Image Fundamental

شیوا رادمنش

اطلاعات گزارش

تاریخ: ۱۳۹۹/۸/۱۳

واژگان کلیدی:

درونیابی دوخطی

تبدیل هندسی

Quantization

Image registration

چکیده

در این گزارش به نحوه‌ی تبدیلات هندسی دوران و image registration، درونیابی و بازیابی پیکسل‌های از دست رفته، کوانتیزه کردن تصاویر و کاهش تعداد سطوح خاکستری آنها، downsampling و upsampling با روش های مختلف و همچنین بررسی نتایج حاصل از این اعمال پرداخته شده است.

۱- مقدمه

نوشتار حاضر به بررسی شیوه‌ی اعمال تبدیلات هندسی، کاهش تعداد سطوح خاکستری تصاویر ، نمونه برداری بر روی تصاویر و همچنین نحوه‌ی به دست آوردن اطلاعات از دست رفته‌ی برخی از پیکسل های تصویر هنگام انجام اعمال هندسی و نمونه برداری، به روش درونیابی دوخطی(bilinear interpolation) با استفاده از زبان برنامه‌نویسی python می‌پردازد.

۲- سوال ۱.۱.۱

در این بخش دو تصویر(تصویر ورودی و تصویر مرجع) از یک صحنه در زمان‌های متفاوت در اختیار داریم. هدف این است با اعمال تبدیلات هندسی تصویر ورودی را به تصویری تبدیل کنیم که با تصویر مرجع همتراز شود.

یکی از چالش‌های حل این مسئله این است که بر خلاف سایر تبدیلات هندسی که تابع تبدیل ورودی به خروجی مشخص است، در اینجا تابع تبدیل هندسی از قبل مشخص نیست و باید آن را به گونه‌ای تخمین بزنیم.

یکی از رویکردها برای حل این مسئله استفاده از نقاط کنترلی است. این نقاط، نقاط متناظر در تصویر ورودی و مرجع هستند که مختصات آنها در هر دو تصویر معلوم است.

فرض کنید که ۴ نقطه‌ی کنترلی داریم که مختصات دقیق هر نقطه در تصاویر ورودی و مرجع را می‌دانیم. (x, y) مختصات نقطه‌ی کنترلی در تصویر مرجع و (v, w)

مختصات نقطه‌ی کنترلی در تصویر ورودی می‌باشد.
یک مدل ساده ی تقریب دو خطی برای تخمین تابع تبدیل هندسی عبارت است از:

$$x = c_1 v + c_2 w + c_3 vw + c_4 \quad (۱)$$
$$y = c_5 v + c_6 w + c_7 vw + c_8 \quad (۲)$$

اگر ۴ نقطه‌ی کنترلی داشته باشیم، می‌توان با حل یک دستگاه ۸ معادله - ۸ مجهولی می‌توان ضرایب c_1 تا c_8 را بدست آورد.
حال می‌توان مختصات هر نقطه‌ی ورودی(v, w) را با استفاده از روابط (۱) و (۲) ، را تبدیل به مختصات نقطه‌ی نظیر آن در تصویر خروجی(x, y) کرد. حال باید مختصات تصویر حاصل را با تصویر مرجع همتراز کرد.

۳- سوال ۱.۱.۲

در این بخش با استفاده از دو تصویر داده شده‌ی Car1 و Car2، باید دو تصویر را به یکدیگر به طوری بدوزیم که تصویر حاصل، یک تصویر پاناروما گردد.
برای انجام این کار در ابتدا دو تصویر را با استفاده از تابع imread کتابخانه openCV خوانده و در متغیر های img1 و img2 نگهداری می‌کردیم. سپس به تعداد ۴ پیکسل در تصویر img1 که تصویر مرجع ما در نظر گرفته شده است در نظر گرفته و سعی می‌کنیم معادل آن را در تصویر img2 که تصویر ورودی ما است، بیابیم.

