

مفاهیم بنیادی

سحر شیخ الاسلامی

اطلاعات گزارش

چکیده

تبديل های هندسی، بخش مهمی از بینایی کامپیووتر هستند، به کمک این تبدیلات می توان سایز و جهت عکس را عوض کرد. تبدیل های هندسی به دو روش مستقیم و معکوس انجام می گیرد. در این گزارش علاوه بر بررسی تبدیل های هندسی، با مفاهیم مقدماتی بینایی کامپیووتر از جمله چندی سازی آشنا می شویم.

تاریخ: ۱۳۹۹/۸/۱۴

واژگان کلیدی:

Geometric transportaion
Interpolation
Image registration
Qunatization
Panorama Image
Rotate
Down_sample
Up_Sample

-۱- مقدمه

برای کار با تصاویر دیجیتال، تغییر سایز و یا چرخش آنها یکی از مهمترین و پرکاربردترین عملیات ها است. در انجام این عملیات یکی از نکات مهم از دست ندادن بخشی از عکس می باشد. با استفاده از روش های مختلف می توان عملیات نامبرده شده را دقیق انجام داد.

$$X = c_1x + c_2y + c_3$$

$$Y = c_4x + c_5y + c_6$$

با بدست آوردن ضرایب، هر نقطه از تصویر اول به تصویر دوم map می شود. در این معادله فرض شده است که بین دو عکس خطی است. با توجه به اینکه تنها تفاوت دو عکس زمان دو عکس بوده است و زاویه و راستای دوربین (بنابراین عکس) عوض نشده است می توان حدس زد دو عکس رابطه‌ی خطی دارند. البته میتوان از معادله های پیچیده‌تری برای رابطه‌ی بین دو عکس استفاده کرد، و فرض کرد که دو عکس، رابطه‌ی غیر خطی دارند. توضیحات بیشتر و رابطه‌های غیر خطی در قسمت ۱.۱.۲ توضیح داده شده است.

قسمت ۱.۱.۲:

تطبیق عکس (image registration) یکی از کاربردهای مهم پردازش تصویر است که راستا و جهت و زاویه های متفاوتی از یک صحنه را بر هم منطبق می کند. در تطبیق عکس، یک عکس ورودی و یک عکس مرجع داریم، که قرار است عکس ورودی را با تبدیل های هندسی، هم راستای عکس مرجع کنیم. برخلاف تبدیل هایی مانند چرخش و shear که ماتریس تبدیل هندسی آنها در دسترس و شناخته شده است، ماتریس های هندسی تطبیق عکس شناخته شده نیستند و باید آنها را بدست بیاوریم.

-۲- شرح تکنیکال

۱.۱.۱:

در این قسمت دو تصویر از یک مکان ثابت که در زمان های متفاوت گرفته شده است، به ما داده شده است و از ما میخواهد یکتابع و یا map پیدا کنیم که این دو عکس را به هم تبدیل کند.

در واقع اگر عکس اول را A و عکس دوم را I در نظر بگیریم، ماتریسی مانند A می خواهیم که:

$$JA = I$$

در واقع این ماتریس با گرفتن پیکسل و مشخصات تصویر اول، به تصویر دوم می رسد و خروجی تصویر اول است در راستای تصویر دوم.

برای انجام این عملیات، با فرض اینکه سه نقطه داشته باشیم، میتوانیم هر نقطه از تصویر اول را تابعی از تصویر اول در نظر بگیریم:

۵. نقاط ابتدایی عکس دوم را ماتریس A ضرب میکنیم
تا محدوده‌ی شروع تصویر دوم بدست بیاید.
۶. از محدوده شروع عکس دوم تا انتهای عکس به ازای هر پیکسل:
- ۶.۱. پیکسل را در A^{-1} ضرب می‌کنیم.
- ۶.۲. با استفاده از interpolation، نقطه معادل آن را بدست می‌آوریم.
- ۶.۳. نقطه بدست امده را در موقعیت مکانی مناسب قرار می‌دهیم.
۷. عکس اول را روی عکس تطبیق داده شده، کپی می‌کنیم.
- با توجه به اینکه راستای عکس دوم، در راستای عکس اول شده است و شروع عکس دوم نیز تطبیق داده شده است، کپی کردن عکس اول یک عکس پاناروما می‌سازد.(خروجی در قسمت نتایج)

قسمت ۱.۱.۳

درون یابی روشی است که از آن در عملیات‌های مانند بزرگنمایی، کوچک‌نمایی، چرخاندن عکس و تصحیح‌های هندسی استفاده می‌شود.

هنگام انجام عملیاتی مانند بزرگنمایی و چرخاندن عکس، در صورت استفاده از ماتریس affine و بدون استفاده از عملیات‌هایی، به ازای برخی از پیکسل‌های مقصد، هیچ مقداری پیدا نخواهد شد و عکس شامل نقاط سیاهی خواهد بود.(شکل ۱)

درون یابی پرسه‌ای است که در آن با استفاده از پیکسل‌هایی که مقدار آنها را می‌دانیم، مقادیر پیکسل‌هایی که مقدار آنها را نمی‌دانیم تخمین می‌زنیم.

در میان روش‌های درون یابی، سه روش bilinear interpolation و neighbor interpolation و bicubic interpolation هستند.

در روش نزدیکترین همسایه، مشخصات هر پیکسل را در معکوس ماتریس affine ضرب میکنیم. می‌دانیم ماتریس affine (مثلاً ماتریس A)، ماتریسی است که هر پیکسل از تصویر مبدأ را به تصویر مقصد می‌برد.

$$xA = y$$

که X یک بردار است که به ترتیب شامل موقعیت عرضی، طولی و شدت رنگ پیکسل دلخواه از عکس مبدأ است. y مشابه برای عکس مقصد)

پس با استفاده از رابطه زیر میتوان هر پیکسل از مقصد را به یک پیکسل از مبدأ map کرد.

میتوان فرض کرد هر نقطه از عکس ورودی تابعی از عکس مرجع است. تابع زیر یکی از تابع‌های ممکن است.

$$X' = c_0x + c_1y + c_2xy + c_3$$

$$Y' = c_4x + c_5y + c_6xy + c_7$$

که X و Y نقاط عکس ورودی و X' و Y' نیز نقاط عکس مبدا هستند. با بدست آوردن این ضرایب، تبدیلی از عکس مبدا به عکس مرجع خواهیم داشت.

می‌توان از معادله‌های پیچیده تری نیز استفاده کرد که دقت بالاتری می‌دهند.

$$X = c_0x + c_1y + c_2xy + c_3x^2 + c_4y^2 + c_5$$

$$Y = c_6x + c_7y + c_8xy + c_9x^2 + c_{10}y^2 + c_{11}$$

حال کافی است 6 نقطه معادل از دو عکس پیدا کرده و ضرایب را با استفاده از این نقاط بدست بیاوریم. نکته مهم و قابل توجه این است که نقاط انتخاب شده، باید پراکندگی قابل قبولی داشته باشند. در صورتی که نقاط انتخابی همه از یک محدوده خاص باشند، خروجی می‌تواند خیلی نا دقیق باشد.(در قسمت نتایج، با شکل نشان داده شده است).

