اصول پردازش تصویر

نیم‌سال دوم ۱۳۹۹-۱۴۰۰

مدرس: دکتر مصطفی کمالی تبریزی

دانشکده علوم ریاضی

تمرین سری دوم

نام و نام‌خانوادگی: امین کشیری

شماره دانشجویی: ۹۷۱۰۱۰۲۶

توضیحات کلی

- لینک تمامی فایل‌ها در گوگل درایو: اینجا
- عکس‌های و فیلم ورودی باید در پوشه‌ی inputs/images و inputs/videos قرار بگیرند.
- خروجی تمام کدها، با همان اسم گفته شده در صورت سوال، در پوشه‌ی outputs/images و فیلم‌ها در پوشه‌ی outputs/videos قرار خواهد گرفت (اگر عکس دیگری نیز در این پوشه باشد، در توضیحات سوال مربوطه نوشته‌ام).
- به جز دو پوشه‌ی بالا، یک پوشه‌ی دیگر نیز باید ساخته شود با نام outputs/temps و در آن نیز تعدادی پوشه برای بخش‌های مختلف سوال وجود دارند. این پوشه‌ها باید حتما از قبل وجود داشته باشند. برای همین کافی است از ساختار پوشه‌بندی این که آپلود کرده‌ام استفاده کنید و تنها ورودی‌ها را در جای خود قرار دهید (عکس‌ها در پوشه‌ی عکس و ویدئوها در پوشه ویدئو).
- بعضی از خط‌ها در کد، دارای comment هستند، که با uncomment کردن آن‌ها معمولاً می‌توانید عکس را در مراحل میانی ببینید (توضیحات بیشتر را در صورت لزوم، در توضیحات هر سوال داده‌ام).
- همراه فایل‌های یک فایل requirements.txt قرار دارد که محیط اجرای کدهای من است (در صورت نیاز).

سوال ۱

توضیحات جامع کد:

از آنجایی که این سوال بخش‌های زیادی دارد، کد هر بخش را در یک تابع جداگانه زده‌ام تا به خوانایی کد کمک کند. ساختار کلی کد از تعدادی توابع تشکیل شده است، که بعضی آن‌ها توابع کمکی هستند و ۸ تابع نیز توابع اصلی اجرای ۸ بخش مختلف برنامه هستند که با نام part i مشخص شده‌اند که i بین ۱ تا ۸ است.
روند کلی اجرای کد به این صورت است که در ابتدا، تعدادی ثابت مقداردهی می‌شوند:

- مقدار اول QUALITY است که آن را یا باید برابر با 1 قرار دهید یا 0.5 . در صورتی که برابر با ۱ باشد، محاسبات با عکس‌های کیفیت اصلی انجام خواهند شد و در غیر این صورت، با کیفیت کوچک شده با ضریب نیم (در هر دو جهت). بعضی از بخش‌ها که محاسبات کمی داشتند را تنها با کیفیت اصلی انجام داده‌ام (مانند بخش ۱). دقت کنید که جنس این متغییر string است. بسیاری از خروجی‌های قسمت‌های مختلف این سوال، به QUALITY وابسته هستند. اگر کیفیت را برابر با 0.5 بگذاریم، در انتهای خروجی مربوط رشته‌ی 0.5_ قرار خواهد گرفت.
- متغییر بعدی بیشترین عددی است که در int۶۴ در پایتون می‌توان ذخیره کرد تقسیم بر ۲. در جای مناسب این را توضیح می‌دهم.
- VIDEO_RESOLUTION اندازه‌ی هر فریم از ویدیو است (که البته دقت کنید به QUALITY وابسته است).
- CORNER_COORDS نیزیک ماتریس است که مختصات ۴ گوشه‌ی هر فریم را نگه می‌دارد.

- متغییر PART4_START_FROM تنها در بخش ۴ استفاده می‌شود، و به این خاطر است که اگر اجرای مرحله‌ی ۴ با مشکل مواجه شد، نیاز نباشد محاسبات را از ابتدا انجام دهیم (زیرا محاسبات قسمت ۴ بسیار وقت گیر است). به این صورت استفاده می‌شود که در صورتی که می‌خواهیم از ابتدا محاسبه کنیم آن را برابر با ۰ می‌گذاریم. در صورتی که از قبل اجرا کرده باشیم، و آخرین تصویر میانی تا عرض x پیش رفته باشد، می‌توانیم این متغییر را برابر با x بگذاریم و محاسبات خود به خود از آنجا ادامه پیدا خواهد کرد (در مورد عرض x در قسمت ۴ بیشتر توضیح خواهم داد). تنها باید به این نکته دقت کنید که x را تنها اعدادی می‌توانید قرار دهید که متناظر با آن‌ها حتما یک عکس میانی (عکس‌هایی که به عنوان temp نگه می‌دارم) از قبل ذخیره شده باشد وگرنه باید از ابتدا محاسبه کنید.
- PATCH_WIDTH نیز عرض نواری که در مرحله‌ی ۴ استفاده می‌کنم است که این را نیز توضیح خواهم داد.
- در متغییر ref_homographies هموگرافی‌های مرجع را نگه می‌دارم.
- تعریف NUMBER_OF_FRAMES هم که مشخص است (با کم کردن این عدد، مثلا برای سوال ۸ می‌توان محاسبات را برای تعداد فریم‌های کمتری انجام داد).
- متغییر MOVING_AVG_WIN را نیز در بخش ۸ توضیح خواهم داد.

پس از تعریف این متغییرها، تعدادی تابع وجود دارد که می‌توانید آن‌ها را کامنت یا آنکامنت کنید.اولین تابع، تابع extract_frames است که فریم‌های تصویر را به دست می‌آورد و آن‌ها را در پوشه‌ی inputs/videos/frames/ قرار می‌دهد (با استفاده از ffmpeg).