نقاط انتخاب شده برای این بخش به صورت زیر می‌باشند:

نقاط تصویر img1:

$tp11 = (379, 834)$
$tp12 = (318, 763)$
$tp13 = (379, 880)$
$tp14 = (451, 564)$

نقاط تصویر img2:

$tp21 = (398, 413)$
$tp22 = (338, 349)$
$tp23 = (399, 457)$
$tp24 = (471, 142)$

برای بدست آوردن تابع تبدیل هندسی، از معادله (۱) و (۲) استفاده شده است.
ماتریس مرجع (reference) بدست آمده به صورت زیر می‌باشد.

$$\begin{bmatrix} 379 & 318 & 379 & 451 \\ 834 & 763 & 880 & 564 \end{bmatrix}$$

برای حل این معادلات، من با استفاده از روش ضرب ماتریس ها، ضریاب c1 تا c8 را بدست می‌آوریم. معادله بدست آوردن ضرایب c به صورت زیر می‌باشد.

$$\begin{bmatrix} 379 & 318 & 379 & 451 \\ 834 & 763 & 880 & 564 \end{bmatrix} \times \begin{bmatrix} 398 & 338 & 399 & 471 \\ 413 & 349 & 457 & 142 \\ 164374 & 117962 & 182343 & 66882 \\ 1 & 1 & 1 & 1 \end{bmatrix}^{-1}$$

و همچنین ماتریس ضرایب (c) به صورت زیر می‌باشد.

$$\begin{bmatrix} c1 & c2 & c3 & c4 \\ c5 & c6 & c7 & c8 \end{bmatrix}$$

حال با داشتن این ماتریس، می‌توان با ضرب دکارتی ماتریس هر نقطه از تصویر ورودی بر روی تصویر مرجع، نقاط ثانویه را بدست آوریم.

شکل ۱- خروجی تصویر پاناروما

شکل ۲- خروجی پاناروما تابع stitched در کتابخانه‌ی opencv

در شکل ۱ مشاهده می‌شود که تبدیل هندسی باعث از دست رفتن دیتای برخی از پیکسل های تصویر ورودی شده است. برای بازیابی مقادیر این پیکسل ها می‌توان از روش‌های درونیابی استفاده کرد.
همانطور که از مقایسه شکل ۱ و ۲ برداشت می‌شود، برای تشکیل شکل ۲ از روش بدست آوردن feature point استفاده شده است که این خود نیاز به پردازش های زیاد و با الگوریتم های خاص است. این در حالی است که در شکل ۱، نقطه ها با استفاده از یک حدس و بدون دلیل علمی بدست آمده اند. تنها ویژگی ای که برای نقاط مدنظر قرار گرفته بود، گوشه بودن و تفاوت آشکار در آن نقطه بود. معیار دیگری که برای انتخاب نقاط استفاده شد، پراکندگی این نقاط بود که بتواند سطح یک صفحه را پوشش دهد. پس با این توصیفات، می‌توان برداشت کرد که تصویر خروجی ما نمی‌تواند همانند تصویر خروجی با استفاده از کلاس Stitcher_create شود.

۴- سوال ۱.۱.۳

در این بخش به دوران و سپس درونیابی تصویر Elaine پرداخته شده است.

۱-۴- تبدیل دوران با زوایای ۳۰ و ۴۵ و ۸۰ درجه

برای اعمال تبدیل دوران با زاویه‌ی Θ ، با استفاده از روابط (۳) و (۴) مختصات جدید هر پیکسل از تصویر ورودی، در تصویر دوران یافته محاسبه می‌شود.
در این رابطه (v, w) مختصات پیکسل در تصویر ورودی و (x, y) مختصات پیکسل در تصویر دوران یافته می‌باشد.

$$x = v\cos\Theta - w\sin\Theta \quad (۳)$$

$$y = v\cos\Theta + w\sin\Theta \quad (۴)$$

ماتریس زیر ماتریس affine مربوط به دوران با زاویه‌ی Θ می‌باشد.

