابع `shape` که در بالا تعریف شد، محاسبه میکنیم. اکنون چهار نقطه‌ی جدید میسازیم که مختصات آنها برابر با مختصات چهار راس یک مستطیل به طول و عرض بدست آمده است. حال چهار نقطه مختلف‌الاضلاع اولیه و چهار نقطه مستطیل هدف را به تابع آماده‌ی `cv2.findHomography` میدهیم تا ماتریس مورد نیاز برای نگاشت نظیر به نظیر چهار نقطه‌ی عکس مبدا به چهار نقطه‌ی مقصد را به ما بدهد. اکنون این ماتریس را بعنوان خروجی بازگشت میدهیم.

تابع `warp` :

ورودی ها:
1) عکس اصلی که کتابی در آن قرار دارد
2) ماتریسی هموگرافی‌ای که تحت آن، کتاب نگاشت پیدا میکند و از رو به رو به صورت صاف در گوشه بالا سمت چپ تصویر قرار میگیرد
3) طول و عرض کتابی که میخواهیم نگاشت دهیم

خروجی ها:
1) تصویر جلد کتاب از رو به رو و به صورت صاف

عملکرد: هدف این است که کتاب نگاشت یافته را داشته باشیم. میدانیم اگر روی کل عکس نگاشت ماتریس هموگرافی داده شده را اعمال کنیم آنگاه کتاب مورد نظر ما از رو به رو به صورت صاف در گوشه بالا سمت چپ تصویر خروجی قرار میگیرد. ولی این کار دو ایراد اساسی دارد. اول آنکه ابعاد عکس اصلی بسیار بزرگ است و اعمال نگاشت روی کل آن زمان زیادی میگیرد. دوما با اثر دهی این نگاشت برخی پیکسل ها در تصویر خروجی خالی میمانند. پس از ایده‌ی وارون نگاشت استفاده میکنیم. در ابتدا یک عکس به ابعاد طول و عرض کتاب، که بعنوان ورودی به تابع داده شده، میسازیم و آن را در متغیر `imgWarped` نگه میداریم. اکنون میخواهیم ببینیم که هر پیکسل از این تصویر چه مقداری از عکس اصلی را داشته باشد تا تصویر جلد کتاب تماماً در این قاب قرار گیرد. این گونه دو ایراد وارد بر روش قبلی حل میشود زیرا `for` ها روی پیکسل های عکس `imgWarped` زده میشوند و چون ابعاد این عکس کوچک است این روند سریع انجام میشود. از طرفی ما در نهایت تک تک پیکسل های این عکس را مقدار دهی کرده ایم پس پیکسل خالی نخواهیم داشت. این کار را با وارون گیری از نگاشت هموگرافی `H` که به تابع داده شده است انجام میدهیم. وارون ماتریس نگاشت `H` را در متغیر `iH` نگه میداریم سپس با

for زدن روی تک تک پیکسل های imgWarped در هر چنل مقدار مطلوب را بدست می آوریم. برای این کار از فرمول زیر استفاده شده است.

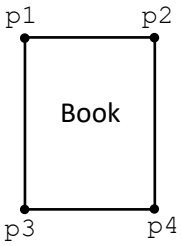
$$H^{-1}\begin{bmatrix} i \\ j \\ 1 \end{bmatrix} = \begin{bmatrix} i' \\ j' \\ a \end{bmatrix} \xrightarrow{\div a} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

که این یعنی هر نقطه با مختصات i و j در تصویر imgWarped باید مقدار نقطه‌ی با مختصات x و y در تصویر اصلی را داشته باشد. ولی ممکن است مقادیر x و y اعشاری باشند که تعریف نشده میشوند. برای حل کردن این مشکل، بخش اعشاری x را متغیر a ذخیره کرده و خود x را به عدد صحیح تبدیل میکنیم. به طور مشابه بخش اعشاری y را در متغیر b ذخیره کرده و خود y را به عدد صحیح تبدیل میکنیم. اکنون مشابه درونیابی از فرمول زیر برای بدست آوردن مقدار مطلوب استفاده میکنیم.

$$f(x+a, y+b) = \begin{bmatrix} 1-a & a \end{bmatrix} \begin{bmatrix} f(x,y) & f(x,y+1) \\ f(x+1,y) & f(x+1,y+1) \end{bmatrix} \begin{bmatrix} 1-b \\ b \end{bmatrix}$$

که در اینجا $f(x,y)$ مقدار سطر x ام و ستون y ام درماتریس img مباشد. به عبارت دیگر یعنی همان img[x,y,ch] است. اکنون تمام داریه‌های imgWarped در هر سه چنل مقداردهی شده و آن را بعنوان خروجی بازگشت میدهیم.

تابع findBook :



ورودی ها:
1) یک عکس که شامل کتاب باشد
2) مختصات راس بالا سمت چپ کتاب
3) مختصات راس بالا سمت راست کتاب
4) مختصات راس پایین سمت چپ کتاب
5) مختصات راس پایین سمت راست کتاب

خروجی ها:
1) عکس جلد کتاب از رو به رو و به صورت صاف

عملکرد: در ابتدا با استفاده از تابع findHomography که بالاتر نحوه‌ی پیاده‌سازی آن گفته شد، ماتریس هموگرافی مطلوب را محسبه میکنیم. سپس ابعاد کتاب را با استفاده از تابع shape که بالاتر شرح داده شد محاسبه میکنیم. سپس عکس اصلی که شامل کتاب است را به همراه ماتریس

هموگرافی و ابعاد کتاب، بعنوان ورودی به تابع `warp` که خودمان پیاده سازی کردیم، پاس میدهیم و عکس جلد کتاب را از رو به رو و مستقیم بدست میاوریم و آن را بعنوان خروجی بازگشت میدهیم.

حال پس از توضیحات مربوط به توابع پیاده سازی شده به توضیحات روند استفاده از آنها و رسیدن به خروجی مطلوب میپردازیم.

مختصات رئوس هر سه کتاب را به صورت دستی و به ترتیب وارد کرده و برای هر کتاب، عکس اصلی `books.jpg` را به همراه نقاط به تابع `findBook` میدهیم و نتیجه را ذخیره میکنیم.