گزارش تمرین سری چهارم

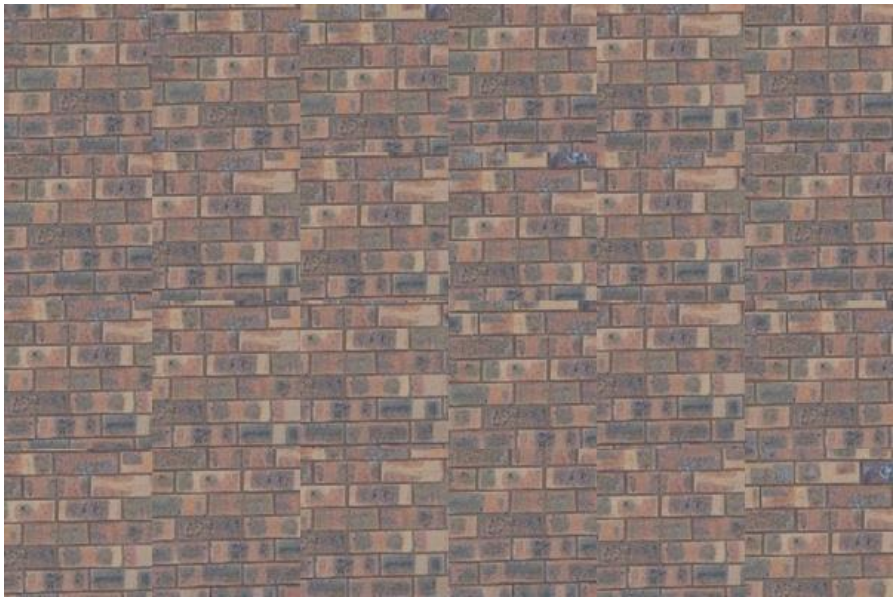
اصول پردازش تصویر

رضا اکبریان بافقی - ۹۵۱۰۰۰۶۱

۱

از این سوال تا سوال ۳ از شکل ۱ به عنوان بافت استفاده کرده‌ام که اندازه آن ۱۵۰ در ۱۵۰ می‌باشد. در این سوال در هر مرحله پنجره‌ای به اندازه ۱۰۰ در ۱۰۰ به‌صورت رندوم انتخاب می‌شود و به‌ترتیب درون عکسی به اندازه ۴۰۰ در ۶۰۰ قرار می‌دهم.

شکل ۱ بافت



شکل۲ گزینش رندوم بافت در کنار هم

۲

در این سوال اندازه پنجره انتخابی را ۱۰۰ در ۱۰۰ در نظر می‌گیرم و میزان هم‌پوشانی را ۲۰ پیکسل در نظر گرفته‌ام. برای افزایش سرعت انتخاب پنجره جدید از بافت، در هر مرحله ۴۰۰ بار به صورت رندوم یک پنجره انتخاب می‌شود و در نهایت پنجره‌ای که کمترین SSD را دارا بود را انتخاب می‌کنیم. بدین صورت هم رندوم‌نس رعایت شده‌است. هم این‌که پنجره‌ای با هم‌پوشانی خوب انتخاب شده‌است. نتیجه را می‌توانید در شکل ۳ ببینید. اندازه این تصویر هم ۴۰۰ در ۶۰۰ می‌باشد.

شکل ۳ انتخاب پنجره‌های با SSD نزدیک به قسمت هم‌پوشانی

در واقع برای این‌که پنجره جدید را قرار دهیم سه حالت وجود داشت که در شکل ۴ قابل مشاهده است. اولین پنجره را به‌صورت رندوم انتخاب می‌کنم و باقی پنجره‌ها بسته به اینکه اولین پنجره چه چیزی است و در کدام یک از حالت‌های شکل ۴ هستیم. پنجره بعدی انتخاب می‌شود.

شکل ۴ سه حالتی که برای محاسبه SSD و همچنین قرار دادن در عکس نتیجه ممکن است رخ بدهد.

