گزارش تمرین سری چهارم

سیستم‌های چندرسانه‌ای

رضا اکبریان بافقی - ۹۵۱۰۰۰۶۱

۱ آشنایی با دستورات متلب

۱,۱

نام دستور	عملکرد/ مثال
videoinput	یک آبجکت از نوع video input می‌سازد که adaptorname نام آداپتوری که برای ارتباط برقرار کردن بین سیستم و متلب استفاده می‌شود. این دستور می‌تواند مقدار deviceID، format و P1 V۱ تا Pn Vn باشد.
	videoinput(adaptorname,deviceID,format,P1,V1,...) videoinput('winvideo', 1) با استفاده از winvideo یک آبجکت برای video input با آیدی ۱ ساختیم.
videoreader	با استفاده از این تابع می‌توان فایل filename ویدئویی را می‌سازیم و با استفاده از جفت name و value می‌توانیم اگر بخواهیم عملیات‌های مختلفی را انجام بدهیم. آبجکتی که ساخته می‌شود از نوع videoreader می‌باشد.
	VideoReader(filename,Name,Value) VideoReader('myfile.mp4','CurrentTime',1.2) فایل myfile.mp4 را از ثانیه ۱,۲ به بعد می‌خوانیم.
videowriter	با استفاده از این دستور می‌توانیم که ویدئو را با اسم filename ذخیره کنیم. هم‌چنین می‌توانیم فرمت فایل را هم مشخص کنیم.
	VideoWriter(filename,profile) VideoWriter('myfile.avi', 'Grayscale AVI') در اینجا ویدئو با اسم myfile.mp4 و به‌صورت یک فایل فشرده‌سازی نشده AVI به‌صورت سیاه و سفید ذخیره می‌شود.
hasframe	مشخص می‌کند که آیا فرمی در ویدئو v موجود می‌باشد برای خواندن یا خیر. به صورت iteration عمل می‌کند رو فرم‌ها.
	hasFrame(v) خروجی‌اش Trueیا False می‌باشد.
getframe	در axes فعلی به عنوان یک فریم فیلم عکس نگه می‌دارد. و هرچه خارج از آن axes باشد را نگه نمی‌دارد. خروجی این دستور cdata که شامل اطلاعات عکس و colormap می‌باشد.
	getframe getframe(gcf,[0 0 560 420]) از gcf در مستطیلی که ابعادش ۵۶۰ در ۴۲۰ و در پایین و سمت چپ gcf است فریم فیلم نگه می‌دارد.
readframe	فریم بعدی ویدئو را می‌خواند.

	video = readFrame(v) در این‌جا ورودی videoreader می‌باشد و فریم بعدی فریمی که تا الان خوانده شده بود را می‌خواند.
movie	فرم‌های یک فیلم را پخش می‌کند. در واقع یک ماتریس M که ستون‌هایش در واقع فرم‌های مختلف فیلم قرار دارد را پخش می‌کند. در اینجا M اجباری است اما می‌توانیم مقادیر دیگر n به معنی تعداد بار پخش فیلم یا fps به عنوان عدد تعداد فرم در یک ثانیه یا h به عنوان مختصات وسط شکل یا loc به عنوان مختصات گوشه چپ پایین فریم. movie(h,M,n,fps,loc)

۲,۱

ابتدا دستور زیر را در متلب وارد کردم و سپس از آبجکت v مقادیر زیر را برای جدول پیدا کردم:

```
v = VideoReader('inputs/Video-1.avi');
```

ویژگی	مقدار همراه با واحد/ نحوه استخراج
تعداد فریم‌های ویدئو	۱۵۰ فریم v.NumberOfFrames
نرخ فریم‌ها	۳۰ فریم بر ثانیه v.FrameRate
نوع فشرده‌سازی	Basic Windows bitmap format. از طریق kmplayer به media information این فایل دسترسی پیدا کردم و نوع فشرده سازی را از آنجا دیدم.
تعداد بیت در هر پیکسل	۲۴ v.BitsPerPixel
فرمت ویدئو	RGB24 v.VideoFormat
طول و عرض	۱۷۶ در ۱۴۴ پیکسل v.Width v.Height

۲ فشرده‌سازی و پردازش ویدئو

۱,۲

۱,۱,۲

دو فیلدی که می‌دهد یکی cdata می‌باشد یکی colormap. در cdata آرایه‌ای که تصویر در آن قرار دارد نگه‌داری می‌شود. در colormap فضای رنگی فریم نگه‌داری می‌شوند. در true color systemها این فیلد تهی می‌باشد. در این فریم هم colormap تهی می‌باشد.