در سوال از ما خواسته شده است که با استفاده از دو عکس داده شده، که دو عکس از یک مکان و در راستا‌های متفاوت است، تصویر پانارومایی بسازیم. برای انجام این کار ابتدا عکس دوم را با توجه به توضیحات داده شده، در راستای عکس اول قرار می‌دهیم. اگر فقط ضرایب c_0 تا c_{11} را بدست آوریم، برای ساختن تصویر دوم، نمی‌توان از interpolation استفاده کرد.(نقاط سیاهی در تبدیل تصویر دوم پدید می‌آید). به همین دلیل، به جای بدست آوردن ضرایب، میتوان کل ماتریس را بدست آورد.

در واقع داریم:

$$\text{Car}_2A = \text{Car}_1$$

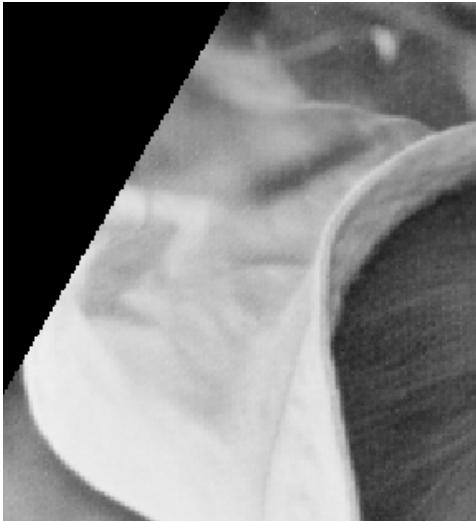
که A ماتریس انتقال است که باید آن را بدست بیاوریم. با استفاده از 6 نقطه، ماتریس A بدست می‌آید.

از طرفی می‌دانیم :

$$\text{Car}_1A^{-1} = \text{Car}2$$

با توجه به توضیحات بالا، حال برای ساختن عکس پاناروما، عملیات زیر را انجام می‌دهیم.

۱. نقطه پراکنده مشترک و نظیر به نظری در دو عکس پیدا می‌کنیم.
۲. با استفاده از نقاط، ماتریس A را بدست می‌آوریم.
۳. معکوس ماتریس A را بدست می‌آوریم.
۴. نقاط انتهایی عکس دوم را در ماتریس A ضرب می‌کنیم تا سایز تصویر پاناروما بدست بیاید.



عکس ۱.۲.۲: خط صاف گوشه تصویر که کنگره های نسبت به روش قبل صاف تر شده است.

در این سوال از ما خواسته شده است که تصویر Elain را با سه زاویه 80° و 30° و 45° درجه و با استفاده از روش های درونیابی چرخش بدھیم. در ضمن سوال گفته که این چرخش باید حول محور انجام شود.

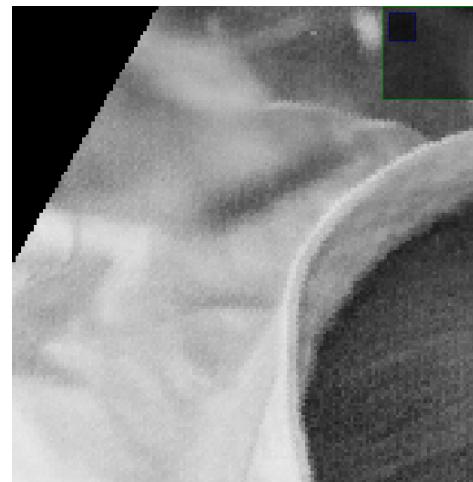
برای انجام این کار پس از بدست آوردن سایز عکس، باید نقاط عکس را طوری تبدیل کنیم که مرکز عکس موقعیت مکانی $(0,0)$ داشته باشد. برای این کار، همه نقاط را منتهای نصف سایز عکس می کنیم. در واقع نقاط عرضی، منتهای نصف عرض عکس و نقاط طولی منتهای نصف نقاط طول عکس.

پس از انجام این کار، ماتریس *affine* مرتبط با تبدیل را بدست می آوریم. حال به ازای هر پیکسل تصویر نهایی، با ضرب کردن آن در معکوس ماتریس *affine*، نظیر آن نقطه *nearest neighbor interpolation* عدد بدست آمده گرد می کنیم. و با توجه به اینکه هر نقطه را نصف سایز عکس در طول و عرض شیفت داده ایم، نقطه بدست آمده را نیز باید در طول و عرض به اندازه منفی آن شیفت بدھیم تا در بازه i عکس در اید. اگر روش *bilinear interpolation* بود، پس از بدست آمدن نظیر نقطه در عکس اصلی، با استفاده از توابع سقف و کف گیر، نزدیکترین عدد های طبیعی به عدد پیدا شده را بدست می آوریم. حال با جایگذاری عدد ها در معادله داده شده، ضرایب را بدست اورده و برای نقطه فعلی، شدت رنگش را بدست می آوریم. خروجی این بخش در قسمت نتایج آمده است.

سایز عکس چرخش داده شده، در اکثر مواقع بزرگتر از عکس اصلی است. برای حل این مشکل دو روش وجود دارد، یا سایز عکس جدید را بزرگتر در نظیر بگیریم، و یا از دست دادن بخشی از عکس را بپذیریم. ما از روش دوم استفاده می کنیم

$$yA^{-1} = x$$

در روش *nearest neighbor interpolation*، بعد از بدست آوردن حاصل عبارت بالا، با توجه که حاصل لزوماً یک عدد طبیعی نیست، عدههای بدست آمده برای موقعیت مکانی را به نزدیکترین عدد طبیعی تبدیل می کنیم. این روش ساده است، ولی باعث ایجاد برخی اثرات نامطلوب می شود. به طور مثال، همانطور که در این سوال از ما خواسته شده بود، در صورتی که بخواهیم با استفاده از این روش، عکسی را بچرخانیم، خطی که در عکس اصلی صاف است، پس از چرخاندن دارای کنگره هایی می شود. (عکس ۱.۲.۱)



عکس ۱.۲.۱: خط صاف کناره i تصویر که پس از چرخاندن کنگره کنگره شده است.