$$\begin{bmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

تصویر دروران یافته با استفاده از روابط (۳) و (۴) ، تصویری خواهد بود که تبدیل دوران بر روی آن نسبت به نقطه‌ی (0, 0) (گوشه‌ی بالا سمت چپ تصویر) اعمال. شده است. برای دوران تصویر حول مرکز تصویر باید نقطه‌ی مرکز را (0, 0) در نظر گرفت یعنی تبدیل دوران را بجای نقاط (v, w) بر روی نقاط (v-a, w-b) اعمال کنیم که در اینجا (a, b) مختصات نقطه‌ی مرکزی می‌باشد و همچنین پس از اعمال تبدیل، مقدار پیکسل (v, w) را به جای نقطه‌ی (x, y) در نقطه‌ی (x+a, y+b) قرار دهیم. نتیجه ی خروجی هر دو حالت دوران برای زاویه ی ۳۰ درجه در شکل۱ و شکل۲ قابل مشاهده می‌باشد.

شکل ۳- تصویر دوران یافته با زاویه‌ی ۳۰ درجه حول نقطه‌ی (0 , 0) بدون اعمال درونیابی

شکل ۴- تصویر دوران یافته با زاویه‌ی ۳۰ درجه حول مرکز عکس، بدون اعمال درونیابی

شکل ۵- تصویر دوران یافته با زاویه‌ی ۴۵ درجه حول مرکز عکس، بدون اعمال درونیابی

شکل ۶- تصویر دوران یافته با زاویه‌ی ۸۰ درجه حول مرکز عکس، بدون اعمال درونیابی

۲-۴- درونیابی تصاویر با استفاده از روش دو خطی

پس از اعمال تبدیل هندسی مشاهده می‌شود که مقدار برخی پیکسل‌های تصویر خروجی برابر با صفر می‌باشد. این مسئله بیان گر از بین رفتن دیتای برخی پیکسل‌ها می‌باشد. در دوران با زوایایی مانند ۱۸۰ و ۹۰ درجه چنین اتفاقی نمی‌افتد و مختصات همه ی پیکسل‌هایی که از map کردن پیکسل های ورودی به تصویر خروجی بدست آمده در تصویر خروجی جای مشخصی دارند. همچنین مشاهده می‌شود که تصویر حاصل از دوران ۸۰ درجه(که به ۹۰ درجه نزدیک‌تر است) نسبت به تصاویر حاصل از دوران ۳۰ درجه و ۴۵ درجه،نقاط سیاه یا در واقع دیتای از دست رفته‌ی کمتری دارد.

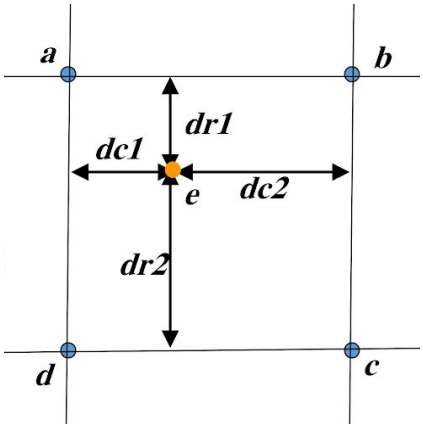
برای بازیابی دیتای از دست رفته می‌توان از روش‌های درونیابی استفاده کرد. در اینجا از درونیابی دوخطی استفاده شده است.

در روش درونیابی دوخطی، با پردازش تصویر تبدیل یافته، نقاط سیاه را پیدا کرده و با ضرب معکوس ماتریس affine در ماتریس مختصات نقطه‌ی سیاه، مختصات این نقطه را در تصویر اولیه را بدست می‌آوریم.

مختصات بدست آمده را نقطه‌ی (p, q) می‌نامیم. سپس مختصات نقاط احاطه‌کننده‌ی(p, q) را بدست میاوریم. این نقاط عبارتند از:

$$a = (p-1,\ q-1)$$
$$b = (p-1,\ q+1)$$
$$c = (p+1,\ q+1)$$
$$d = (p+1,\ q-1)$$

با استفاده از این نقاط، می‌توان مقدار نقطه‌ی (p, q) را با توجه به رابطه‌ی (۵) بدست آورد.



