

به نام او



پردازش تصویر

استاد : دکتر حامد آذرنوش

دانشجو : حمیدرضا ابوئی

شماره دانشجویی : ۹۷۳۳۰۰۲

آزمون پایانترم

فهرست مطالب

۲	۱	سوال اول
۲	۱.۱	توضیحات تکمیلی روند کد
۳	۲.۱	ورودی برنامه
۳	۳.۱	خروجی برنامه
۵	۲	سوال دوم
۵	۱.۲	توضیحات تکمیلی روند کد
۵	۲.۲	ورودی برنامه
۶	۳.۲	خروجی برنامه
۷	۳	سوال سوم
۷	۱.۳	توضیحات تکمیلی روند کد
۷	۲.۳	ورودی برنامه
۸	۳.۳	خروجی برنامه
۹	۴	سوال چهارم
۹	۱.۴	توضیحات تکمیلی روند کد
۱۰	۲.۴	ورودی برنامه
۱۲	۳.۴	خروجی برنامه

۱ سوال اول

۱.۱ توضیحات تکمیلی روند کد

در قسمت آ برای یافتن هیستوگرام و اطلاعات مربوط به نویز تصویر روشی که استفاده شده است، جدا کردن قسمتی از تصویر که اطلاعات مربوط به نویز را در خود جای دهد. بدین منظور با توجه به تصویر داده شده، می‌توانیم اطراف تصویر که به نظر مشککی با نویز است انتخاب کنیم و اطلاعات مربوط به نویز را استخراج کنیم. نکته ای که وجود دارد آن است که هرچه تعداد داده های ما بیشتر باشد فراوانی دقیق تر و بهتر به دست می‌آید بنابراین در کد، دو نوار سمت چپ و راست به قدری که وارد قسمت مجمله نشود انتخاب شده و اطلاعات آن استخراج شد.

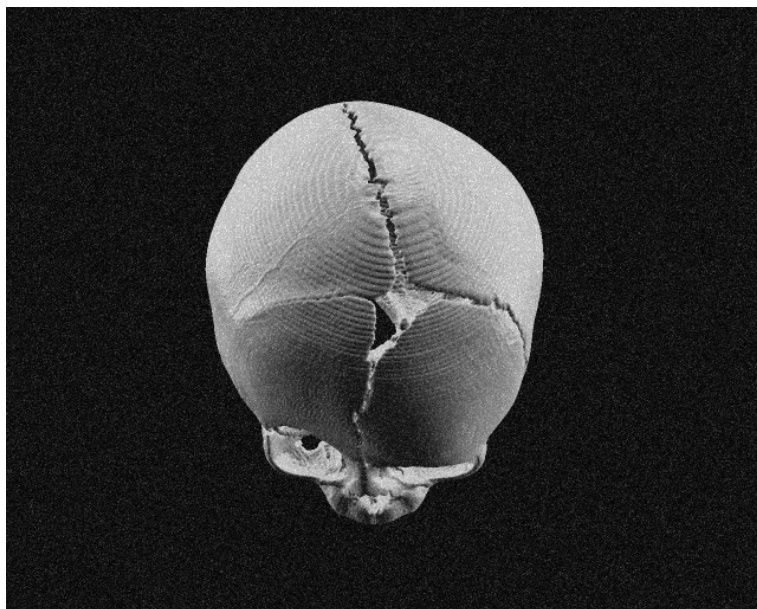
از روی شکل نویز به دست آمده، به نظر می‌رسد که جنس تابع چگالی احتمال از جنس Rayleigh باشد. در قسمت و خواسته شده که بهبود برای کاهش نویز بررسی شود. همانطور که در تصویر خروجی و هیستوگرام آن مشخص است، میزان پراکندگی نویز در این حالت بسیار کمتر است. بنابراین می‌توان گفت که در نقاط غیر لبه ای، اثر نویز کاهش یافته است. البته در مکان هایی که ما تغییر داریم مانند لبه ها، چون حفظ اطلاعات برای ما ارزشمند است تغییراتی که اعمال می‌کنیم کمتر است و در آن نقاط نویز تصویر ما تغییر محسوسی نمی‌کند. اما لبه ها حفظ می‌شوند.