تابع دوم compute_homographies است. این تابع برای کار کردن به هموگرافی فریم‌های مرجع احتیاج دارد. برای همین در ابتدای کار هموگرافی فریم‌های مرجع را محاسبه می‌کند. سپس برای هر فریم، یک هموگرافی از آن به فریم به فریم مرجع مربوط به خود پیدا می‌کنیم، سپس با ضرب هموگرافی مرجع در این هموگرافی از سمت چپ، هموگرافی نهایی آن فریم به فریم۴۵۰ را به دست می‌آوریم. تمام این هموگرافی ها را در یک ماتریس ذخیره می‌کنیم و آنها را با نام homographies.tiff (از فرمت tiff استفاده کردم تا بتوانم مقادیر غیر صحیح را ذخیره کنم) در پوشه‌ی temp ذخیره می‌کنیم (و باز هم دقت کنید اگر کیفیت برابر نیم بود در انتهای این خروجی 0.5_ قرار خواهد گرفت). از الان به بعد، در تمام کد می‌توانیم با خواندن این فایل، تمام هموگرافی‌ها را داشته باشیم و نیازی به دوباره محاسبه کردن آن‌ها نیست و بنابراین می‌توانیم صدا کردن این تابع را کامنت کنیم. در مراحل محاسبه کردن هموگرافی‌ها، از چند تابع کوچک تر استفاده می‌کنیم. این توابع عبارتند از:

- compute_ref_homographies
- get_homography
- get_matches
- get_matched_keypoints_and_coords

توابع دوم تا چهارم، دقیقا توابعی هستند که در تمرین سری ۱ هستند بنابرین آن‌ها را دوباره توضیح نمی‌دهم. به صورت خلاصه تابع get_homography با گرفتن دو عکس ورودی، یک هموگرافی از تصویر یک به دو پیدا می‌کند و در این راه از دو تابع کمکی دیگر نیز استفاده می‌کند. تابع compute_ref_homographies نیز تنها کاری که انجام می‌دهد این است که این محاسبه هموگرافی‌ها را برای فریم‌های مرجع انجام دهد و نتیجه را ذخیره کند.

در خط بعدی تابع compute_background_resolution_and_translate_matrix را صدا می‌کنم. این تابع با استفاده از تمام هموگرافی‌ها، بیشترین و کمترین x و y را برای تمام فریم‌ها محاسبه می‌کند. با این کار حداقل اندازه‌ی تصویر پس‌زمینه‌ای را پیدا می‌کنیم که تمام عکس‌ها پس از warp شدن در آن قرار می‌گیرند. چپ‌ترین و بالاترین مختصات عکس‌های warp شده را نیز محاسبه می‌کنیم و به کمک آن یک ماتریس انتقال به دست می‌آوریم که به ما کمک می‌کند که طوری تمام عکس‌های وارپ شده را انتقال دهیم که تمام آن‌ها داخل کادر قرار بگیرند. سپس این ماتریس انتقال و اندازه‌ی تصویر پس‌زمینه را خروجی می‌دهیم.

پس از تمامی این مراحل، می‌توانیم کارهای خواسته شده در صورت سوال را انجام دهیم. دقت کنید که دو تابع compute_homographies و extract_frames تنها نیاز دارند که یک بار اجرا شوند (البته برای هر کیفیت یک بار). بعد از آن می‌توانید این دو را کامنت کنید.

بخش ۱

در این بخش، ابتدا دو فریم خواسته شده را می‌خوانیم. سپس هموگرافی از فریم ۲۷۰ به ۴۵۰ را با تابع توضیح داده شده به دست می‌آوریم. یک مستطیل روی فریم ۴۵۰ می‌کشیم، سپس با استفاده از وارون هموگرافی، مختصات این مستطیل را به فریم ۲۷۰ تبدیل می‌کنیم و در آن فریم نیز این چهارضلعی را می‌کشیم و دو عکس را با نام خواسته شده ذخیره می‌کنیم. سپس با استفاده از هموگرافی، فریم ۲۷۰ را به صفحه‌ی ۴۵۰ و با اندازه‌ی پس زمینه‌ای که در مرحله قبلی به دست آورده بودیم می‌بریم. حال در پیکسل‌هایی که قرار بوده فریم ۴۵۰ قرار بگیرد را در جای خود قرار می‌دهیم و نتیجه را ذخیره می‌کنیم.

نکته: تبدیل مختصات نقاط با استفاده از یک هموگرافی با استفاده از تابع perspectiveTransform صورت می‌گیرد، که از تابعی با همین اسم در opencv استفاده می‌کند، تنها برای استفاده از کد خودم تابعی ساده‌تر نوشتم.

بخش ۲

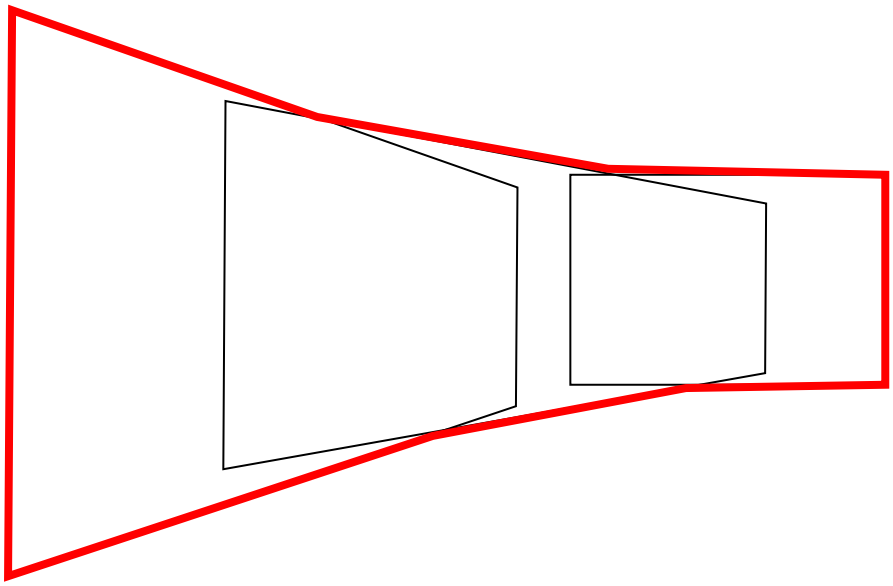
در این سوال، از ۳ تابع کمکی دیگر استفاده می‌کنم. روند کار به این صورت است که تصویر اصلی را ابتدا فریم ۴۵۰ در نظر می‌گیرم. سپس فریم‌های دیگر را یکی یکی به این تصویر اصلی اضافه می‌کنیم و به صورت مناسبی ادغام می‌کنیم. در نتیجه یک کار یکسان ۴ بار تکرار شده است. به همین دلیل من فقط برای مثال اولین تکرار و فریم ۲۷۰ را در نظر می‌گیرم. برای بقیه فریم‌ها نیز دقیقا همین کار را انجام می‌دهیم.

در ابتدای کد تصاویر مورد نیاز را لود می‌کنیم و هموگرافی‌های مرجع را محاسبه می‌کنیم (می‌شود از هموگرافی‌های ذخیره شده استفاده کرد ولی برای ۴ هموگرافی دوباره محاسبه کردن هزینه‌ی زیادی ندارد). تصویر اصلی را به فریم پس‌زمینه منتقل می‌کنیم.