شکل ۱ فریم اول ویدیو اول در تصویر بالا و colormap در تصویر پایین نمایش داده شده است.

۲,۱,۲

در فیلتر گوسی که من تشکیل دادم. ابعاد آن ۵ در ۵ و با سیگمای ۱ یک فیلتر تشکیل داده است. که در این فیلتر هر چه از وسط فیلتر به گوشه‌های فیلتر می‌رویم ضرایب کوچک‌تر می‌شود ولی خانه‌های وسط فیلتر ضریب بالایی دارند. از این فیلتر برای کاهش نویز استفاده می‌شود. این عمل باعث می شود تا حدی حجم خروجی کم شود.

۳,۱,۲

در این قسمت از فیلتر median باید استفاده می‌کردیم به این‌گونه که در هر رنگ از هر فریم باید این فیلتر اعمال شود و سپس این سه رنگ با هم یک فریم را تشکیل می‌دهند و در نهایت با استفاده از فریم‌ها ویدئو را می‌سازیم. در این حالت بار محاسباتی بیشتر می شود نسبت به حالت قبل ولی فشرده سازی هم به مراتب بهتر می شود. همچنین کیفیت تصویر نیز از حالت قبل بهتر می باشد.

۴,۱,۲

در این قسمت هر فریم را با فریم قبلی و بعدی‌اش مقایسه می‌کنیم. آن‌هایی که از فریم قبلی و از فریم بعدی اختلاف MSE بیش‌تر از ۱۰ داشته باشند را نگه می‌داریم و اگر کمتر از این مقدار MSE داشته باشد فریم قبل یا بعد آن را به‌جای آن تکرار می‌کنیم.
این کار باعث می‌شود که ویدیو سریع تر نشود و تعداد فریم ها ثابت بماند. ولی باعث می‌شود که اختلاف فریم های متوالی حداقل بشود و با اینکار حجم ویدیو کاهش یابد.
در واقع ما در هر مرحله فریم را فریم قبلی و بعدی مقایسه میکنیم و در صورت نیاز از فریم قبلی یا بعدی بجای آن استفاده می‌کنیم و فریم بعدی را در فریم بعدی آن قرار می‌دهیم. در مرحله بعدی ما دو فریم به جلو می‌پریم و همین مرحله را تکرار می‌کنیم. اگر بخواهیم نشان دهیم که چگونه فریم‌های I و B قرار می‌گیرند می‌توانیم به صورت IBI در نظر بگیریم که بجای فریم وسطی فریم قبلی یا بعدی قرار می‌گیرد.

۵,۱,۲

در این قسمت هر فریم از ویدئو را می‌خوانیم سپس با استفاده از دستوری imresized هر فریم را ابعادش را نصف می‌کنیم و در ویدئو جدید قرار می‌دهیم.

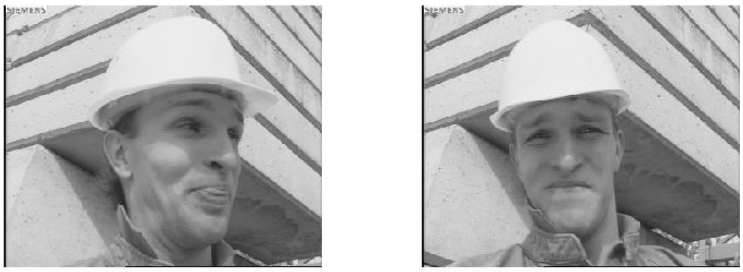
۶,۱,۲

برای دو ویدئو و برای هر فریم از آن‌ها، ابتدا آن فریم را به فضای YcBcR می‌بریم، سپس مقدار Y را به عنوان Luminance انتخاب می‌کنیم و سپس بین این دو psnr را محاسبه می‌کنیم. psnr برای فریم ها را جمع می‌کنیم و در نهایت تقسیم بر تعداد فریم‌ها می‌کنیم. از این رو میانگین psnr فریم‌های این دو ویدئو به‌دست می‌آید. این مقدار برای این دو ویدئو ۳۲,۷۱۸ به‌دست می‌آید.