روش میانه گیری با روش فیلتر تطبیقی یکسان عمل نمی‌کند زیرا روش میانه گیری کاری با محتوای درون کرنل ندارد و صرفاً به این که داخل هر کرنل مقدار میانه چه عددی است کار دارد. اما در روش فیلتر تطبیقی، محتوای داخل کرنل اهمیت می‌یابد. بدین صورت که میزان واریانس و میانگین نقاط داخل کرنل روی میزان کاهش نویز تاثیر دارد. بنابراین زمانی که به خروجی نگاه می‌کنیم در نقاطی که ما اطلاعات مهم داریم، در روش میانه گیری این اطلاعات از بین می‌روند ولی در روش فیلتر تطبیقی این اطلاعات مهم از بین نمی‌روند. اما صرفاً از دید کاهش نویز، فیلتر میانه بهتر عمل می‌کند اما هوشمندانه خیر.

البته لازم به ذکر است که هنگام پیاده سازی فرمول این فیلتر که برابر است با $f(x, y) = g(x, y) - \frac{\sigma_g^2}{\sigma_L^2} [g(x, y) - m_L]$ و اگر همینجوری فقط واریانس پنجره ها را در فرمول بگذاریم، به مشکلاتی بر می‌خوریم. برای مثال مشاهده می‌کنیم که مقادیر منفی در نتیجه آمده است. با بررسی مجدد متوجه می‌شویم که باید این واریانس ها و واریانس کلی را به روشی مناسب تر به دست آوریم؛ بدین صورت که ابتدا واریانس هر پنجره را به دست می‌آوریم. سپس σ_η^2 برابر است با میانگین واریانس های پنجره ها. سپس برای منفی نشدن مقادیر،

واریانس هایی که کمتر از واریانس η است را برابر با واریانس η می گذاریم و با این مقادیر فرمول بالا را پیاده سازی می کنیم

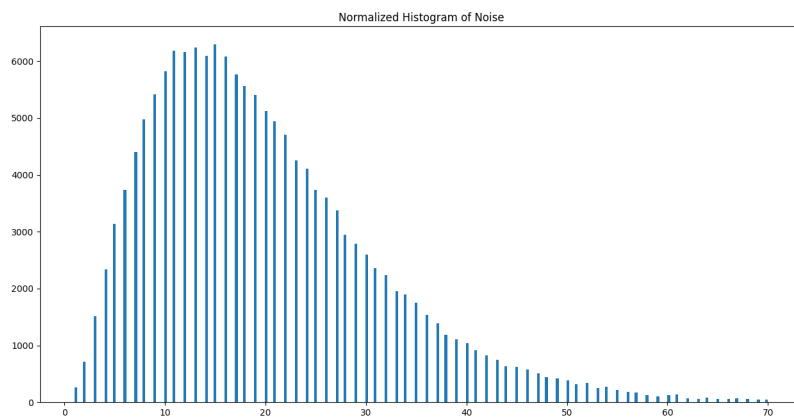
۲.۱ ورودی برنامه



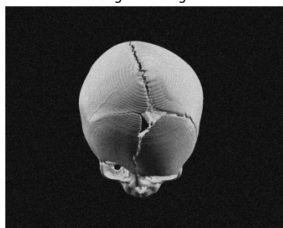
۳.۱ خروجی برنامه

Mean of noise is: 19.9309765625

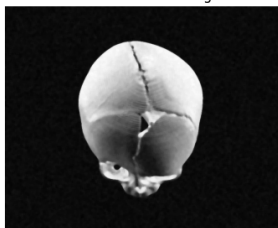
Variance of noise is: 129.10471493174234



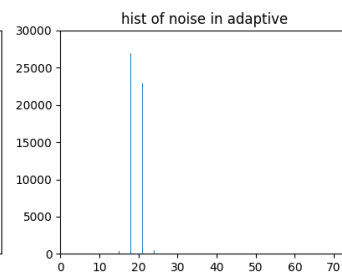
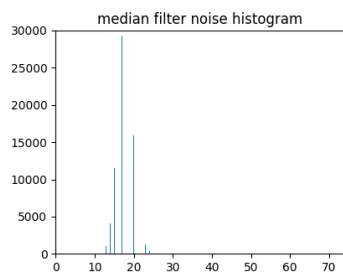
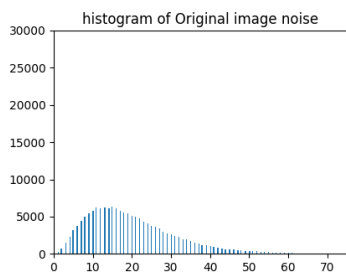
Original image