به همین دلیل، روش قوی تری پیشنهاد شده است. در روش *bilinear interpolation* که روش قوی تری است، به جای استفاده از نزدیکترین نقطه بدست آمده، بسته به موقعیت تصویر، از میانگین وزن دار چهار نقطه نزدیک استفاده می کنیم. فرض کنیم که (x,y) موقعیت مکانی یک نقطه در تصویر مقصد باشد که میخواهیم به آن مقداری نسبت دهیم. و فرض کنیم که $V(x,y)$ نیز شدت رنگ پیکسل با موقعیت مکانی ذکر شده باشد. برای $v_{bi\ i\ near}$ با استفاده از معادله زیر یک مقدار برای پیکسل دلخواه بدست بیاوریم:

$$v(x,y) = ax + by + cxy + d$$

که چهار ضریب a,b,c,d از روی چهار نقطه نزدیکترین نقطه دلخواه که مقدار انها را می دانیم، بدست می آیند. این روش نتیجه خیلی بهتری از روش *nearest neighbor interpolation* می دهد. البته از نظر محاسباتی نیز، از روش قبلی سنگین تر است. (عکس ۱.۲.۲ که نشان می دهد کنگره های لبه تصویر تا حدی بهتر شده است)

روش دیگر نیز برای downsample کردن، این است که به ازای هر مربع 2×2 ، یک پیکسل (مثلاً پیکسل بالا سمت چپ) را به عنوان نماینده قرار دهیم. به بیان دیگر، سطرها را یکی در میان و ستونها را نیز یکی در میان حذف کنیم. در ادامه سوال نیز از ما خواسته شده است که تصویر کوچک bilinear را با دو روش pixel replication و interpolation، بزرگ نمایی کرده و به سایز اصلی برگردانیم.

در روش pixel replication، به ازای هر مربع 2×2 در عکس جدید، یک مقدار در عکس اصلی داریم و همه‌ی مقادیر آن مربع را با همان مقدار جایگزین می‌کنیم. روش bilinear interpolation قبليتر در قسمت ۱.۱.۳ توضیح داده شده است. در این روش برای هر پیکسل عکس جدید، میانگین وزن داری از نزدیکترین پیکسل‌های اطرافش را می‌گذاریم. نتیجه در قسمت نتایج به تفصیل توضیح داده شده است.

زیرا عموماً قسمت‌هایی که خارج از کادر می‌افتد شامل اطلاعات مهمی نمی‌باشد.

قسمت ۱.۲.۱:

در این سوال از ما خواسته شده است که تصویر را با یکبار بدون نرمال کردن هیستوگرام و یکبار با نرمال کردن آن، به level gray ها مختلف ببریم.

(Histogram Equalization) نرمال کردن هیستوگرام روشی است که در آن contrast تصویر را با دستکاری هیستوگرام آن بهبود می‌دهیم. این روش عموماً کنتراست کلی تصویر را افزایش می‌دهد. خصوصاً وقتی که رنگ‌های عکس خیلی به هم نزدیک باشند. در این روش، توزیع نرمال و بهتری از رنگ‌ها ($0 - 255$) خواهیم داشت.

در ادامه، سوال از ما خواسته است که تصویر را به level gray های $8, 16, 32, 64$ ببریم. برای انجام این کار کافی است که عکس را تقسیم بر عدد مورد نظر کرده و کف عدد بدست آمده را در عدد مورد نظر ضرب کنیم. با استفاده از این روش (اگر عدد دلخواه را X فرض کنیم)، تمام اعداد بین $(CX, CX + X)$ به عدد CX تبدیل می‌شوند. زیرا حاصل تقسیم مورد نظر برابر C است (پس از گرفتن کف). که پس از ضرب کردن برابر CX خواهد شد. با استفاده از روشی که در بالا توضیح داده شد، تصویر اولیه و تصویر نرمال شده (با استفاده از histogram quantize) را equalization می‌کنیم.

برای بررسی نتیجه نیز خواسته شده ازتابع mean square error استفاده کنیم. این تابع دو ماتریس (در اینجا عکس) را مقایسه می‌کند. به ازای هر خانه از ماتریس، مربع تفاوت دو ماتریس را بدست می‌آوردم. و میانگین این تفاصل را به عنوان خروجی بر می‌گرداند. بنابراین هر چه عدد بزرگتر باشد، نشانه ای تفاوت بیشتر دو ماتریس است. نتیجه و بررسی نتیجه این بخش در قسمت نتایج به تفصیل بیان شده است.

قسمت ۱.۲.۲:

در سوال از ما خواسته شده است که یک تصویر را ابتدا یک بار با استفاده از فیلتر میانگین گیر، و بار دیگر بدون استفاده از آن down sample کنیم. (بانرخ ۲) یعنی طول و عرض عکس داده شده نصف می‌شود و تصویر نهایی، $\frac{4}{1}$ عکس نهایی است.

در downsample کردن با استفاده از فیلتر میانگین گیر، به ازای هر مربع 2×2 در تصویر اصلی، میانگین این ۴ پیکسل را در تصویر خروجی قرار می‌دهیم.

نتایج :

۱.۱.۲ قسمت

همانطور که در قسمت شرح توضیح داده شد، یکی قسمت کلیدی در پیدا کردن نقاطی نظری هم در دو تصویر است. اگر نقاط پیدا شده در یک محدود باشند، تصویر تطبیق داده شده می تواند خیلی نادقيق باشد. در شکل زیر دو نمونه از نقاط و خروجی (عکس register شده) نشان داده شده است.



شکل ۲.۲.۲ عکس تطبیق داده شده با استفاده از نقاط انتخاب شده - ۲-

شکل ۲.۳ نیز خروجی پاناروما عکس است.



خروجی پاناروما

Car	549	379	379	559	302	584
1	389	880	817	459	753	501
Car	398	399	406	479	523	321
2	398	457	122	136	162	357

جدول ۲.۲.۱ نقاط انتخاب شده - ۱



شکل ۲.۲.۱ عکس تطبیق داده شده با استفاده از نقاط انتخاب شده - ۱

Car	389	399	389	344	475	459
1	545	803	941	456	940	560
Car	406	418	409	357	490	540
2	122	383	513	25	511	459

جدول ۲.۲.۲ جدول نقاط انتخاب شده - ۲

قسمت ۱.۱.۳:

شکل ۲.۳.۱ و ۲.۳.۲ دوران ۳۰ درجه عکس را با استفاده از bilinear و nearest neighbor interpolation و نزدیک ترین همسایه interpolation نشان می‌دهد.



شکل ۲.۳.۳: دوران ۴۵ درجه عکس با استفاده از روش درون یابی نزدیک ترین همسایه



شکل ۲.۳.۱: دوران ۳۰ درجه عکس با استفاده از روش درون یابی نزدیک ترین همسایه



شکل ۳.۴: دوران ۴۵ درجه عکس با استفاده از روش درونیابی bilinear
شکل ۲.۳.۶ و ۲.۳.۵ نیز دوران ۸۰ درجه ای تصویر را نشان می‌دهد.



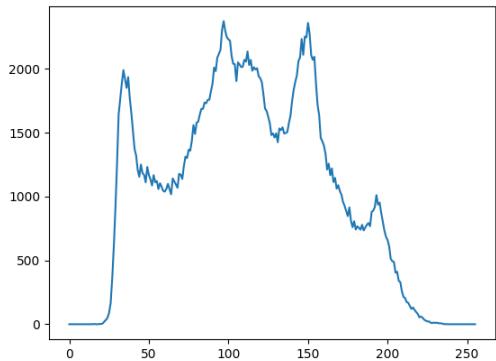
عکس ۲.۳.۲: دوران ۳۰ درجه عکس با استفاده از روش درونیابی bilinear
شکل ۲.۳.۳ و ۲.۳.۴ دوران ۴۵ درجه را با روش های ذکر شده نشان می‌دهد.