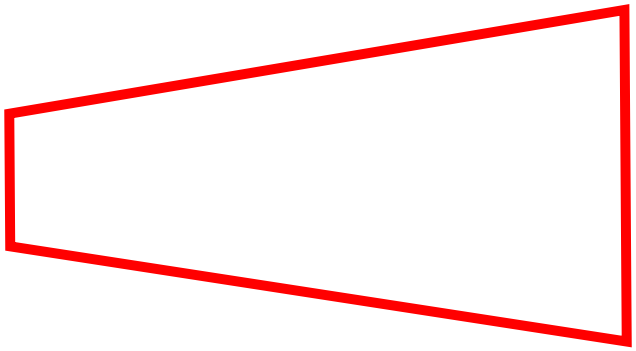
- get_mask این تابع به عنوان ورودی یک هموگرافی می‌گیرد. سپس گوشه‌های یک فریم را با این هموگرافی به تصویر پس‌زمینه منتقل می‌کنیم، سپس یک چهارضلعی با این ۴ نقطه می‌کشیم و یک ماسک درست می‌کنیم. دقت کنید هموگرافی‌ای که در این تابع استفاده می‌شود، در دل خود انتقال را نیز باید داشته باشد. به عبارت دیگر، با استفاده از هموگرافی یک فریم، ماسکی برای مکان آن فریم در پس‌زمینه پیدا می‌کنیم.

در هر مرحله، فریم بعدی را وارپ می‌کنیم و به فریم پس‌زمینه می‌بریم، سپس آن را با فریم اصلی (فریم پس‌زمینه که تا الان کامل شده است) ادغام می‌کنیم. برای ادغام کردن از seams ها کمک گرفتم (البته تغییراتی نیز داده‌ام).

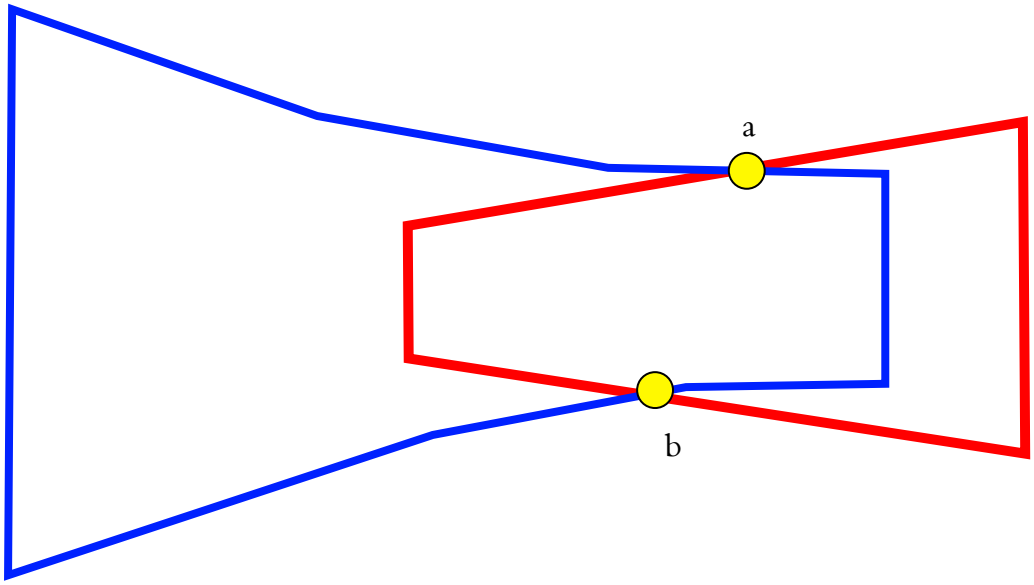
در هر مرحله، ماسک پس‌زمینه را نگه می‌داریم. منظور از ماسک پس‌زمینه ماسکی است که در تمام نقاطی که حداقل یک تصویر وجود دارد ۱ باشد وگرنه صفر. یعنی در مرحله‌ی اول برابر است با ماسک فریم ۴۵۰، بعد از اضافه کردن فریم ۲۷۰، برابر می‌شود با or ماسک فریم ۴۵۰ و ۲۵۰ و ... . سپس در هر مرحله با استفاده از ماسک پس‌زمینه، مرزهای تصویر تا الان ساخته شده را پیدا می‌کنیم. برای مثال در مرحله‌ی دوم، اگر تصویر فریم ۴۵۰ و ۲۷۰ را با هم ترکیب کنیم، یک تصویر حاصل می‌شود که ترکیبی از فریم ۴۵۰ و ۲۷۰ است، اما دور آن پیکسل‌های سیاه وجود دارند. مکانی که پیکسل‌های سیاه (پس‌زمینه) به مرز فریم ۲۷۰ یا ۴۵۰ می‌رسند مرز ما می‌شود. دقت کنید که در مرحله‌ی اول، مرز تصویر یک مستطیل است. اما در مرحله‌ی دوم دیگر یک ۶ ضلعی (غیرمحدب) است. و در مراحل بعدی پیچیده‌تر هم می‌شود. اما در صورتی که ماسک پس‌زمینه را داشته باشیم (که در این جا داریم و روش به دست آوردن آن را نیز توضیح دادیم، که در هر مرحله باید ماسک فریم جدید را با ماسک فعلی or کنیم)، می‌توانیم با استفاده از morphology مرزهای این ماسک را پیدا کنیم (یک dilation و سپس کم کردن تصویر اصلی). حال در مرحله، اشتراک مرز تصویر فعلی و ماسک فریم جدید را محاسبه می‌کنیم، و این به ما نقاطی را می‌دهد که مرزها همدیگر را قطع می‌کنند. برای واضح تر شدن یک مثال می‌زنیم. در مرحله‌ی ۴م، سه فریم ۹۰ و ۲۷۰ و ۴۵۰ ادغام شده‌اند. ماسک تصویر اصلی شکلی شبیه به زیر خواهد داشت:



و ماسک فریم ۶۳۰ نیز شکلی مثل زیر خواهد داشت:



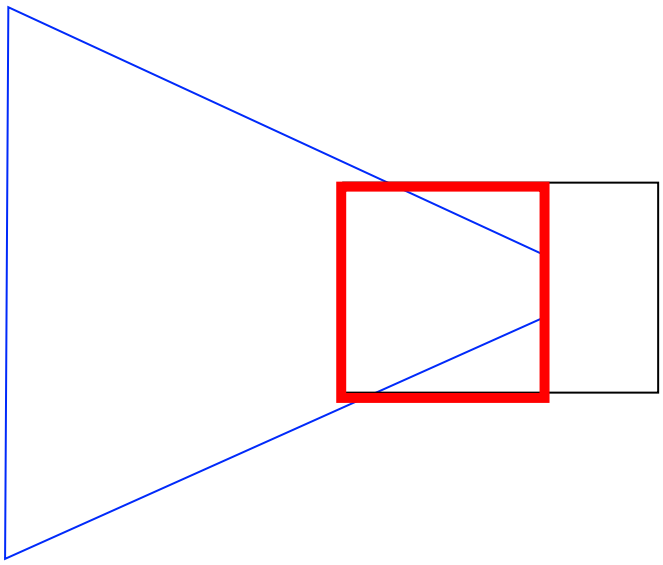
حال، الگوریتم من به دنبال یافتن نقاط a و b است:



اما چگونه این دو نقطه را محاسبه می‌کنم؟ همانطور که اشاره کردم، با نگه داشتن ماسک‌ها و or کردن آن‌ها و سپس با استفاده از مورفولوژی مرزهای قرمز و آبی در شکل بالا را به دست می‌آورم. سپس با and کردن ماسک این دو مرز، مجموعه نقاطی نزدیک به a و b تشکیل می‌شوند. اما مشکل اینجاست که لزوما دو نقطه نخواهیم داشت (به دلیل این که مرزها دقیقا یک خط با ضخامت ۱ و صاف نیستند). برای این که این مشکل حل شود، با استفاده از یک عملیات دیگر مورفولوژی به نام opening استفاده می‌کنیم. همانطور که می‌دانید opening به این صورت است که ابتدا با یک dilation نواحی اشتراک را گسترش می‌دهیم. این کار، باعث میشود که نقاط تکی ولی نزدیک به هم به یکدیگر بچسبند و تشکیل یک مولفه‌ی همبندی بدهند. در مرحله‌ی بعد با یک erosion دوباره این مولفه‌ها را کمی کوچک می‌کنیم. این کار باعث می‌شود در شکل ما تنها

دو مولفه‌ی همبندی نزدیک a و b به وجود بیاید. حال، با استفاده از الگوریتم non maximum supression که در تمرین سری ۱ پیاده‌سازی کرده بودم، یک نقطه از هر کدام از این دو مولفه به دست می‌آورم. اما برای این که این نقطه بیشتر به مرکز مولفه‌ها نزدیک باشد، ابتدا روی عکس binary یک فیلتر گوس اعمال می‌کنم. این کار باعث می‌شود پیکسل‌هایی که به مرکز نزدیک ترند، مقدار بزرگتری داشته باشند. و این باعث می‌شود الگوریتم nms ما نقاط مرکزی تر را خروجی دهد. بالاخره با استفاده از تمام کاری‌های بالا، نقطه‌ی a و b را محاسبه می‌کنیم.

در مرحله‌ی بعدی برای ادغام تصویر اصلی و فریم بعدی، دو تصویر را روی هم قرار می‌دهیم و مستطیلی که اشتراک دو تصویر (اشتراک ماسک تصویر اصلی و فریم جدید) را شامل می‌شود در نظر می‌گیریم. برای مثال در شکل زیر مستطیل قرمز:



حال در این مستطیل، دو تصویر را از هم کم می‌کنم، به توان دو می‌رسانم، و مقادیر را در راستای ۳ کانال جمع می‌زنم. سپس کوتاه‌ترین مسیر بین a و b را که در این مستطیل قرار می‌گیرد محاسبه می‌کنم.

الگوریتمی شبیه به کار بالا در یکی از تمرین‌های پردازش تصویر داشتیم بنابرین من آن الگوریتم (با استفاده از DP) را دوباره توضیح نمی‌دهم، و تنها تفاوت‌های آن را ذکر می‌کنم. در این مسئله بر خلافه آنجا، نقطه‌ی شروع و پایان ثابت است. برای این که همان کد سوال پردازش تصویر من در اینجا کار کند، مقادیر سطر اول را به جز نقطه‌ی شروع یک مقدار خیلی زیاد گذاشتم (و INT64_MAX_VALUE همینجا کاربرد دارد). همچنین بعد از انجام DP ، بهترین مسیری که به b می‌رسد را پیدا کردم (نه تنها به پایین مستطیل).

یک مشکل دیگر نیز وجود دارد، و آن این است که مسیر ما تنها حق دارد از نقاط اشتراک بگذرد! و حق ندارد از نواحی‌ای از این مستطیل بگذرد که دو تصویر اشتراک ندارند. به این منظور نیز در آن مستطیلی که اختلاف‌ها را نگه داشتم، مقادیر خانه‌هایی که بیرون ماسک قرار می‌گرفتند را بسیار زیاد قرار دادم (که کوتاه ترین مسیر مجبور شود از این خانه‌ها نگذرد). همچنین، در سوال پردازش تصویر این محدودیت را داشتیم که هر باری که مسیر یک خانه پایین می‌رود، تنها حق دارد یک خانه به راست یا چپ هم برود. اما در این سوال، با این شرط ممکن است لزوما یک مسیر از a به b پیدا نشود. به همین دلیل این محدودیت را ۲ پیکسل گذاشتم (و به صورت تجربی جواب بسیار مناسب بود).

پس از پیدا کردن کوتاه‌ترین مسیر نیز، مقادیر سمت راست را از تصویر راست می‌آورم و برعکس. ممکن است سوال باشد که چرا اصلا این کارها را کردم؟

نکته‌ای که به نظر من رسید این بود که از هر نقطه‌ای ما بخواهیم یک تقسیم انجام بدهیم، مقداری از مرز بین دو تصویر قابل تشخیص می‌شد. اما دو نقطه‌ی a و b این خاصیت را دارند که اگر از آنجا عکس‌ها بریده شوند، به جز خود کوتاه ترین مسیر، هیچ مرزی باقی نمی‌ماند. راه‌های دیگری نیز وجود داشتند. مثلا می‌شد بین a و b یک خط صاف کشید و سپس با استفاده از 2 band blending از این مرز تصاویر را ادغام کنیم (به جز روش‌های ساده‌تر 2 band blending). اما هیچ کدام از نتایج به خوبی این روشی که توضیح دادم نشدند. احتمالا اگر 2 band blending را نیز روی کوتاه ترین مسیری (seams) که پیدا کردم می‌زدم، نتیجه از این نیز بهتر می‌شد، اما دیگر این کار را انجام ندادم.

تابع find_min_cut_mask نیز دقیقا همین الگوریتم DP است که توضیح دادم. و چون از تمرین‌های پردازش تصویر است بیشتر توضیح نمی‌دهم.

بخش ۳

در این بخش دقیقا مانند توضیحات سوال عمل کردم. ابتدا هموگرافی‌هایی که در مراحل قبلی محاسبه کرده بودم را لود می‌کنم. سپس به ازای هر فریم، هموگرافی متناظر را از راست در ماتریس انتقال ضرب می‌کنم تا ماتریس نهایی وارپ به صفحه‌ی پس‌زمینه را به دست آورم. دقت کنید در صورت سوال گفته‌است کوچک‌ترین اندازه‌ی مناسب برای تصویر پس‌زمینه را به دست آورید. من این مقدار را اول کد محاسبه کرده‌ام، و قبل از توضیحات بخش ۱ نحوه‌ی محاسبه را نیز توضیح دادم. سپس با استفاده از ماتریس هموگرافی نهایی تصویر را وارپ می‌کنیم و نتیجه را ذخیره می‌کنیم. پس از این که این کار را برای هر ۹۰۰ فریم انجام دادیم، با استفاده از ffmpeg فریم‌ها را به یک فیلم با نام خواسته شده تبدیل می‌کنیم.