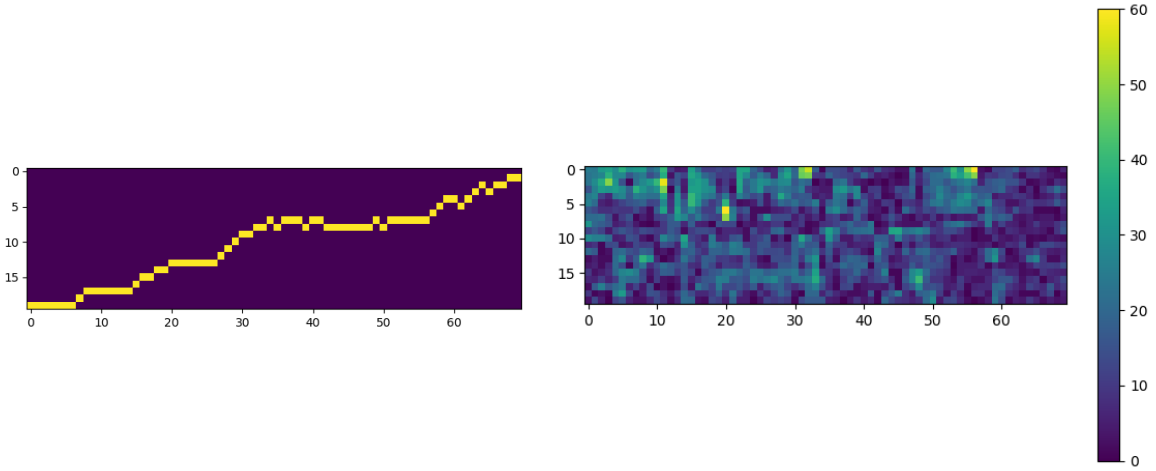
۳

این قسمت را با مطالعه مقاله زیر پیاده‌سازی کرده‌ام:

https://www.ipol.im/pub/art/2017/171/article.pdf

در این قسمت با استفاده از Dynamic Programming بهترین برشی که در قسمت هم‌پوشانی به‌وجود می‌آید را نتیجه می‌گیریم. مانند قسمت قبل سه حالت ممکن است رخ بدهد. حالت افقی، عمودی و L-شکل. حالت افقی را توضیح می‌دهم باقی حالت‌ها شبیه به هم هستند.

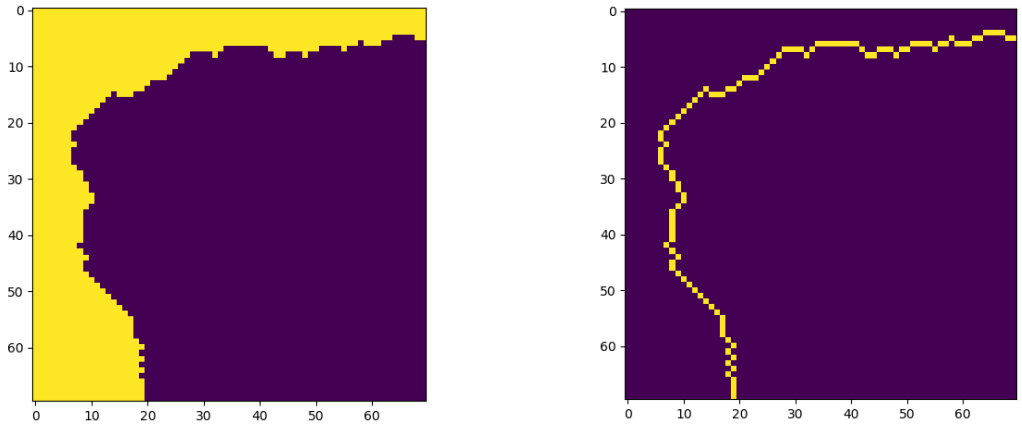
در حالتی که قسمت هم‌پوشانی افقی باشد، ابتدا هزینه‌ای که بین پیکسل‌های ستون اول قسمت هم‌پوشانی پنجره جدید با قسمت هم‌پوشانی عکس نتیجه است را به دست می‌آوریم سپس از ستون دو به بعد مشخص می‌کنیم که هزینه آن پیکسل چقدر است و اضافه می‌کنیم به کمترین هزینه‌ای که از ستون قبلی قابل دسترس هستند (همسایه این پیکسل هستند). در یک آرایه دیگر اندیس این پیکسل همسایه از ستون قبل را نگه می‌داریم. زمانی که به ستون آخر رسیدیم، آنگاه به صورت بازگشتی کمترین هزینه در ستون آخر را پیدا می‌کنیم. در آن اندیس در هر مرحله با توجه به آرایه اندیس‌ها به ستون قبل می‌رویم تا به ستون اول برسیم. بدین صورت یک مسیر پیدا کرده‌ایم که کمترین هزینه را برای جدا سازی قسمت بالا و پایین این هم‌پوشانی داشته‌است. در شکل ۵ اختلاف SSD و بهترین مسیر مشخص شده‌است.