قسمت ۱.۲.۱:

شکل ۲.۴.۱ نشان دهندهٔ تصویر ورودی (بعد از سیاه-سفید کردن عکس) و شکل ۲.۴.۲ نیز هیستوگرام آن را نشان می‌دهد.



شکل ۲.۴.۱: عکس اصلی (سیاه-سفید)



شکل ۲.۴.۲: هیستوگرام عکس اصلی (سیاه-سفید)

عکس ۲.۴.۳ نشان دهندهٔ عکسی است که در قسمت شرح توضیح دادیم، نرمال کردن هیستوگرام باعث افزایش کنترast است می شود. و باعث افزایش کنترast است که ناحیه هایی که کنترast محلی آنها کمتر است، می شود.

همانطور که در مقایسه دو عکس مشخص می شود، صورت فرد سفیدتر و موها مشکی تر شده است. و راه های شلوار در عکس ۲.۴.۳ واضح تر دیده می شود.



شکل ۲.۳.۵: دوران ۸۰ درجه عکس با استفاده از روش درون یابی نزدیکترین همسایه

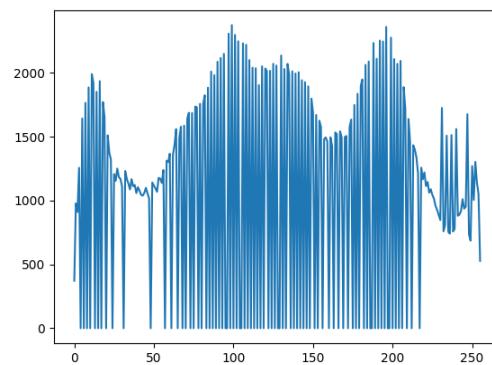


شکل ۲.۳.۶: دوران ۸۰ درجه عکس با استفاده از روش درونیابی bilinear

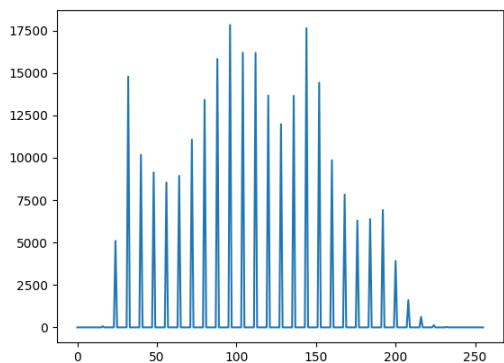


شکل ۲.۴.۳ تصویر بعد از histogram equalized

عکس ۲.۴.۴ ، هیستوگرام عکس ۲.۴.۳ را نشان می دهد. در واقع این هیستوگرام، equalized شده همان هیستوگرام شکل ۲.۴.۳ است. با مقایسه بین دو هیستوگرام، به این نتیجه میرسیم که توزیع هیستوگرام ۲.۴.۴ uniform تر بوده و سعی شده که همه level ها چگالی نزدیک به هم داشته باشند



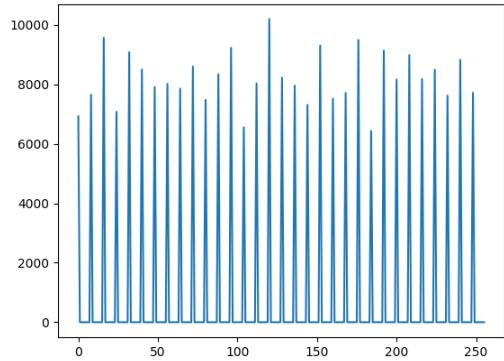
شکل ۲.۴.۴ هیستوگرام عکس ۲.۴.۳



شکل ۲.۴.۷ هیستوگرام عکس اصلی با \wedge gray level



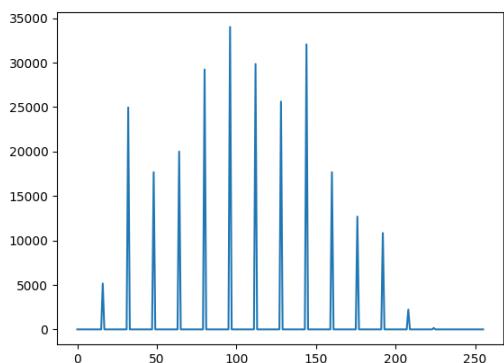
شکل ۲.۴.۵ عکس اصلی با \wedge gray level



شکل ۲.۴.۸ هیستوگرام عکس نرمال شده با \wedge gray level



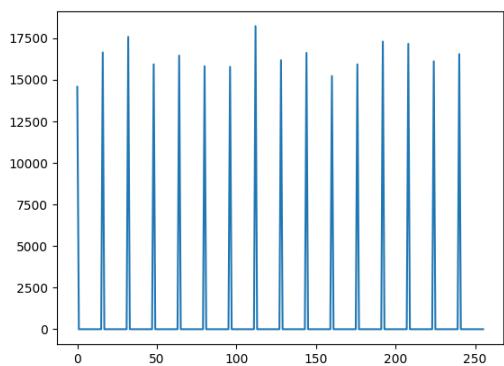
شکل ۲.۴.۶ تصویر نرمال شده با \wedge gray level



شكل ٢.٤.١١ هیستوگرام عکس اصلی با ١٦ gray level



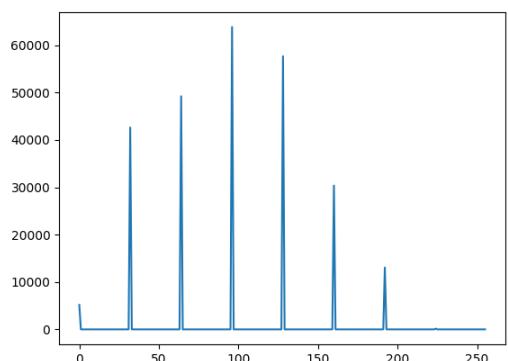
شكل ٢.٤.٩ عکس اصلی با ١٦ gray level



شكل ٢.٤.١٢ هیستوگرام عکس نرمال شده با ١٦ gray level



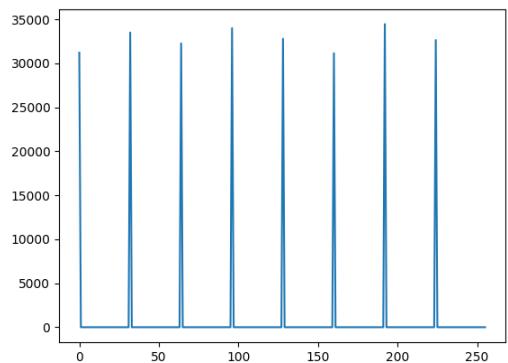
شكل ٢.٤.١٠ عکس نرمال شده با ١٦ gray level