شکل ۷- محاسبه‌ی مقدار هر پیکسل با استفاده از درونیابی دوخطی

$$e = (a \times dc2 \times dr2) + (b \times dc1 \times dr2) + (d \times dc1 \times dr2) + (c \times dc1 \times dr1) \quad (۵)$$

شکل ۸- تصویر دوران یافته با زاویه‌ی ۳۰ درجه پس از درونیابی

شکل ۹- تصویر دوران یافته با زاویه‌ی ۴۵ درجه پس از درونیابی

شکل ۱۰- تصویر دوران یافته با زاویه‌ی ۸۰ درجه پس از درونیابی

۵- سوال ۱.۲.۱

در این بخش به بررسی و مقایسه‌ی تاثیر quantization در ۸، ۱۶، ۳۲، ۶۴، ۱۲۸ سطح بر یک تصویر و

تصویر حاصل از همسان سازی هیستوگرام در تصویر اولیه، پرداخته می‌شود.

۵-۱- همسان سازی هیستوگرام

با استفاده از تابع equalizeHist در کتابخانه‌ی opencv عمل همسان سازی هیستوگرام بر روی تصویر Barbara انجام شده است. مشاهده می‌شود که کنتراست تصویر بهبود یافته است.

شکل ۱۱- مقاسیه‌ی تصویر اصلی و تصویر حاصل از همسان سازی هیستوگرام

۵-۲- اعمال quantization در ۸، ۱۶،۳۲،۶۴،۱۲۸ سطح خاکستری

برای کاهش سطوح خاکستری از ۲۵۶ سطح به تعداد سطح دلخواه از رابطه‌ی (۶) استفاده شده است. در این رابطه c برابر با ضریب quantization می‌باشد.

$$c = 256 / level$$

$$result = floor(\frac{pixel\ value}{c}) \times c \quad (۶)$$

شکل ۱۲- تصویر کوانتیزه شده در ۸ سطح خاکستری

شکل ۱۳- تصویر کوانتیزه شده در ۸ سطح خاکستری که بر روی آن همسان سازی هیستوگرام انجام شده بود.

شکل ۱۴- تصویر کوانتیزه شده در ۱۶ سطح خاکستری

شکل ۱۶- تصویر کوانتیزه شده در ۳۲ سطح خاکستری

شکل ۱۵- تصویر کوانتیزه شده در ۱۶ سطح خاکستری که بر روی آن همسان سازی هیستوگرام انجام شده بود.

شکل ۱۷- تصویر کوانتیزه شده در ۳۲ سطح خاکستری که بر روی آن همسان سازی هیستوگرام انجام شده بود.

شکل ۱۸- تصویر کوانتیزه شده در ۶۴ سطح خاکستری

شکل ۲۰- تصویر کوانتیزه شده در ۱۲۸ سطح خاکستری

شکل ۱۹- تصویر کوانتیزه شده در ۶۴ سطح خاکستری که بر روی آن همسان سازی هیستوگرام انجام شده بود.

شکل ۲۱- تصویر کوانتیزه شده در ۱۲۸ سطح خاکستری که بر روی آن همسان سازی هیستوگرام انجام شده بود.

در تصاویر بالا مشاهده می‌شود که با کاهش تعداد سطوح خاکستری بخشی از جزئیات تصویر حذف می‌شود. تصاویر با تعداد سطوح خاکستری ۱۲۸و ۶۴ (شکل۱۸ و شکل۲۰) از نظر بصری کاملا شبیه هستند

در حالی که در تصویر با ۳۲ سطح خاکستری (شکل۱۶) مجموعه ای از ساختارها به طور نامحسوس وجود دارد که شدت خاکستری آنها ثابت است. در تصویر با ۱۶ سطح خاکستری(شکل۱۴) این ساختار ها واضح تر هستند.