بخش ۴

این بخش ایده این است که برای هر پیکسل، مقدار آن در تمام پیکسل‌ها را بگیریم و از آن‌ها میانه بگیریم تا مقدار آن در پس‌زمینه را به دست آوریم. مد، لزوما جواب خوبی به ما نمی‌دهد (چون مقایر یک پیکسل در فریم‌های مختلف به هم نزدیک‌اند، اما لزوما یکی نیستند). اما میانه این مشکل را ندارد. هرچند دقت کنید در زمان‌هایی که پیکسل اصلا جزو تصویرها نیست و صفر است آن را در نظر نگیریم. اما مشکلی که وجود دارد این است که لود کردن تمام عکس‌ها و درست کردن یک لیست برای هر پیکسل و نگه داشتن همه در موری خیلی هزینه بر و غیر ممکن است. بنابرین ما ساخت پاناروما‌ی پس‌زمینه را نوار به نوار انجام می‌دهیم. مقدار PATCH_WIDTH اندازه‌ی این نوار را مشخص می‌کند. برای حالت با کیفیت اصلی، مقداری که RAM من برای آن کافی بود حدود ۲۰ پیکسل بود (و البته لپتاپ از کار نیفتد!). اما کد به صورتی طراحی شده که برای هر عددی کار کند. بنابرین اگر موقع اجرا سیستم قوی‌تری دارید می‌توانید این عدد را تا هرجا می‌خواهید زیاد کنید.

قبل از توضیح روش کار، یک متغییر PART4_START_FROM نیز وجود دارد که درمورد آن در ابتدای توضیحات این سوال توضیح دادم. به صورت خلاصه‌ی شماره‌ی آخرین تصویری که کد من ذخیره کرده‌است (در پوشه‌ی outputs/temps/q1_4_frames) را می‌توانید در این متغییر بگذاریم و کد من محاسبات را از آنجا به بعد انجام می‌دهد تا دوباره از اول محاسبات را انجام ندهیم (اگر به هر دلیلی کد متوقف شد. چون این سوال نسبت به بقیه سوال‌ها خیلی کند است).

روند اجرای کد به این صورت است که یک آرایه colors در نظر می‌گیرم. به ازای هر فریم، اگر یک پیکسل در نقاط داخل تصویر آن فریم باشد، مقدار رنگ آن را در این آرایه ذخیره می‌کنم. زمان‌هایی که یک پیکسل جزو پس‌زمینه است اما، در colors برای آن ۰ نمی‌گذارم، بلکه به صورت یکی در میان $256-$ و 256 می‌گذارم. پس از این که این کارها را انجام دادیم، در نهایت به ازای هر پیکسل، یک لیست دارم که در هر خانه‌ی آن یا یک عدد بین ۰ تا ۲۵۵ است، یا 256 و یا $256-$. چون به ازای پیکسل‌های پس‌زمینه یکی در میان این دو عدد را اضافه کردیم و این اعداد در بین اعداد خود ما وجود ندارند و در دو سر طیف قرار دارند، وجود آن‌ها در میانه «هیچ» تاثیری ندارد. البته یک نکته‌ی مهم وجود دارد و آن این است که اگر تعداد ۲۵۶ها یکی بیشتر از منفی ۲۵۶ها بود چه؟ برای این کار نیز در صورتی که برای یک پیکسل، این اتفاق رخ داد، به لیست آن منفی ۲۵۶ اضافه می‌کنم و سپس میانه می‌گیرم (برای همین نیز اگر دقت کنید دو بار میانه گرفته‌ام). در نهایت، این استراتژی میانه را تغییر نمی‌دهد و می‌توانم کاملا با استفاده از توابع numpy به هدف خود برسم. آرایه alternate_mask یک ماتریس است که همین پیچیدگی‌های محاسباتی ذکر شده را هندل می‌کند. در صورتی که نبود باید روی پیکسل‌ها for بزنیم و ... ، کد ما خیلی کند می‌شد. اما به کمک این ماتریس یکی در میان به لیست پیکسل‌ها ۲۵۶ و منفی آن اضافه می‌کنیم (اگر در پس‌زمینه سیاه بودند). پس به صورت نواری تصویر پس‌زمینه را می‌سازیم، و در آخرین مرحله عکس‌نهایی را ذخیره می‌کنم.

بخش ۵

در این بخش نیز دقیقا مانند توضیحات عمل می‌کنیم. دقت کنید که این بخش به خروجی بخش ۴ احتیاج دارد. ابتدا هموگرافی‌ها و عکس پس‌زمینه را لود می‌کنیم. سپس به ازای هر فریم هموگرافی وارون آن را محاسبه می‌کنیم و روی تصویر پس‌زمینه اعمال می‌کنیم و نتیجه را در پوشه‌ی outputs/temps/q1_5_frames ذخیره می‌کنیم. در نهایت با کمک تمام این تصویرها ویدیو پس‌زمینه را به دست می‌آوریم (و با کمک ffmpeg).

بخش ۶

در این بخش ابتدا میزان threshold را که به صورت تجربی محاسبه کرده‌ام مقداردهی می‌کنم. سپس برای هر فریم، خود فریم و تصویر پس‌زمینه را از هم کم می‌کنم، و بزرگی اختلاف را در هر پیکسل به دست می‌آورم. سپس تمام پیکسل‌هایی که مقدار آن‌ها از threshold بیشتر بود را پیش‌زمینه تشخیص می‌دهم و از آن یک ماسک درست می‌کنم. برای حذف کردن نویزها، ابتدا از یک opening استفاده می‌کنیم تا نویزهای کوچک حذف شوند. سپس چون ماسک پیش‌زمینه کمی سوراخ سوراخ است (داخل ماشین‌ها) و الان بیشتر نویزها حذف شده‌اند، با استفاده از یک closing این حفره‌ها را پر می‌کنم. سپس در مکان‌های پیش‌زینه مقدار قرمز عکس را ۱۰۰ واحد زیاد و دو رنگ دیگر را کم می‌کنم. پس از آن نیز مقادیر بیشتر از ۲۵۵ و کمتر از ۰ را نیز به بازه ۰ تا ۲۵۵ ، clip می‌کنم و نتیجه را ذخیره می‌کنم. در نهایت نیز به کمک تمام این فریم‌ها، فیلمی از پس‌زمینه می‌سازم و با نام مورد نظر سوال ذخیره می‌کنم.