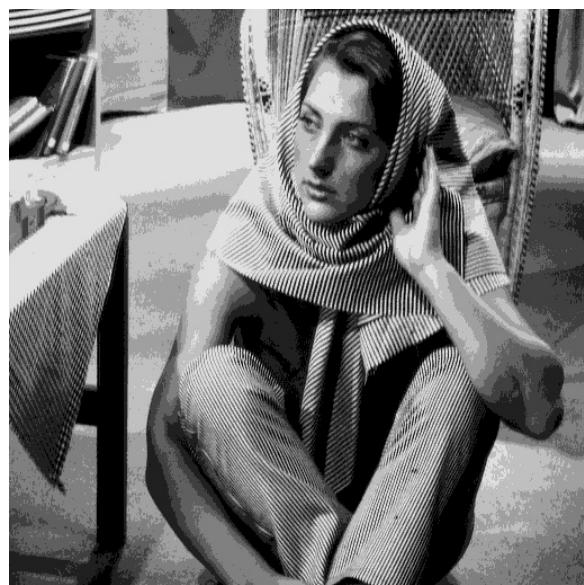
عکس ۲.۴.۱۵ هیستوگرام عکس اصلی با gray level



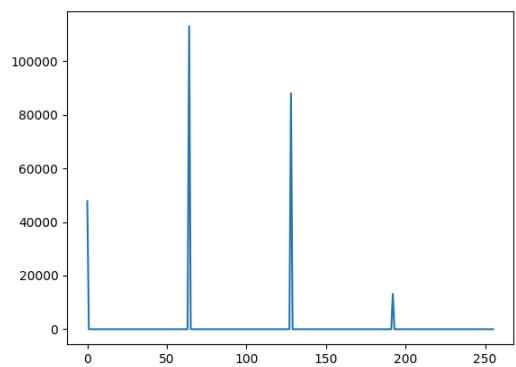
عکس ۲.۴.۱۳ عکس اصلی با gray level



عکس ۲.۴.۱۶ هیستوگرام عکس نرمال شده با gray level



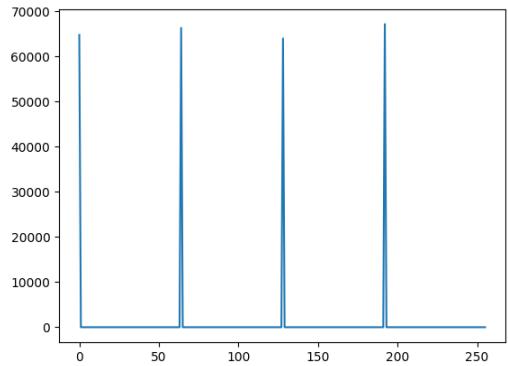
عکس ۲.۴.۱۴ عکس نرمال شده با gray level



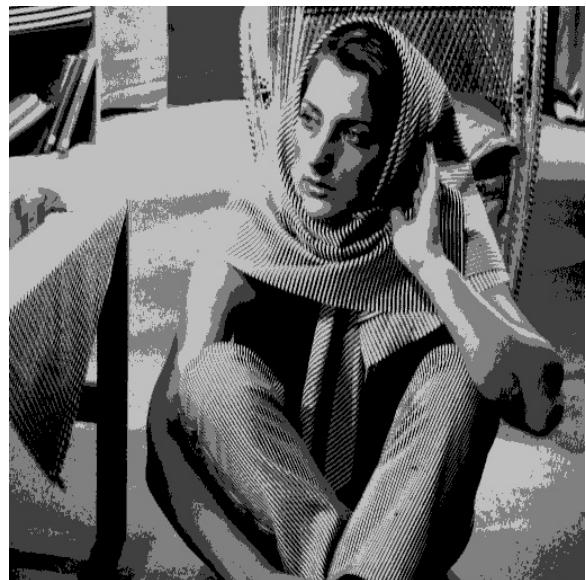
شکل ۲.۴.۱۹ هیستوگرام عکس اصلی با 64 gray level



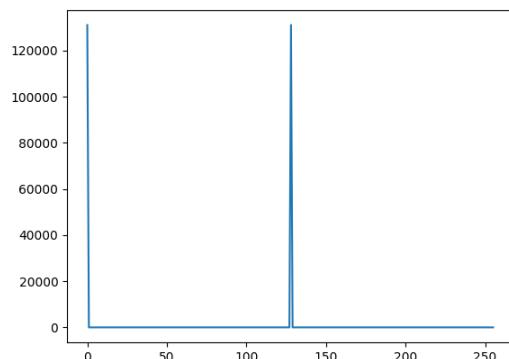
شکل ۲.۴.۱۷ عکس اصلی با 64 gray level



شکل ۲.۴.۲۰: هیستوگرام عکس نرمال شده با 64 gray level



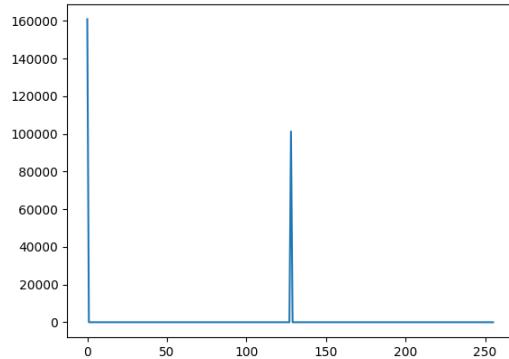
شکل ۲.۴.۱۸ عکس نرمال شده با 64 gray level



شکل ۲.۴.۲۳: هیستوگرام عکس اصلی با ۶۴ gray level



شکل ۲.۴.۲۱: عکس اصلی با ۱۲۸ gray level



شکل ۲.۴.۲۴: هیستوگرام عکس نرمال شده با ۶۴ gray level

همانطور که در تصاویر دیده می شود، با افزایش gray level روشنابی های بیشتری از عکس به یک عدد map می شوند، و یعنی با افزایش gray level ها تعداد رنگ های به کار رفته در عکس کمتر شده و عکس به سمت سیاه و سفید می روند. با مقایسه هیستوگرام عکس اصلی و عکس نرمال شده، توزیع uniform هیستوگرام عکس نرمال شده قابل مشاهده است.



شکل ۲.۴.۲۲: عکس نرمال شده با ۱۲۸ gray level

level	8	16	32	64	128
Without histeq	17.51	71.14	91.6	107.5	108.23
with histeq	106.1	107.9	109.9	110.2	110.9

جدول ۲.۴.۱: mean square error برای حالات مختلف عکس و عکس اصلی



عکس ۲.۵.۲ تصویر down sample شده



عکس ۲.۵.۳ تصویر up sample شده با روش bilinear interpolation (با فیلتر) تصویر ۲.۵.۱

همانطور که در شرح توضیح داده شده، mean square error یک مقایسه برای مقایسه نزدیکی دو ماتریس است. هر چه این مقدار عدد بزرگتری باشد، به معنای آن است که دو عکس (دو ماتریس) تفاوت بیشتری دارند.

همانطور که قابل انتظار بود، با افزایش gray level تفاوت عکس بدست آمده و عکس اصلی بیشتر می‌شود. زیرا همانطور که در شرح توضیح داده شد، اعداد بازه بزرگتری به عدد اول بازه map می‌شوند.