همچنین با مقایسه‌ی کاهش تعداد سطوح خاکستری در تصویر معمولی و تصویری که همسان سازی هیستوگرام بر آن اعمال شده است، مشاهده می‌شود که تصویری که بر روی آن همسان سازی هیستوگرام اعمال شده به طور کلی جزئیات کمتری را از دست می‌دهد و تصویر آن از نظر بصری به تصویر اصلی نزدیک تر است.

۳-۵- مقایسه‌ی تصاویر با استفاده از خطای میانگین مربعات(MSE)

برای مقایسه‌ی تصاویر حاصل از معیاری به نام خطای میانگین مربعات(MSE) استفاده شده است. نتایج این مقایسه در جدول ۱ قابل مشاهده است.

level	8	16	32	64	128
original	91.5	76.1	17.5	3.4	0.4
equalizeHist	109.9	107.9	106	106.3	109.5

جدول ۱- MSE محاسبه شده برای تعداد سطوح خاکستری مختلف نسبت به تصویر اصلی با دقت یک رقم اعشار

مشاهده می‌شود که در تصویر اولیه با افزایش تعداد سطوح خاکستری MSE به طور قابل توجهی کاهش می‌یابد آنقدر که MSE برای تصویر اولیه با ۱۲۸ سطح خاکستری نزدیک به صفر است. همانطور که از نظر بصری بررسی شد تصاویر با ۱۲۸ و ۶۴ سطح خاکستری اختلاف MSE بسیار کمی دارند. با کاهش تعداد سطوح خاکستری به ۳۲ سطح MSE مقداری افزایش یافته و همانطور که اختلاف تصویر اصلی با تصویر با ۱۶ خاکستری از نظر بصری کاملا واضح بود، با کاهش تعداد سطوح خاکستری تا ۱۶ سطح MSE مقدار قابل توجهی زیاد می‌شود.

همچنین مشاهده می‌شود که با وجود اینکه از نظر بصری جزئیات بیشتری در تصاویری که همسان سازی هیستوگرام بر روی آنها انجام شده وجود دارد، MSE این تصاویر بسیار بالا بوده و همچنین با تغییر تعداد سطوح خاکستری تغییر چشمگیری در MSE مشاهده نمی‌شود.

۶- سوال ۱.۲.۲

در این بخش در ابتدا عمل downsampling با استفاده از روش میانگین گیری و روش حذف سطر و ستون ها بر روی تصویر Goldhill اعمال می‌شود. سپس بر روی تصویر نهایی عمل upsampling با استفاده از درونیابی دوخطی و روش pixel replication انجام می‌شود.

۱-۶- عمل downsampling

برای انجام این عمل با یک نرخ نمونه برداری خاص، پیکسل های تصویر ورودی را نمونه برداری می‌کنیم. نرخ نمونه برداری در این مسئله ۲ می‌باشد. یعنی برای هر دو پیکسل در یک سطر و همچنین در یک ستون، یک پیکسل در نظر می‌گیریم که مقدار این پیکسل در هر روش نمونه برداری متفاوت است. تصویر حاصل از نمونه برداری با نرخ ۲، تصویری با تعداد سطر و ستون هایی نصف تعداد سطر و ستون های تصویر اصلی می‌باشد.

شکل ۲۲- تصویر Goldhill قبل از اعمال downsampling با ابعاد ۵۱۲*۵۱۲

۶-۱-۱- downsampling با استفاده از روش فیلتر میانگین

در این روش برای محاسبه‌ی مقدار پیکسل مورد نظر در تصویر خروجی یک فیلتر ۲*۲ در نظر گرفته و آن را روی تصویر ورودی حرکت می‌دهیم. مقدار پیکسل متناظر در تصویر خروجی میانگین مقدار پیکسل های درون فیلتر می‌باشد.

شکل ۲۳- تصویر downsample شده با روش فیلتر میانگین با ابعاد ۲۵۶*۲۵۶

۶-۱-۲- downsampling با استفاده از روش حذف سطر و ستون

در این روش برای محاسبه‌ی مقدار پیکسل مورد نظر از هر پیکسل در هر سطر و در هر ستون مقدار یکی را به عنوان مقدار پیکسل متناظر در تصویر خروجی در نظر می‌گیریم و پیکسل‌های دیگر حذف می‌شوند.