شکل ۵ در سمت راست SSD تک تک پیکسل‌ها با عکس اصلی در قسمت هم‌پوشانی می‌باشد و سمت چپ مسیری است که الگوریتم برای بهترین برش داده است.

در L-شکل تنها تفاوتی که دارد این است که ابتدا حالت افقی و سپس حالت عمودی اجرا می‌شود سپس در نقطه (i,i) ای که جمع هزینه کمینه می شود در هر دو را پیدا می‌کنیم سپس از این نقطه به سمت راست برای حالت افقی و به سمت پایین برای حالت عمودی حرکت می‌کنیم.

در قسمت fill path هم قسمتی که باید از تصویر عکس نتیجه باقی بماند مشخص می‌شود و باقی تصویر از قسمت پنجره جدید انتخاب می‌شود. در شکل ۶ مشخص شده‌است.

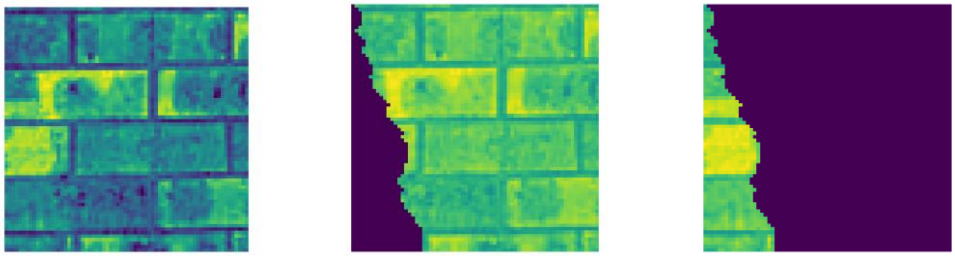


شکل ۶ عکس سمت چپ از قسمت عکس راست به‌دست می‌آید.

در شکل ۷ می‌توانیم مشاهده کنیم در تابع make_resault چگونه از مسیری که از قسمت‌های قبل به دست آمده بود شکل جدید را می‌سازیم در واقع عکس اول از راست از تصویر نتیجه برمی‌داشتیم و عکس وسط را از پنجره جدید از بافت برداشتیم و و در نهایت عکس اول از سمت چپ را در تصویر نتیجه قرار می‌دهیم.

من در همه قسمت‌ها از تصویر سیاه سفید برای SSD گرفتن استفاده می‌کنم ولی در عکس رنگی هم هم‌زمان تغییرات را اعمال می‌کنم و نتیجه به‌صورت عکس رنگی در شکل ۸ قابل مشاهده است.

در این سوال اندازه پنجره را ۷۰ در ۷۰ و قسمت هم‌پوشانی را ۲۰ پیکسل در نظر گرفته‌ام.



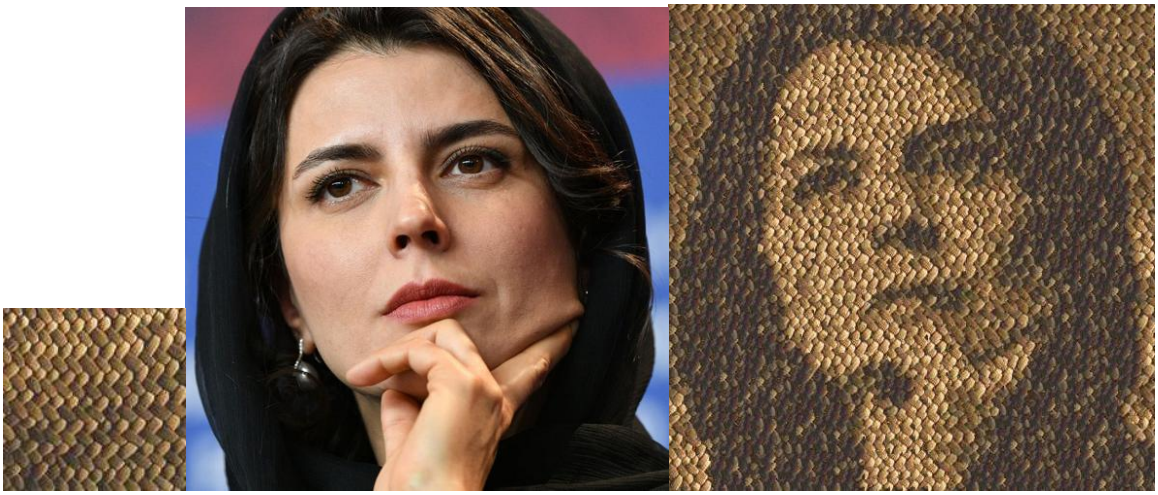
شکل ۷ از ترکیب دو عکس سمت راست، عکس سمت چپ به‌دست می‌آید.