median filtered image



adaptive filter



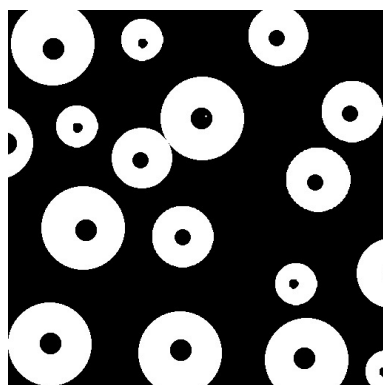
۲ سوال دوم

۱.۲ توضیحات تکمیلی روند کد

با بررسی بیشتر تصویر مشاهده میکنیم که با این که تصویر نسبتاً به صورت باینری است ، اما وجود برخی نویز های بسیار کوچک ممکن است کار ما را خراب کند بنابراین ابتدا با یک ترشهولد گیری آن را کاملاً باینری می‌کنیم.

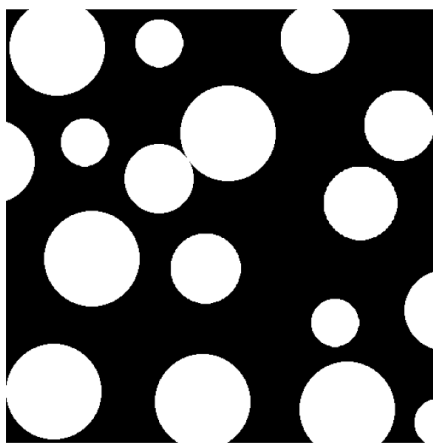
برای گرفتن موقعیت کلیک کاربر از `plt.canvas.mpl_connect` استفاده شده است. همچنین برای بررسی این که آیا دکمه ی اسکپ زده شده یا خیر نیز می‌توان از این استفاده کرد که کد آن در تابع `esc` مشخص است. هنگامی که تابع پر کردن حفره فراخوانی می‌شود و مختصات در ورودی داده می‌شود، به سراغ اعمال الگوریتم می‌رویم. در اینجا ما با دایلیت کردن با کرنل + ، همسایگی نقاط اولیه خود را می‌یابیم. سپس با بررسی این که آیا این نقاط داخل تصویر اصلی نیز سفید هستند یا خیر؛ متوجه می‌شویم که آیا این نقاط جدید را باید به ناحیه ی خود اضافه کنیم یا خیر. سپس مجدداً همین مسیر را پیش می‌رویم تا جایی که با دایلیت کردن و مقایسه ناحیه ی جدید، دیگر هیچ پیکسل جدیدی به ناحیه ی ما اضافه نشود. حال همان تصویر را که در این جا به صورت گلوبال مشاهده کردیم تغییر می‌یابد و پس از اتمام حلقه، نسخه ی جدید آن را مشاهده می‌کنیم.

۲.۲ ورودی برنامه



۳.۲ خروجی برنامه

با کلیک روی مرکز های دایره ها به راحتی تصویر زیر به دست می آید



۳ سوال سوم

۱.۳ توضیحات تکمیلی روند کد

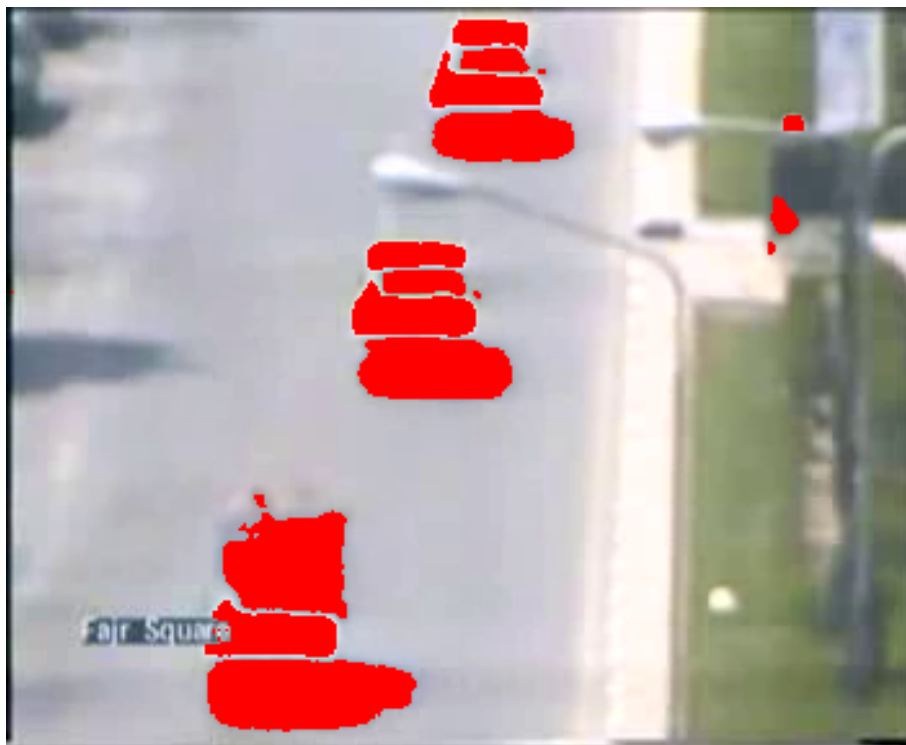
برای یافتن تصویر پس زمینه می توان از راه های مختلف استفاده کرد. یکی از راه ها استفاده از میانگین تصاویر است. راه دیگری که می توان پس زمینه را به دست آورد و در کد موجود است استفاده از میانه است بدین صورت که تمام ویدیو را در یک لیست می گذاریم و سپس در محور زمان از آن با استفاده از تابع مدین بکگراند را استخراج می کنیم. این روش بهتر از روش قبل کار می کند و نتیجه ی بهتری دارد زیرا در حالت اول تمام تحرک ها در بکگراند ما تاثیر می گذاشتند.