در این مسئله مقدار پیکسل در مختصات (x, y) در تصویر خروجی برابر با مقدار پیکسل $(x/2, y/2)$ در تصویر ورودی در نظر گرفته شده است.

شکل ۲۴- تصویر downsample شده با روش حذف سطر و ستون با ابعاد ۲۵*۲۵۶

۲-۶- عمل upsampling

در این مرحله از روی تصویر downsample شده در مرحله‌ی قبل تصویر اولیه را بازیابی می‌کنیم.

در حین انجام عمل downsampling مقدار بسیاری از پیکسل ها از بین رفته و برای تبدیل تصویر با سایز ۲۵۶ *۲۵۶ به تصویری با سایز ۵۱۲*۵۱۲ لازم است مقداری برای سایر پیکسل‌ها تعریف کنیم.

برای تعیین مقادیر پیکسل‌هایی که مقدار ندارند، روش‌های مختلفی وجود دارد که در این گزارش به دو مورد از آنها اشاره شده است.

۱-۲-۶- upsampling با استفاده از روش pixel replication

در این روش به ازای مقدار پیکسل هایی از تصویر خروجی که مقدار ندارند مقدار پیکسل‌های متناظر در تصویر downsample شده تکرار می‌شود.

شکل ۲۵- تصویر upsample شده با روش pixel replication روی تصویر downsample شده با روش حذف سطر و ستون

شکل ۲۶- تصویر upsample شده با روش pixel replication روی تصویر downsample شده با روش فیلتر میانگین

۲-۲-۶- upsampling با استفاده از روش درونیابی دو خطی

در این روش مقدار پیکسل های تصویر خروجی که مقدار ندارند را با استفاده از روش درونیابی دوخطی که در بخش ۲-۵ توضیح داده شد، بدست می‌آوریم.

شکل ۲۷- تصویر upsample شده با روش درونیابی دوخطی روی تصویر downsample شده با روش حذف سطر و ستون

شکل ۲۸- تصویر upsample شده با روش درونیابی دوخطی روی تصویر downsample شده با روش فیلتر میانگین

شکل ۲۹- تصویر upsample شده با روش درونیابی دوخطی روی تصویر downsample شده با روش فیلتر میانگین در حالت zoom

شکل ۳۰- تصویر upsample شده با روش درونیابی دوخطی روی تصویر downsample شده با روش حذف سطر و ستون در حالت zoom

شکل ۳۱- تصویر upsample شده با روش pixel replication روی تصویر downsample شده با روش حذف سطر و ستون در حالت zoom

شکل ۳۲- تصویر upsample شده با روش pixel replication روی تصویر downsample شده با روش فیلتر میانگین در حالت zoom

مشاهده می‌شود که تصاویری که با فیلتر میانگین downsample شده اند مات تر هستند و لبه ها در این تصاویر نرم و smooth تر هستند.

با مقایسه‌ی شکل ۳۱ و شکل ۳۰ که هر دو با روش حذف سطر و ستون downsample شده اند،مشاهده می‌کنیم که تصویری که با روش درونیابی دوخطی upsample شده مات تر از تصویری است که با روش دیگر Upsample شده است.

با مقایسه ی شکل ۲۹ و ۳۲ که هر دو با استفاده از روش فیلتر میانگین downsample شده اند، مشاهده می‌شود که تصویر upsample شده با استفاده از روش pixel replication نسبت به تصویر upsample شده با درونیابی دوخطی، مات تر است.

مات بودن تصاویر در شکل‌های ۳۲و ۲۹ به دلیل عمل downsampling با استفاده از فیلتر میانگین می‌باشد اما اینکه تصویر ۳۲ نسبت به تصویر ۲۹ مات تر است به این دلیل است که upsampling با روش pixel replication باعث می‌شود که پیکسل‌های تصویر downsample شده تکرار شوند و از آنجایی که هر دو تصویر با فیلتر میانگین downsample شده اند، روش pixel replication باعث می‌شود تصویر مات تر شود.