در جدول ۲.۴.۱ مشاهده می‌کنیم یک gap بزرگ بین level ۸ و level ۱۶ است. شکل ۲.۴.۵ و شکل ۲.۴.۱ نیز شباهت بالایی دارند، به طوری که شاید قابل تشخیص نباشند. میزان رشد در آخر نمودار به شدت کم شده است. به طور کلی میزان افزایش mse، نزولی است.

اعداد گزارش شده برای تصویر نرم‌الشدن، به طور کل بیشتر است.

قسمت ۱.۲.۲



عکس ۲.۵.۱ تصویر down sample شده با استفاده از فیلتر میانگین گیر



عکس ۲.۵.۷ تصویر up sample شده با روش pixel replication تصویر ۲.۵.۲(بدون فیلتر)



عکس ۲.۵.۴ تصویر up sample شده با روش bilinear interpolation تصویر ۲.۵.۲(بدون فیلتر)

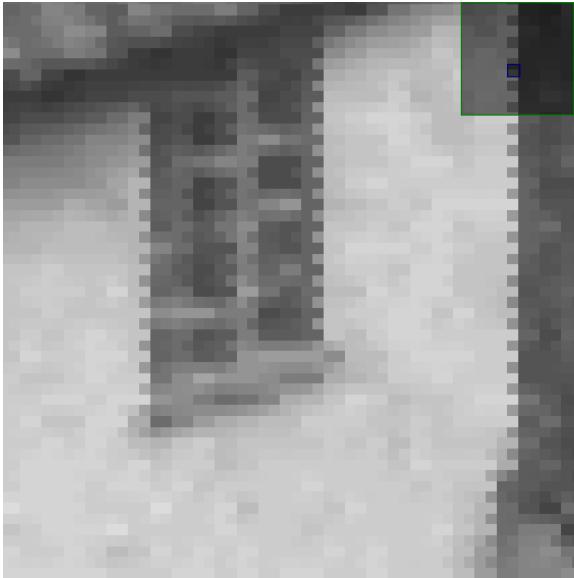
دو شکل پایین(۲.۵.۸ و ۲.۵.۹) تصویر زوم شده‌ی شکل‌های ۲.۵.۱ و ۲.۵.۲ هستند. همانطور که مشاهده می‌شود، خروجی با فیلتر کمی نرم تر است و پیکسل‌ها با سرعت کمتری تغییر می‌کنند.



شکل ۲.۵.۸ تصویر زوم شده شکل ۲.۵.۱(با فیلتر)

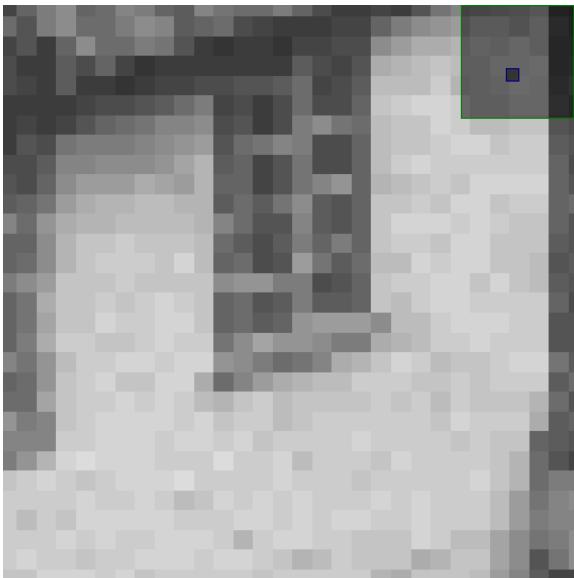


عکس ۲.۵.۶ تصویر up sample شده با روش pixel replication تصویر ۲.۵.۱(با فیلتر)

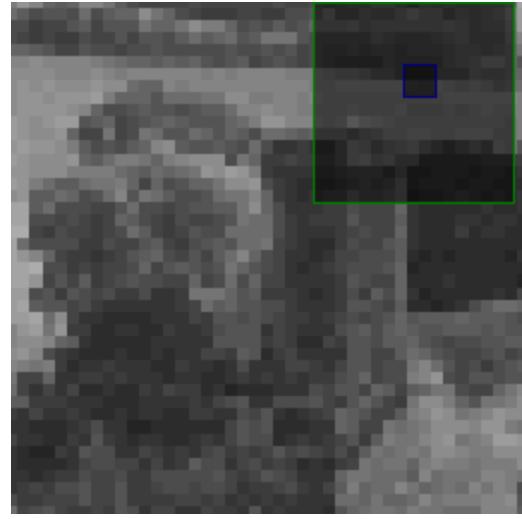


شکل ۲.۵.۱۰: تصویر زوم شده روش bilinear interpolation و Removing

اما در روش pixel replication این عیب دیده نمی شود و همانطور که مشاهده شده، mse کمتری نیز دارد.(شکل ۲.۵.۱۱)



شکل ۲.۵.۱۰: تصویر زوم شده روش pixel replication و Removing



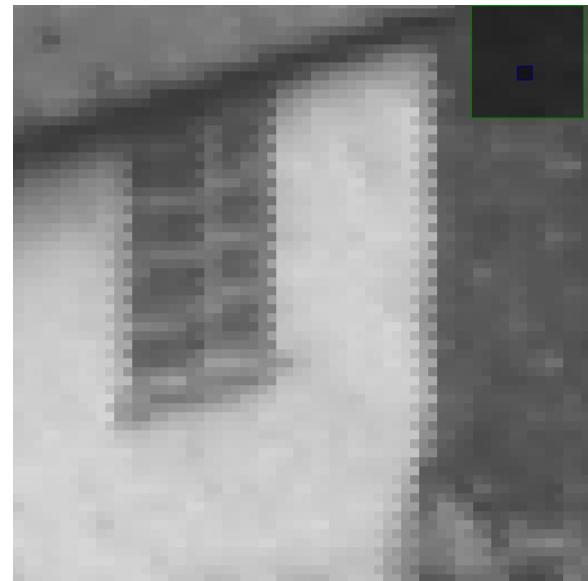
شکل ۲.۵.۹ تصویر زوم شده ای شکل ۲.۵.۲(بدون فیلتر)

در ادامه گزارشی از mse ها آورده و با عکس زوم شده، تفاوت خروجی ها را برس می کنیم:

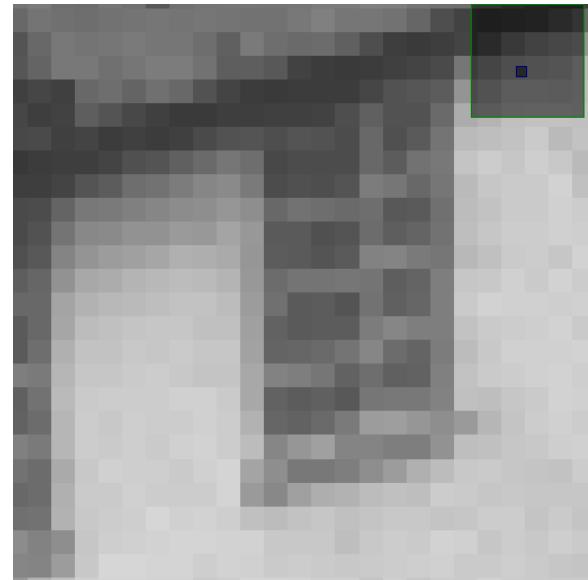
	Pixel Replication	Bilinear Interpolation
Removing	23.08	28.54
Averaging	35.25	39.5

جدول ۲.۵.۱ mean square error برای حالت های مختلف همانطور که در جدول گزارش شده است، کمترین میزان خطأ مربوط به removing columns and row برای روش upsampling و pixel replication و downsampling و upsampling است.