حال می توانیم مجددا فیلم را بازخوانی کنیم و در همین حین، بکگراند را از آن کم کنیم و جاهایی که تفاوت زیادی دارد. بدین معناست که یه شی در حال جابجایی است. حال برای این که اشیا در حال جابجایی را با رنگ قرمز مشخص کنیم ابتدا قسمت هایی که در حال تغییر است را صفر می کنیم و سپس در آن قسمت ها رنگ قرمز را اضافه می کنیم تا نتیجه نهایی به دست بیاید.

۲.۳ ورودی برنامه



۳.۳ خروجی برنامه



پس زمینه ی دیتکت شده به صورت زیر است :



۴ سوال چهارم

۱.۴ توضیحات تکمیلی روند کد

در این سوال برای یافتن پاسخ نهایی ابتدا با استفاده از همان روش `plt.canvas.mpl_connect` مختصات نقطه‌ی دانه را از کاربر می‌گیریم سپس وارد تابع می‌شویم. می‌خواهیم ناحیه‌ی یافته شده را در یک تصویر مشکی با ابعاد تصویر اصلی نشان دهیم. (black)

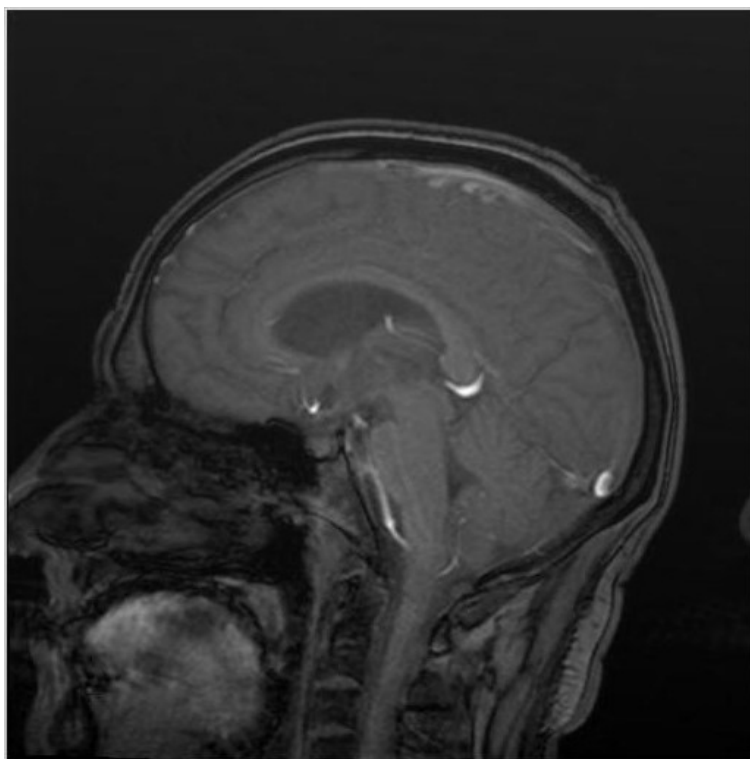
سپس در این سوال دایلیت می‌کنیم و هر سری بررسی می‌کنیم که آیا پیکسل‌های جدید که به دست آمده اند، داخل سگمنت قرار می‌گیرند یا خیر. این مقایسه می‌تواند به شکل‌های متفاوتی صورت گیرد. اگر ما هر سری که دایلیت می‌کنیم و همسایگی‌های ۴ یا ۸ تایی (برای مشخص کردن این که در حال استفاده از کدام هستیم کرنلمان را با توجه به متغیر `neighborhood_4` به دست می‌آوریم) را با پیکسل کناری که دایلیت شده است مقایسه کنیم، اتفاقی که رخ می‌دهد نشت کردن است. یعنی به مرور ممکن است سطح چیزی که مقایسه می‌کنیم تغییر محسوسی کند. بدین منظور می‌توانیم از راه‌های دیگری برای جلوگیری از این اتفاق استفاده کنیم. برای مثال کلا شدت پیکسل انتخاب شده توسط کاربر را ملاک در نظر بگیریم و یا این که میانگین شدت‌های پیکسل‌های داخل محدوده را به عنوان ملاک شباهت یابی در نظر بگیریم که خوبی این روش این است که محدوده‌ی انتخاب شده بسیار خوب و دقیق تر از حالت قبل انتخاب می‌شود. حال باید در پیکسل‌های جدید بگردیم و این `thresholding` را در مقایسه با ملاکی که بالاتر توضیح داده شد بگیریم و اگر قدر مطلق تفاوت‌ها کمتر از سطح داده شده (۱۱) باشد، پیکسل داخل محدوده است و جزو پیکسل‌های سگمنت شده محسوب می‌شود. این کار را می‌توان به این صورت انجام داد که در هر ایتريشن، یک بار کل