۲,۲

۱,۲,۲

ما برای پیدا کردن بردار انتقال بین دو بلاک در دو تصویر نیاز داریم تا MSE متناظرشان را پیدا کنیم، استفاده از تصویر grayscale نسبت به RGB تاثیر بیش‌تری در سرعت این پیدا کردن دارد. همچنین نتیجه قابل قبول و نزدیک نسبت به استفاده از RGB به ما می‌دهد.



شکل ۲ تصویر سیاه و سفید دو فریم

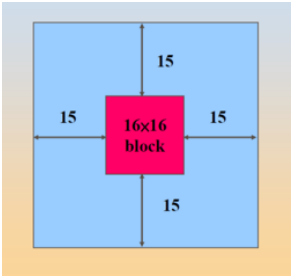
۲,۲,۲

فریم دوم را از فریم اول کم کردیم و نتیجه در شکل ۳ قابل مشاهده می‌باشد.

شکل ۳ اختلاف دو فریم با یکدیگر

۳,۲,۲

ما هر بلاک ۱۶ در ۱۶ در تصویر دوم را برمی‌داریم و در تصویر اول در پنجره‌ای که در شکل ۴ مشخص شده‌است، به‌دنبال آن می‌گردیم. این پنجره را انتخاب کردیم تا بتوانیم با سرعت بیش‌تری عمل block matching را انجام بدهیم.



شکل ۴ پنجره جست‌وجویی که در تصویر اول به دنبال بلاک متناظر با بلاک مشخص‌شده در تصویر دوم می‌گردیم.

می‌توانید اختلاف تصویر حاصل از motion compensation را با فریم اول در شکل ۵ مشاهده کنید. همان‌طور که مشخص است نسبت به شکل ۳ پیکسل‌های بیش‌تری سیاه شده‌اند.

شکل ۵ تصویر حاصل از اختلاف motion compensation از فریم اول با استفاده از روش full search

۴,۲,۲

ما مانند شکل ۴ پنجره جست‌وجو خواهیم داشت و آن را به ۴ قسمت تقسیم می‌کنیم سپس آن قسمتی که کمترین MSE را با بلاک ما دارد انتخاب کرده و به‌صورت بازگشتی به مرور محیط جست‌وجو را کوچک‌تر می‌کنیم تا جایی که دیگر نتوانیم جست‌وجو انجام بدهیم. در آن‌جا مکانی که قرار داریم و هم‌چنین اختلاف بلاک با آن ناحیه را محاسبه می‌کنیم و به عنوان خروجی می‌دهیم. در ماتریس mov بردار حرکت هر بلاک تصویر دوم نسبت به فریم اول ذخیره شده‌است. می‌توانید در شکل ۶ خروجی تابع را مشاهده کنید.

شکل ۶ تصویر حاصل از اختلاف motion compensation از فریم اول با استفاده از روش divide and conquer

۳.۲

ابتدا تابع cutDetection() را توضیح می‌دهم که به این صورت عمل می‌کند که ۱۰ فریم، ۱۰ فریم بررسی می‌کند که آیا تغییر زیادی ویدیو داشته است یا خیر. این عمل را با استفاده از MSE محاسبه می‌کند. اما قبل از آن با استفاده از فیلتر گاوسین که رو این دو فریم ابتدایی و انتهایی اعمال می‌شود تا نویز ها و تغییرات کوچک در MSE به حساب نیایند.