در اینجا روش bilinear بدتر عمل میکند، زیر مثلا برای یک خط سفید که کنار خط سیاه قرار دارد، یک میانگین وزن دار از این مقدار قرار می دهد که باعث کنگره شدن تصویر می شود.(شکل ۲.۵.۱۰)



شکل ۲.۵.۱۱: تصویر زوم شده روش bilinear interpolation و Averaging



شکل ۲.۵.۱۲: تصویر زوم شده روش pixel replication و Averaging

از مقایسه عکس های ۲.۵.۱۱ و ۲.۵.۱۲ با عکس های ۲.۵.۹ و ۲.۵.۱۰ ، متوجه می شویم در عکس هایی که با استفاده از روش averaging تغییر سایز داده اند، تاری و ماتی بیشتر دیده می شود. این ویژگی خاصیت فیلتر میانگین گیر است. در این روش پیکسل های مجاور، مقادیر نزدیک تری به هم دارند.

```

cv.imshow("registered" , reg_img_2)

for i in range(0,height):
    for j in range(0,width):
        reg_img_2[i - min_h][j] = img1[i][j]

cv.imwrite("result/1.1.2/panorama_2.jpg" , reg_img_2)

```

:ۯ.۱.۱

```

import cv2 as cv
import numpy as np
import math

theta = int(input("Enter the degree"))
theta = (np.pi / (180 / theta))
path = "image/Elaine.jpg"
img = cv.imread(path ,
cv.IMREAD_GRAYSCALE)
width , height = img.shape

# cv.imshow("image" , img)
# cv.waitKey(0)

trans = np.array([[np.cos(theta) ,
np.sin(theta)],[ -np.sin(theta) ,
np.cos(theta)]])
inv_trans = np.linalg.inv(trans)

res = np.zeros((height, width) ,
np.uint8)
mode = input("Enter mode: \n 1 for
nearest neighbour interpolation \n 2
for bilinear interpolation")
half = int(width / 2)
for r in range(int(-(width/2)) ,
int((width/2))):
    for c in range(-int(height / 2)
, int(height/2)):

        x , y =
np.dot(np.array([r,c]) , inv_trans)

        if(mode == "1"):


```

Appendix:

:ۯ.۱.۱

```

import cv2 as cv
import numpy as np

img1 = cv.imread("image/Car1.jpg")
img2 = cv.imread("image/Car2.jpg")
height , width , channel = img1.shape

roi_1 = np.array([[389, 545], [399, 802], [389, 941], [344,
456], [475, 940], [459, 560]])
roi_2 = np.array([[406, 122], [418, 383],[409, 513], [357,
25], [490, 511], [479, 136]])

A = np.empty((0,6), np.uint8)
for x,y in roi_2:
    A = np.vstack((A , np.array([x ,y ,x*y,x*x,y*y,1])))

B = np.empty((0,6) , np.uint8)

for x,y in roi_1:
    B = np.vstack((B , np.array([x ,y ,x*y,x*x,y*y,1])))

trans = np.linalg.solve(A , B)
inv_trans = np.linalg.inv(trans)
left   =   np.array(np.dot(np.array([0,0,0,0,0,1])   ,
trans),np.int64)
right = np.array(np.dot(np.array([height - 1,0,0,(height -
1) ** 2 ,0,1]) , trans),np.int64)
up = np.array(np.dot(np.array([0,width - 1 ,0,0,(width -
1) ** 2 ,1]) , trans),np.int64)
down = np.array(np.dot(np.array([height - 1,width -
1,(width - 1)*(height - 1),(height - 1)**2,(width - 1) ** 2,
1]) , trans),np.int64)

min_h = min(left[0] , right[0] , up[0] , down[0])
max_h = max(left[0] , right[0] , up[0] , down[0])
min_w = min(left[1] , right[1] , up[1] , down[1])
max_w = max(left[1] , right[1] , up[1] , down[1])

reg_img_2 = np.zeros((1200 ,1540 , 3) , np.uint8)

for i in range(min_h,max_h):
    for j in range(min_w, max_w):
        pixel = np.array([i , j , i*j , i*i , j*j , 1], np.int64)
        target = np.dot(pixel, inv_trans)
        x , y = int(np.floor(target[0])) , int(np.floor(target[1]))
        if x >= height or x < 0 or y < 0 or y >= width:
            continue
        reg_img_2[i - min_h][j] = img2[x][y]

cv.imwrite("result/1.1.2/registered.jpg" , reg_img_2)

```

```

        b =
np.array([[img[fx + half][fy + half]] , [img[fx + half][cy + half]] , [img[cx + half][fy + half]] , [img[cx + half][cy + half]]])

        if
np.linalg.matrix_rank(A) != A.shape[0]:
                                value =
img[fx + half][fy + half]
                else:
                                coef =
np.linalg.solve(A , b)
                                value =
np.dot(np.array([r+half , c+ half , (r+half)*(c+half),1]) , coef)
                                res[int(r +
width / 2)][int(c + height / 2)] =
int(value)

name = "result/1.1.3/rotate_" + ("NNI_"
if mode == "1" else "BLI_") +
str((theta * 180) / np.pi) + ".jpg"
cv.imwrite(name , res)
cv.imshow("rotated" , res)
cv.waitKey(0)

```

:1.7.1

```

import cv2 as cv
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
import numpy as np

path = "image/Barbara.jpg"
img = cv.imread(path ,
cv.IMREAD_GRAYSCALE)
cv.imwrite("result/1.2.1/balck_white.jp
g" , img)
# cv.imshow("barbara" , img)
# cv.waitKey(0)

width , height = img.shape
hist = cv.calcHist([img] , [0] , None ,
[256] , [0,256])
plt.plot(hist)