پیکسل‌های تصویر را بخوانیم و اگر جزو پیکسل‌های جدید بود، شرط بودن در سگمنت را بررسی کنیم. اما این روش بهینه نیست و زمان زیادی طول می‌کشد مخصوصاً اگر سگمندی که قرار است انتخاب شود، بزرگ باشد و نیاز به ایتريشن زیادی داشته باشد (همانطور که در سوال ۲ نیز مطرح شد، هنگامی که تعداد پیکسل‌های سگمنت در یک ایتريشن تغییر نکند متوجه می‌شویم که کار سگمنتیشن به پایان رسیده است)

حال الگوریتمی که می‌توان استفاده کرد برای این که سرعت را بهبود دهیم این است که از برخی روش‌ها و توابع پایه و معمولی نامپای که در تمام کدها استفاده می‌کردیم، استفاده کنیم و تغییر خیلی کوچکی در بررسی شرط بودن در سگمنت انجام دهیم که در ادامه به توضیح آن می‌پردازیم.

همانطور که در بالا گفتیم ابتدا شاخص که همان میانگین پیکسل‌های داخل سگمنت هست را در نظر

میگیریم. سپس پیکسل هایی که قرار است چک شوند (دایلیت شده ی سگمنت منهای خود سگمنت برای به دست آوردن پیکسل های جدید) را ابتدا در تصویر می یابیم و آن را از شاخص کم میکنیم. بدین ترتیب برای مثال اگر شاخص ما ۶۰ باشد، تصویر ما از ۰ تا ۲۵۵ به ۶۰ تا ۱۹۵ تغییر می یابد. حال از این تصویر قدر مطلق میگیریم. بدین ترتیب تصویر ما به ۰ تا ۱۹۵ تبدیل می شود که مقادیر منفی مثبت شده اند. حال ترشهولدینگ را با قطع از سطح ترشهولد داده شده (۱۱) انجام می دهیم. حال این ترشهولد شده را ابتدا معکوس می کنیم بدین صورت که مقادیر ۰ تا سطح ترشهولد (۱۱) ماکزیمم و مقادیر بیشتر، صفر شوند. حال این ترشهولد گرفته شده را با نقاط جدید and می کنیم و نقاطی که به دست می آید، پیکسل های جدید صدق شده در شرط است و حال این نقاط را به نقاط سگمنت شده اضافه می کنیم. همانطور که مشاهده می شود در این روش ما تک تک نقاط تصویر را یک به یک بررسی نکردیم بلکه برای تصویرمان یک الگوریتم یافتیم که سرعت را چندین برابر زیاد می کند. همچنین در کد یک پرچم تعریف شده است که عملیات فقط بار اول که تصویر داریم اجرا شود و اگر دفعات بعد کلیک کردیم اتفاقی نیافتد

۲.۴ ورودی برنامه



۳.۴ خروجی برنامه

segmented by region based growing with cross 3x3 kernel



segmented by region based growing with rectangle 3x3 kernel