اگر تغییر زیادی بین این دو فریم که از هم فاصله ۱۰ فریم دارند مشاهده شد، آنگاه یک به یک در فریم‌ها جلو می‌آید تا بتواند دقیقا محل آن تغییر را شناسایی کند. این عمل هم با استفاده از فیلتر گاوسین و MSE انجام می‌شود.

این تابع را در ویدیو Cut.mpeg و cbswipe.mpg اعمال کردم و نتایج زیر حاصل شد:

index cut in Cut.mpeg:
154

index cut in cbswipe.mpg:
42 214 293 349 445 563 574

که این یعنی به عنوان مثال در فایل Cut.mpeg در فریم ۱۵۴ تغییر رخ داده است و در فریم ۱۵۵ صحنه دیگری نمایش داده شده است.

شکل ۷ در این‌جا بین فریم ۵۶۳ و ۵۶۴ از ویدیو cbswipe یک cut تشخیص داده شده است.

حال dissolveDetection() را توضیح می‌دهیم. مانند حالت قبلی ۲۰ فریم، ۲۰ فریم به سمت جلو می‌رویم و به دنبال تغییر بزرگ می‌گردیم. وقتی یک تغییر بزرگ را مشاهده کردیم. آن‌گاه در این محدوده ۲۰ فریمی به دنبال نقطه شروع و پایان انجام افکت dissolve می‌گردیم. این کار را از این طریق انجام می‌دهیم که یک بار از آخر شروع می‌کنیم و به سمت نقطه شروع حرکت می‌کنیم و به دنبال فریمی می‌گردیم که نسبت به فریم آخر تغییرات زیاده را داشته باشد. آن فریم را به عنوان فریم پایانی افکت در نظر می‌گیریم. یک بار هم از اول شروع می‌کنیم و به سمت نقطه پایان می‌رویم تا فریم اولیه افکت را به‌دست آوریم.

سپس با استفاده از دو نقطه‌ای که که به عنوان شروع و پایان dissolve شناخته شده‌اند، با یک عمل خطی این دو فریم را با هم ترکیب می‌کنیم و با فریم متناظرش در ویدیو MSE را محاسبه می‌کنیم. اگر مقدار آن از ترشهولد کم‌تر بود، یک counter

را اضافه می‌کنیم. اگر در آخر این counter برابر با فاصله بین این دو نقطه شروع و پایان بود یعنی این تغییر به عنوان یک تغییر dissolve شناخته شده‌است.

شکل ۸ در اینجا الگوریتم از فریم ۱۶۲ تا فریم ۱۷۹ تشخیص حضور یک dissolve را در ویدیو اول کرده است.

با اعمال تابع dissolveDetection() روی دو ویدیو Dissolve.mpeg و cbswipe.mpg نتایج زیر حاصل شده‌اند:

index dissolve in Dissolve.mpeg:
  162 179

index dissolve in cbswipe.mpeg:

همان‌طور که مشاهده می‌شود در فایل cbswipe.mpeg هیچ dissolveای تشخیص داده نشده است.

و آخرین تابع که wipeDetection() می‌باشد، تمام مراحلش مانند dissolveDetection() انجام می‌شود با این تفاوت که پس از اینکه به صورت خطی این عمل را انجام داد و به دنبال dissolve گشت، اگر dissolve نبود پس wipe می‌باشد. به عنوان مثال اگر نصف بیشتر از آن فریم‌هایی که خطی به دست آمده‌اند که نسبت به فریم متناظرشان ترشهولد کم‌تری داشت آنگاه به عنوان wipe شناخته می‌شود.

با اعمال تابع wipeDetection() روی دو ویدیو Wipe.mpg و cbswipe.mpg نتایج زیر حاصل شده‌اند:

index wipe in Wipe.mpg:
  161 179

index wipe in cbswipe.mpg:
  201 213 361 367

در اینجا به عنوان مثال در فایل cbswipe.mpeg دو عدد wipe تشخیص داده شده است که یکی از بازه [۲۱۳ ,۲۰۱] رخ داده است و دیگری در بازه [۳۶۷ ,۳۶۱] رخ داده است.

شکل ۹ wipe تشخیص داده شده بین فریم های ۲۰۱ و ۲۱۳ در ویدیو دوم