بخش ۷

کد این بخش، دقیقاً همان کد بخش ۵ است، با این تفاوت که پس از برگرداندن فریم به مکان خود با کمک وارون هموگرافی، آن را در تصویری با عرض بزرگتر قرار می‌دهم (عرض تصویر را از سمت راست افزایش می‌دهیم). مشکلی که اینجا رخ می‌دهد این است که برای فریم‌های اولیه این کار خیلی خوب است. اما مثلا برای فریم ۹۰۰، دیگر چیزی سمت راست آن قرار ندارد (که بتوانیم آن را عرض کنیم). بنابرین ساخت ویدیو را تنها تا فریمی ادامه می‌دهم که هنوز تصویر کاملی به ما می‌دهد. این نتیجه نیز منطقی است. اگر شما با سرعت یکسان، از تصویر یکسان بخواهید یک ویدئو پاناروما درست کنید، اگر عرض تصویر شما زیاد تر شود، برای طی کردن همان زاویه دید قبلی، به زمان کمتری احتیاج دارید. بنابرین منطقی است که زمان چنین ویدئویی در صورت افزایش زاویه دید آن کاهش یابد.

بخش ۸

ابتدا ایده‌ای کلی راه حل را توضیح خواهم داد و سپس جزییات پیاده‌سازی کد را توضیح می‌دهم. ما از هر فریم یک هموگرافی به فریم ۴۵۰ داریم. اگر تنها دوران داشته باشیم (که این جا این فرض را می‌کنیم)، می‌توانیم ثابت کنیم که بین هر دو فریم (و به خصوص بین هر فریم و فریم ۴۵۰)، یک هموگرافی داریم که خاصیت زیر را دارد:

$$H_i = K R_i K^{-1}$$

که R_i یک ماتریس دوران است. اگر دستگاه مختصات را به این صورت در نظر بگیریم که فریم ۴۵۰، صفحه‌ی XY ما باشد، به صورتی که محور x در راستای افقی و به سمت راست، و محور y به صورت عمودی و به سمت پایین باشد، آنگاه محور z باید به سمت درون فریم باشد (تا دستگاه راستگرد باشد). این دستگاه مختصات را دستگاه مرجع می‌گیریم. حال برای هر کدام از فریم‌ها اگر بتوانیم R_i را پیدا کنیم، می‌توانیم آن را در سه راستا تجزیه کنیم و سه زاویه دوران x و y و z را به دست می‌آوریم (زوایای دوران را x و y و z گرفتم که مثلا x دوران حول محور x است! در ادامه این کار به شهود ما کمک می‌کند). حال اگر ما تمام این زوایا را برای ۹۰۰ تا فریم پشت هم نشان دهیم، می‌بینیم که خیلی نوسان دارد. و از هر فریم به فریم بعدی، مثلا ناگهان x خیلی تغییر کرده است. حال اگر ما بتوانیم کاری کنیم که این تغییرات هموار باشد، آنگاه تغییرات ما از هر فریم به فریم بعدی اینقدر ناگهانی نخواهد شد. حال، برای فریم i م، اگر بتوانیم این زوایای هموار شده را به دست آوریم، آنگاه می‌توانیم فریم i را به صفحه فریم ۴۵۰ ببریم، سپس آن را با ماتریس دوران جدیدی که به دست آوردیم به صفحه‌ی خود برگردانیم. آنگاه اتفاقی که می‌افتد این است که هر فریم مقداری جابجا شده است. اما، تغییرات هر دو فریم متوالی بسیار هموار شده است. در واقع ما باید به کمک مراحلی که گفتیم، x و y و z را پیدا کنیم، به کمک آن یک هموگرافی وارون از فریم ۴۵۰ پیدا کنیم، و با آن فریم i را دوباره بسازیم. به صورت خلاصه اگر این ماتریس هموگرافی که آن را H' می‌نامیم را پیدا کنیم، می‌توانیم تبدیل زیر را انجام دهیم:

$$im'_i = (H')^{-1} H (im_i)$$

که منظور این است که فریم i م را با ماتریس $(H')^{-1}H$ وارپ کنیم. و طبق توضیحاتی که دادم، فریم‌های جدید این خاصیت را دارند که دیگر به صورت هموار تغییر می‌کنند و نویزها حذف می‌شوند. حال جزییات بیشتر روش:

- اولین مشکلی که داریم، این است که ما اصلا ماتریس K را نداریم که بتوانیم R را محاسبه کنیم و سپس تجزیه کنیم. برای به دست آوردن K اگر فرض کنیم Principal Point در وسط تصویر است و فاصله‌ی کانونی در دو راستا برابر است و skewness نداریم، آنگاه کافی است f را به دست بیاوریم. پس مرحله‌ی اول به دست آوردن f است.

به این منظور، به ازای هر کدام از هموگرافی‌ها می‌توانیم تعدادی معادله برحسب f و زوایای دوارن به دست آوریم. به صورت دقیق تر، برای هر هموگرافی می‌توانیم بنویسم:

$$H_i = KR_iK^{-1}$$

$$H = \begin{pmatrix} f & 0 & \frac{w}{2} \\ 0 & f & \frac{h}{2} \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} \cos(z) & -\sin(z) & 0 \\ \sin(z) & \cos(z) & 0 \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} \cos(y) & 0 & \sin(y) \\ 0 & 1 & 0 \\ -\sin(y) & 0 & \cos(y) \end{pmatrix} \cdot \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos(x) & -\sin(x) \\ 0 & \sin(x) & \cos(x) \end{pmatrix} \cdot \begin{pmatrix} f & 0 & \frac{w}{2} \\ 0 & f & \frac{h}{2} \\ 0 & 0 & 1 \end{pmatrix}^{-1}$$

$$H = \begin{pmatrix} \frac{f*\cos(y+z)+f*\cos(y-z)-w*\sin(y)}{2*f} & \frac{-f*\cos(x+y+z)+f*\cos(x-y+z)-f*\cos(x+y-z)+f*\cos(x-y-z)+w*\sin(x+y)+w*\sin(x-y)-2*f*\sin(x+z)+2*f*\sin(x-z)}{4*f} & \frac{2*} \\ \frac{-h*\sin(y)+f*\sin(y+z)-f*\sin(y-z)}{2*f} & \frac{2*f*\cos(x+z)+2*f*\cos(x-z)+h*\sin(x+y)+h*\sin(x-y)-f*\sin(x+y+z)+f*\sin(x-y+z)+f*\sin(x+y-z)-f*\sin(x-y-z)}{4*f} & \frac{2} \\ \frac{-\sin(y)}{f} & \frac{\sin(x+y)+\sin(x-y)}{2*f} & \end{pmatrix}$$