شکل ۸ تصویر بدست آمده با استفاده از روش کم‌ترین برش

۴

در این سوال موقع محاسبه SSD این تغییر را در سوال قبل ایجاد کرده‌ام که SSD مربوط به تصویر target با ضریب ۰,۷ با SSD مربوط به قسمت هم‌پوشانی با ضریب ۰,۳ جمع می‌شوند و SSD جدید را می‌سازند. سایز پنجره تفاوت با سوالات قبل دارد بدین صورت که در این سوال پنجره کوچک‌تری در نظر گرفته شده‌است. اندازه پنجره ۲۰ در ۲۰ است که قسمت هم‌پوشانی آن ۷ پیکسل می‌باشد. هم‌چنین در انتخاب پنجره هم مانند سوالات قبل عمل شده‌است. بدین صورت که ۵۰۰ دفعه به صورت رندوم یک پنجره انتخاب می‌شود و آن پنجره‌ای که کوچک‌ترین SSD ای را دارد انتخاب می‌شود.

نتیجه در شکل ۹ قابل مشاهده است.



شکل ۹ از بافت سمت چپ و عکس هدفی که وسط قرار دارد، عکس سمت راست ساخته شده‌است.

۵

در این سوال از تابع `calcOpticalFlowFarneback` از پکیج `openCV` استفاده کرده‌ام. بدین صورت که در هرم هر لایه نصف لایه قبلی است و تعداد لایه‌های هرم را ۳ قرار داده‌ام. اندازه پنجره را ۲۰ در نظر گرفتم. تعداد ایتریشنی که در هر لایه از هرم انجام می‌شود را ۸ در نظر گرفتم. تعداد همسایه‌هایی که برای معادله هر پیکسل استفاده می‌شود را ۷ در نظر گرفتم که با تابع گاوس سیگمای ۱,۴ اسموث شده‌است. تابع در زیر آمده‌است.

```
flow = cv2.calcOpticalFlowFarneback(im1,im2, None, 0.5, 3, 20, 10, 7, 1.4, 0)
```

شکل ۱۰ نقاط پر رنگ بیشتر از نقاط دیگر جابجایی داشته‌اند هم‌چنین نقاط هم‌رنگ در یک جهت حرکت داشته‌اند.

برای warp کردن این flow بدست آمده، (u,v) هر پیکسل را با مختصات نقطه‌اش جمع می‌کنم. سپس با استفاده از تابع remap پکیج openCV استفاده می‌کنیم. در این تابع هر نقطه im11 را با استفاده از این تابع به مختصاتی که در flow آورده‌شده‌است منتقل می‌کنم. در واقع این قسمت را از فایل زیر برداشته‌ام (روش بهتری را نتوانستم پیاده‌سازی کنم که شبیه به کد زیر هم نباشد.):

https://github.com/npinto/opencv/blob/master/samples/python2/opt_flow.py

```
h, w = flow.shape[:2]
flow = -flow
flow[:, :, 0] += np.arange(w)
flow[:, :, 1] += np.arange(h)[:, np.newaxis]
res = cv2.remap(im11, flow, None, cv2.INTER_LINEAR)
cv2.imwrite('im5.jpg',res)
```

شکل ۱۱ شکل سوم در واقع تصویری است که شار نوری روی آن اعمال شده‌است. شار نوری از مقایسه بین تصویر اول و دوم حاصل شده‌است.