۳-۶- مقایسه ی تصاویر با استفاده از خطای میانگین مربعات(MSE)

	Pixel replication	Bilinear interpolation
averaging	35.5	38

Remove row&column	23	27

جدول ۲- MSE محاسبه شده برای تصاویر نمونه برداری شده با روش های متفاوت

مشاهده می‌شود که تصاویری که با روش درونیابی دوخطی upsample شده اند، MSE بیشتری نسبت به تصاویر upsample شده با روش pixel replication دارند.

همچنین MSE تصاویری که با روش حذف سطر و ستون downsample شده‌اند به طور نسبتا قابل توجهی از تصاویری که با فیلتر میانگین downsample شده اند، کمتر است.

مراجع

[1] Digital Image Processing, Rafael C. Gonzalez, Rechard E. Wood

Appendix

1.1.2

```python
import cv2
import numpy as np
from math import floor, ceil

img1 = cv2.imread('src/img/Car1.jpg', 0)
img2 = cv2.imread('src/img/Car2.jpg', 0)

r1, c1 = img1.shape
r2, c2 = img2.shape

# tie point for Car1
tp11 = (379, 834)
tp12 = (318, 763)
tp13 = (379, 880)
tp14 = (451, 564)
# tp14 = (382, 949)
```

```
# tie points for Car2
tp21 = (398, 413)
tp22 = (338, 349)
tp23 = (399, 457)
tp24 = (471, 142)
# tp24 = (403, 520)

ref = np.array(([ tp11[0], tp12[0], tp13[0], tp14[0]], [tp11[1], tp12[1],
tp13[1], tp14[1]]))

input_mat = np.array((
                    [tp21[0], tp22[0], tp23[0], tp24[0]],
                    [tp21[1], tp22[1], tp23[1], tp24[1]],
                          [tp21[0] * tp21[1], tp22[0] * tp22[1], tp23[0] *
tp23[1], tp24[0] * tp24[1]],
                    [1, 1, 1, 1]
))

c = np.matmul(ref, np.linalg.inv(input_mat))

origin_mat = np.array(([0], [0], [0], [1]))
org_new_cord = np.matmul(c, origin_mat)
org_x = floor(org_new_cord[0])

res_img = np.zeros(shape=(r1 +r2 // 4, c1 + c2))
res_img[0: r1, 0:c1] = img1.astype('uint8')

for i in range(r2):
   for j in range(c2):
       pmat = np.array(([i], [j], [i * j], [1]))
       new_cord = np.matmul(c, pmat)
       x = floor(new_cord[0])
       y = floor(new_cord[1])
       if x < r1 + r2 and x >= 0 and y < c1 + c2 and y >= 0:
           if (res_img[x, y] == 0):
               res_img[x, y] = img2[i, j]
           count += 1

cv2.imshow('img', res_img.astype('uint8'))
cv2.waitKey(0)
```

```
cv2.imwrite('1_1_2.jpg', res_img.astype('uint8'))
```