```

```

        x , y =
int(np.round(x) + (width / 2)) ,
int(np.round(y) + (height / 2))
                                if x < 0 or y <
0 or x >= width or y >= height:
                                continue
                                res[int(r +
width / 2)][int(c + height / 2)] =
img[x][y]

                if(mode == "2"):
                                fx , cx =
int(np.floor(x)) , int(np.ceil(x))
                                fy , cy =
int(np.floor(y)) , int(np.ceil(y))

                                up =
np.dot(np.array([fx,fy]) ,trans) + half
                                right =
np.dot(np.array([cx,fy]) ,trans )+ half
                                down =
np.dot(np.array([fx,cy]) , trans)+ half
                                left =
np.dot(np.array([cx,cy]) , trans)+ half

                A =
np.array([[up[0] , up[1] , up[1] *
up[0] , 1],
[down[0] , down[1] , down[1] * down[0]
, 1] ,
[right[0] , right[1] , right[1] *
right[0] , 1],
[left[0] , left[1] , left[1] * left[0]
, 1]])
                                d = np.array(A
, np.uint8)

                                if cx + half >=
width or cy + half >= height or fx +
half < 0 or fy + half < 0:
                                continue

```

```

    new_img =
np.uint8((np.floor(eqlzd_img / level))
* level)
    name = "result/1.2.1/gray_level_" +
str(level) + "_histeq.jpg"
    cv.imwrite(name , new_img)
    level_hist = cv.calcHist([new_img]
, [0] , None , [256] , [0,256])
    plt.figure(str(level))
    plt.plot(level_hist)
    name = "result/1.2.1/gray_level_" +
str(level) + "_hist_histeq.png"
    plt.savefig(name)
    result[1].append(np.square(new_img
- img).mean())
    print(np.square(new_img -
img).mean())

print(result)

```

:1.2.1

```

import cv2 as cv
import numpy as np

path = "image/Goldhill.bmp"
img = cv.imread(path ,
cv.IMREAD_GRAYSCALE)
width, height = img.shape

# cv.imshow("Goldhill" , img)
# cv.waitKey(0)

# down sample using averging filter
shrgked_img = np.zeros((int(width / 2)
, int(height / 2)) , np.uint8)
for x in range(0, int(width / 2)):
    for y in range(0, int(height /
2)):
        shrgked_img[x][y] = (int(img[2
* x][2 * y]) + int(img[2 * x + 1][2 *
y]) + int(img[2 * x][2 * y + 1]) +
int(img[2 * x + 1][2 * y + 1])) / 4

name =
"result/1.2.2/downSample_Avg_Filer.jpg"
cv.imwrite(name , shrgked_img)

```

```

#
plt.savefig("result/1.2.1/histogram.png"
")
# plt.hist(img.ravel() , 256 , [0,256])

eqlzd_img = cv.equalizeHist(img)
eqlzd_hist = cv.calcHist([eqlzd_img] ,
[0] , None , [256] , [0,256])

plt.plot(eqlzd_hist)
#
plt.savefig("result/1.2.1/eqlzd_histogram.png")

name =
"result/1.2.1/equalized_image.jpg"
cv.imwrite(name , eqlzd_img)

# cv.imshow("equal" , eqlzd_img)
# cv.waitKey(0)

gray_level = np.array([8 , 16 , 32 , 64
, 128])
result = [[],[]]

# for normal image
for level in gray_level:
    new_img = np.uint8((np.floor(img /
level)) * level)
    name = "result/1.2.1/gray_level_" +
str(level) + ".jpg"
    cv.imwrite(name , new_img)
    level_hist = cv.calcHist([new_img]
, [0] , None , [256] , [0,256])
    # plt.figure(str(level))
    # plt.plot(level_hist)
    # name = "result/1.2.1/gray_level_"
+ str(level) + "_hist.png"
    # plt.savefig(name)
    result[0].append(np.square(new_img
- img).mean())

# quantized equlized image
for level in gray_level:

```

```

fy = int(np.floor(c/2))
cy = int(np.ceil(c/2))
# print(r,c,fx,cx,fy,cy)
if r % 2 == 0:
    A = np.array([[2 * fx , 2 *
fy , 4*fx * fy , 1],[2 * cx + 2, 2 * fy
, 4 * (cx + 1) * fy , 1],[2 * fx ,2 *
cy ,4*fx*cy , 1],[2* cx + 2 ,2 * cy ,
4*(cx + 1)*cy, 1]])
    if c % 2 == 0:
        A = np.array([[2 * fx , 2 *
fy , 4*fx * fy , 1],[2 * cx, 2 * fy , 4
* cx * fy , 1],[2 * fx ,2 * cy + 2
,4*fx*(cy + 1) , 1],[2* cx + 2 ,2 * cy ,
4*cx*(cy + 1), 1]])
    else:
        A = np.array([[2 * fx , 2 *
fy , 4*fx * fy , 1],[2 * cx, 2 * fy , 4
* cx * fy , 1],[2 * fx ,2 * cy ,4*fx*cy
, 1],[2* cx ,2 * cy , 4*cx*cy, 1]])
    if cx >= width / 2 or cy >=
height / 2:
        bi_upsample_img[r][c] =
shrngked_1_img[fy][fx]
        continue
    b =
np.array([[shrngked_1_img[fx][fy]], [sh
rngked_1_img[cx][fy]], [shrngked_1_img[fx
][cy]], [shrngked_1_img[cx][cy]]])
    # print(A , b)
    if np.linalg.matrix_rank(A) !=
A.shape[0]:
        bi_upsample_img[r][c] =
shrngked_1_img[fx][fy]
        continue
    coef = np.linalg.solve(A , b)
    color = np.dot([[r,c,r*c,1]],
coef)
    bi_upsample_img[r][c] =
int(color)

name =
"result/1.2.2/upSamle_BLI_avg.jpg"
cv.imwrite(name , bi_upsample_img)
cv.imshow("GodhillBLI" ,
bi_upsample_img)
cv.waitKey(0)

```

```

# cv.imshow("GodhillAVG" ,
shrngked_img)
# cv.waitKey(0)

# down sample without averaging filter
shrngked_1_img = np.zeros((int(width /
2) , int(height / 2)) , np.uint8)
for x in range(0, int(width / 2)):
    for y in range(0, int(height /
2)):
        shrngked_1_img[x][y] =
(int)(img[2 * x][2 * y])

name = "result/1.2.2/DownSample.jpg"

cv.imwrite(name , shrngked_1_img)
# cv.imshow("GodhillADi" ,
shrngked_1_img)
# cv.waitKey(0)

# upsample image using pixel
replication
pr_upsample_img = np.zeros((width ,
height) , np.uint8)
for x in range(0 , width):
    for y in range(0 , height):
        pr_upsample_img[x][y] =
shrngked_1_img[int(np.floor(x /
2))][int(np.floor(y / 2))]

name =
"result/1.2.2/upSample_pxl_replc_avg.jp
g"
cv.imwrite(name , pr_upsample_img)
cv.imshow("GodhillPR" ,
pr_upsample_img)
cv.waitKey(0)

# upsample image using bilinear
interpolation

bi_upsample_img = np.zeros((width ,
height) , np.uint8)

for r in range(0,width):
    for c in range(0, height):
        fx = int(np.floor(r/2))
        cx = int(np.ceil(r/2))

```

```
result =  
np.array([[np.square(pr_upsample_img -  
img).mean()] ,  
  
[np.square(bi_upsample_img -  
img).mean()]])  
print(result)
```