و همانطور که می‌بینید این روابط آنقدر پیچیده‌اند که حتی در صفحه جا نشده است :) . ما با حل کردن این تساوی، می‌توانیم x و y و z و f را به دست آوریم. اما چون این معادله بسیار پیچیده است، ما با توجه به هموگرافی‌هایی که داشتیم، فرض کردیم که $x = 0$ و یعنی دوران را فقط در دو راستا نظر گرفتیم. معادلات ما به صورت زیر کاهش می‌یابد:

$$H_i = KR_iK^{-1}$$

$$H = \begin{pmatrix} f & 0 & \frac{w}{2} \\ 0 & f & \frac{h}{2} \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} \cos(z) & -\sin(z) & 0 \\ \sin(z) & \cos(z) & 0 \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} \cos(y) & 0 & \sin(y) \\ 0 & 1 & 0 \\ -\sin(y) & 0 & \cos(y) \end{pmatrix} \cdot \begin{pmatrix} f & 0 & \frac{w}{2} \\ 0 & f & \frac{h}{2} \\ 0 & 0 & 1 \end{pmatrix}^{-1}$$

$$H = \begin{pmatrix} \frac{f*\cos(y+z)+f*\cos(y-z)-w*\sin(y)}{2*f} & -\sin(z) & \frac{2*f*w*\cos(y)-f*w*\cos(y+z)-f*w*\cos(y-z)+w^2*\sin(y)+2*f*h*\sin(z)+2*f^2*\sin(y+z)+2*f^2*\sin(y-z)}{4*f} \\ \frac{-h*\sin(y)+f*\sin(y+z)-f*\sin(y-z)}{2*f} & \cos(z) & \frac{2*f*h*\cos(y)-2*f*h*\cos(z)-2*f^2*\cos(y+z)+2*f^2*\cos(y-z)+h*w*\sin(y)-f*w*\sin(y+z)+f*w*\sin(y-z)}{4*f} \\ \frac{-\sin(y)}{f} & 0 & \frac{2*f*\cos(y)+w*\sin(y)}{2*f} \end{pmatrix}$$

اما یک مشکل دیگر وجود دارد و آن این است که ماتریس هموگرافی ۸ درجه‌ی آزادی دارد. بنابرین هر ضریبی از H می‌تواند در این تساوی قرار بگیرد. برای این که این مشکل حل شود، تمام درایه‌های هر دو ماتریس را بر درایه‌ی $(3,3)$ تقسیم می‌کنیم و سپس رابطه را برقرار می‌کنیم:

$$H = \begin{pmatrix} \frac{f*\cos(y+z)+f*\cos(y-z)-w*\sin(y)}{2*f*\cos(y)+w*\sin(y)} & \frac{-2*f*\sin(z)}{2*f*\cos(y)+w*\sin(y)} & \frac{2*f*w*\cos(y)-f*w*\cos(y+z)-f*w*\cos(y-z)+w^2*\sin(y)+2*f*h*\sin(z)+2*f^2*\sin(y+z)+2*f^2*\sin(y-} \\ \frac{-h*\sin(y)+f*\sin(y+z)-f*\sin(y-z)}{2*f*\cos(y)+w*\sin(y)} & \frac{2*f*\cos(z)}{2*f*\cos(y)+w*\sin(y)} & \frac{2*f*h*\cos(y)-2*f*h*\cos(z)-2*f^2*\cos(y+z)+2*f^2*\cos(y-z)+h*w*\sin(y)-f*w*\sin(y+z)+f*w*\sin(y-} \\ \frac{-2*\sin(y)}{2*f*\cos(y)+w*\sin(y)} & 0 & \frac{4*f*\cos(y)+2*w*\sin(y)}{1} \end{pmatrix}$$

حال، با حل این معادلات می‌توانیم متغییرهای خواسته شده را به دست آوریم. با استفاده از روابط بالا، مثلا با تقسیم درایه $(1,2)$ بر $(2,2)$ ، می‌توانیم tan(z) و در نتیجه z را پیدا کنیم و ... در نهایت با حل این معادلات به رابطه‌های زیر می‌رسیم:

$$z = -tg^{-1}(\frac{H_{12}}{H_{22}})$$

$$sin(y) = a \cdot f \rightarrow a = \frac{H_{31}}{H_{12}} sin(z)$$

$$f = \frac{\sqrt{1-(\frac{2cos(z)-awH_{33}}{2H_{33}})^2}}{a}$$

و این یعنی با استفاده از هر کدام از هموگرافی‌ها احتمالا می‌توانیم یک f به دست بیاوریم. دقت کنید این f به صورت یکتا محاسبه می‌شود (زیرا همواره مثبت است). اما در اینجا به دلیل مشکلات محاسباتی و همچنین حذف کردن متغییر x لزوما به جواب نمی‌رسیم (مثلا ممکن است زیر رادیکال منفی شود). اما در اکثر موارد می‌توانیم یک f محاسبه کنیم. برای تمام فریم‌ها f را محاسبه می‌کنیم، سپس داده‌های پرت را حذف می‌کنیم و میانگین آن‌ها را به عنوان f خروجی می‌دهیم. به صورت خلاصه به کمک تمام هموگرافی‌ها توانستیم یک تخمین خوب برای f پیدا کنیم. نتیجه‌ی محاسبات من، برای عکس‌ها با کیفیت اصلی برابر بود با:

$$f = 1524.66$$

تمام محاسبات بالا در تابع compute_f انجام می‌شود. در این تابع ابتدا هموگرافی را بر درایه‌ی $(3,3)$ اش تقسیم می‌کنیم. سپس به کمک معادلات بالا f را محاسبه می‌کنیم. در نهایت انحراف از معیار تمام f های محاسبه شده را به دست می‌آوریم و f هایی که بیش از دو انحراف از معیار فاصله دارند را حذف می‌کنیم. در نهایت نیز دوباره f را با میانگین گرفتن محاسبه می‌کنیم و خروجی می‌دهیم.

- در مرحله‌ی بعدی ما K را می‌سازیم.