1.1.3

```
def rotate(img, degree):
                affine      =      np.array([[math.cos(math.radians(degree)),
math.sin(math.radians(degree)), 0],
                                             [-(math.sin(math.radians(degree))),
math.cos(math.radians(degree)), 0],
                     [0, 0, 1]])
   rotated = np.zeros_like(img)
   half_x = np.size(img, 0) / 2
   half_y = np.size(img, 1) / 2

   for v in range(np.size(img, 0)):
       for w in range(np.size(img, 1)):
           x, y = mapped_coordinate(v - half_x, w - half_y, affine)
           x = round(x + half_x)
           y = round(y + half_y)
            if(x < np.size(img, 0) and y < np.size(img, 1) and x > 0 and y
> 0):
               rotated[x, y] = img[v, w]
   return rotated
```

```
def mapped_coordinate(v, w, affine):
   in_coordinate = np.array([v, w, 1])
   out_coordinate  = np.matmul(in_coordinate, affine)
   x = out_coordinate[0]
   y = out_coordinate[1]
   return x, y
```

```
def bilinear(img, rotated, degree):
                affine      =      np.array([[math.cos(math.radians(degree)),
math.sin(math.radians(degree)), 0],
                                             [-(math.sin(math.radians(degree))),
math.cos(math.radians(degree)), 0],
```

```
                    [0, 0, 1]])
    half_x = np.size(img, 0) / 2
    half_y = np.size(img, 1) / 2
    invert_affine = np.linalg.inv(affine)
    for x in range(np.size(rotated, 0)):
        for y in range(np.size(rotated, 1)):
            if  rotated[x, y] == 0 :
                        v, w = mapped_coordinate(x - half_x , y - half_y,
invert_affine)
                v = v + half_x
                w = w + half_y
                 if(v < np.size(img, 0) - 1 and w < np.size(img, 1) - 1 and
v > 0 and w > 0):
                    rotated[x, y] = bilinear_val(img, v, w)
    return rotated
```

```
def bilinear_val(img, v, w):
    ax = math.floor(v)   #ax = bx = floor(v)
    ay = math.floor(w)   #ay = dx = floor(w)
    bx = math.floor(v)
    by = math.ceil(w)    #by = cy = ceil(w)
    dx = math.ceil(v)   #cx = dx = ceil(v)
    dy = math.floor(w)
    cx = math.ceil(v)
    cy = math.ceil(w)

    dr1 = abs(v - ax)
    dr2 = abs(v - dx)
    dc1 = abs(w - ay)
    dc2 = abs(w - by)
     value = (img[ax, ay] * dr2 * dc2) + (img[bx, by] * dc1 * dr2) +
(img[dx, dy] * dr1 * dc2) + (img[cx, cy] * dr1 * dc1)
    return value
```

1.2.2

```
def downsample(img, rate):
```

```
    downsampled  =np.zeros([int(np.size(img,  0)/rate),  int(np.size(img,
1)/rate)], dtype = np.uint8)

   for i in range(np.size(downsampled, 0)):
       for j in range(np.size(downsampled, 1)):
           downsampled[i, j] = img[i*rate, j*rate]

   return downsampled
```

```
def avg_downsample(img, rate):
    avg_downsampled =np.zeros([int(np.size(img, 0)/rate), int(np.size(img,
1)/rate)], dtype = np.uint8)

   for i in range(np.size(avg_downsampled, 0)):
       for j in range(np.size(avg_downsampled, 1)):

           summ = 0
           #calculate the average
           for x in range(i*rate, i*rate + rate):
               for y in range(j*rate, j*rate + rate):
                   summ += img[x, y]

           avg = round(summ / (rate*rate))
           avg_downsampled[i, j] = avg

   return avg_downsampled
```

```
def replication_upsample(img, rate):
    upsampled = np.zeros([np.size(img, 0)* rate, np.size(img, 1)* rate],
dtype = np.uint8)

   for i in range(np.size(upsampled, 0)):
       for j in range(np.size(upsampled, 1)):
```

```
            upsampled[i, j] = img[math.floor(i/2), math.floor(j/2)]

    return upsampled
```

```
def bilinear_upsample(img, rate):
    upsampled = np.zeros([np.size(img, 0)* rate, np.size(img, 1)* rate],
dtype = np.uint8)

    for x in range(np.size(upsampled, 0)):
        for y in range(np.size(upsampled, 1)):
            v = x/rate
            w = y/rate
            if (v < np.size(img, 0)-1 and w < np.size(img, 1)-1):
                if(math.floor(v) == v or math.floor(w) == w):
                    upsampled[x, y] = img[int(v), int(w)]
                else:
                    upsampled[x, y] = bilinear_val(img, v, w)
    return upsampled
```

1.2.1

```
def quantize(img, level):
    coef = 256 / level
    quantized = np.floor(np.true_divide(img, coef)) * coef
    quantized = quantized.astype("uint8")
    return quantized
```