- در حلقه‌ی بعدی، به کمک هر H_i ، R_i را محاسبه می‌کنیم. سپس با استفاده از کلاس Rotation ، تابع R_i خود را تجزیه می‌کنیم. سپس این زوایا را ذخیره می‌کنیم.
- در مرحله‌ی بعدی این زوایا را هموار می‌کنیم. این کار با استفاده از تابع get_smooth_rotations انجام می‌شود. روند کار این تابع نیز ساده است. مقدار یک زوایه برای فریم i را با استفاده از میانگین این زاویه در یک پنجره حول آن به دست می‌آوریم. در اصل انگار روی زاویه‌ها و در راستای تغییر فریم‌ها یک moving average به دست می‌آوریم (برای فریم‌های دو سر بازه که بازه‌ی آن‌ها یک طرفه می‌شد، کاری که انجام دادم این است که آرایه زوایا را از دو طرف با همان مقادیر دو سر آن extend کردم). نتیجه‌ی این moving average ها را می‌توانید در سه عکس x_rotations و y_rotations و z_rotations در پوشه‌ی outputs/temps ببینید (برای دیدن این تصاویر در هنگام اجرای کد، خط‌های کامنت شده‌ی انتهای تابع را می‌توانید آنکامنت کنید).
- در نهایت پس از هموار شدن زوایا، به کمک تابع create_rotation_matrix برای هر فریم یک ماتریس دوران جدید پیدا می‌کنم و ماتریس H' را که در بالا توضیح دادم می‌سازم. در نهایت نیز فریم را با ماتریس $H(H')^{-1}$ وارپ می‌کنیم و از تمامی آن‌ها یک ویدئو درست می‌کنم.

توجه: وقتی تصویرها را وارپ می‌کنیم، اتفاقی که می‌افتد این است که تصویرها لزوما بر پنجره‌ی قبلی خود fit نمی‌شوند (و این طبیعی است چون آن‌ها را مقداری جابجا کرده‌ایم). برای این که ویدیو معقول به نظر برسد، با کمک تابع make_frame_bigger هر فریم را نسبت به وسط آن کمی scale می‌کنیم (معادلا میشد اندازه‌ی هر فریم ویدیو را کمی کاهش داد) تا در اطراف ویدیو فضای سیاه اضافی باقی نماند.

توجه: در ابتدای متن به این نکته اشاره کردم که بین هر دو فریم اگر تنها دوران باشد، یک هموگرافی وجود دارد و داریم:

$$H = KRK^{-1}$$

روش محاسبه‌ی این عبارت به صورت زیر است (اگر تنها دوران داشته باشیم):

$$x_i = PX_i = K[R|t]X = KRX$$
$$x_{450} = PX_i = K[I|0]X = KX$$
$$x_{450} = Hx_i \Rightarrow Hx_i = KX \Rightarrow HKRX = KX \Rightarrow H = KR^{-1}K^{-1}$$

از طرفی وارون هر ماتریس دوران نیز یک ماتریس دوارن است و در نتیجه:

$$H = KR'K^{-1}$$

و به کمک این رابطه می‌توانیم زوایای R' را طبق توضیحات بالا پیدا کنیم. حال اگر زوایای به دست آمده را هموار کنیم و دوباره با آن یک ماتریس دوران مثل R'' بسازیم، آنگاه خواهیم داشت: $H' = KR''K^{-1}$ و بقیه مراحل نیز طبق توضیحاتی است که دادم.

سوال ۲

کد این قسمت به این صورت است که ابتدا مختصات دنیای واقعی نقاط تقاطع صفحه‌ی شطرنجی را می‌سازیم (در متغییر real_coords). سپس مراحل زیر را برای هر چهار بازه‌ی گفته شده انجام می‌دهیم:

به تعداد عکس‌هایی که داریم تعداد نقاط را تکثیر می‌کنیم (زیرا تابع آماده opencv نیاز دارد که به ازای هر عکس، یک بار به آن مختصات واقعی را بدهیم). سپس با کمک تابع findChessboardCorners نقاط تقاطع‌های صفحه‌ی شطرنجی را به دست می‌آوریم. اگر این تابع موفق به پیدا کردن این نقاط نشده بود، از تعداد بارهایی که نقاط واقعی را تکثیر کرده بودیم یک واحد کم می‌کنیم (هرچند این اتفاق در کد ما رخ نمی‌دهد). تابع cornerSubPix تنها کاری که انجام می‌دهد این است که به کمک گرادیان مختصات تقاطع را دقیق تر می‌کند. در نهایت نیز به کمک تابع calibrateCamera ، ماتریس دوربین را به دست می‌آوریم و چاپ می‌کنیم. نتایج به صورت زیر است (به ترتیب برای حالت ۱ تا ۴):

$$\begin{bmatrix} 2.93177372e+03 & 0.00000000e+00 & 9.11524247e+02 \\ 0.00000000e+00 & 2.95269004e+03 & 5.51574153e+02 \\ 0.00000000e+00 & 0.00000000e+00 & 1.00000000e+00 \end{bmatrix}$$

$$\begin{bmatrix} 3.00205033e+03 & 0.00000000e+00 & 8.81605705e+02 \\ 0.00000000e+00 & 2.99789552e+03 & 5.28453022e+02 \\ 0.00000000e+00 & 0.00000000e+00 & 1.00000000e+00 \end{bmatrix}$$

$$\begin{bmatrix} 3.04620014e+03 & 0.00000000e+00 & 7.18347521e+02 \\ 0.00000000e+00 & 3.03488142e+03 & 5.48591099e+02 \\ 0.00000000e+00 & 0.00000000e+00 & 1.00000000e+00 \end{bmatrix}$$

$$\begin{bmatrix} 2.98248908e+03 & 0.00000000e+00 & 8.36998212e+02 \\ 0.00000000e+00 & 2.99047508e+03 & 5.12824251e+02 \\ 0.00000000e+00 & 0.00000000e+00 & 1.00000000e+00 \end{bmatrix}$$

که در خانه‌های $(1,1)$ ، $(2,2)$ ، $(1,3)$ و $(2,3)$ به ترتیب f_x ، f_y ، p_x و p_y قرار گرفته‌اند.
همانطور که می‌بینید، فاصله‌های کانونی با تقریب تا حد خوبی به هم نزدیک هستند. اما Principal Point ها نه و میزان اختلاف آن‌ها قابل توجه است.
اگر فرض‌های اضافه شده در سوال را نیز اضافه کنیم (در کد می‌توانید خط ۴۴ را کامنت کنید و خط‌های ۴۵ تا ۵۰ را آنکامنت کنید)، می‌توانیم نتایج را دوباره محاسبه کنیم و این بار با استفاده از تمام فریم‌ها داریم:

$$\begin{bmatrix} 3.00513497e+03 & 0.00000000e+00 & 7.49500000e+02 \\ 0.00000000e+00 & 3.00513497e+03 & 4.99500000e+02 \\ 0.00000000e+00 & 0.00000000e+00 & 1.00000000e+00 \end{bmatrix}$$

که نتیجه می‌دهد:

$$f = 3005px$$

با توجه به نتایجی که گرفتیم، می‌توانیم برداشت کنیم که این روش تقریبا خوب عمل کرده است و توانسته است حتی با تعداد کمتری عکس نتیجه‌ی مشابه با تمام عکس‌ها بگیرد. اما در رابطه با Principal Point هرچه تعداد نمونه‌ی بیشتری داشته باشد به عدد بهتری همگرا شده است.