دانشگاه تهران

پردیس دانشکده‌های فنی

دانشکده مهندسی برق و کامپیوتر

عنوان

موزاییک ویدیو بی درنگ برای نوسان‌های حرکتی تند دوربین فیلم‌برداری

توسط

علی جهانی امیری

استاد راهنما

دکتر هادی مرادی

گزارش پروژه کارشناسی

رشته مهندسی برق

گرایش کنترل

شهریور 94

چکیده

استفاده از وسایل الکترونیکی امروزه بسیار گسترش یافته است و اکثر این وسایل قابلیت ضبط ویدئو را دارند و هر فردی می‌تواند در هر جایی و در هر موقعیتی ویدئویی را ضبط کند. در صورت قرار نداشتن دوربین روی پایه و یا سطح ثابت، این ویدئو تحت تاثیر لرزش‌های جزیی و بعضا حرکت‌های بسیار شدید قرار می‌گیرد و باعث پایین آمدن کیفیت تصویر ویدئو می‌شود.

حال اگر این دوربین از جایی آویزان شده باشد و یا به پایه غیرسلبی متصل باشد، مانند دوربین های آویزان شده در بالن عواملی مانند باد میزان تاثیر حرکت دوربین بر روی ویدئو به شدت افزایش می دهند و تصویری بسیار متحرک خواهیم داشت که فهم آن از محیط بسیار مشکل است.

هدف این گزارش پیاده سازی الگوریتمی بر پایه ویژگی[1] برای استفاده صحیح از تمام فریم های ویدیو برای ساخت موزاییک ویدیو[2] ای که بتواند بی درنگ[3] باشد می باشد .در عین حال تثبیت سازی ویدیو[4] نیز انجام شده است . بعنوان مثال در دوربینی که از بالن آویزان است با استفاده از این روش می توان نقش کنترل کننده برای زاویه دید دوربین که سخت افزاری است را حذف کرد که نتیجه آن کاهش وزن و صرفه جویی در هزینه و... می باشد با پیاده سازی این روش، تکان خوردن دوربین در جهات مختلف توسط باد نه تنها منفی بلکه مثبت تلقی می شود.

کلمات کلیدی:

تثبیت سازی ویدیو ، ویدیو موزاییک بی درنگ ، پانوراما ویدیو ، پانوراما اتوماتیک

Keywords:

Video stabilization, Real-time video mosaic, Automatic video panorama, video stitching

[1] Feature based
[2] Video Mosaic
[3] Real-time
[4] Video Stabilization

فهرست مطالب

فهرست شکل‌ها

فهرست جداول

۱- مقدمه

حرکت و لرزش ناخواسته دوربین از مشکلاتی است که همواره کیفیت خروجی فیلم را تحت تاثیر قرار می دهد. در دوربین‌های خانگی این لرزش عمدتاً ناشی از لرزش‌های دست فیلمبردار می‌باشد که با قرار دادن دوربین روی سه‌پایه تا حد قابل توجهی رفع می‌شود. اما در کاربردهایی که دوربین روی جسمی غیر صلب و یا در حال حرکت قرار دارد، شرایط بسیار دشوارتر است. به عنوان مثال، وقتی یک دوربین نظارتی از یک بالن در فرآیند رسیدگی به حادثه دیدگان آویزان است و یا وقتی فیلبردار در حال دویدن می‌باشد میزان نوسانات دوربین با لرزش‌های ناشی از حرکت دست در یک دوربین خانگی قابل مقایسه نیست. روش‌های متعددی برای رفع خرابی‌های ناشی از لرزش دوربین پیشنهاد شده‌اند، ولی اکثر آن‌ها روی لرزش‌های جزیی ناشی از لرزش دست تمرکز کرده‌اند. هدف از این پروژه ارائه راهکاری بدون استفاده از سنسور های حرکتی برای کاهش تاثیر لرزش‌هایی است که در حرکت دوربین به صورت آونگی رخ می‌دهد مانند دوربینی که به یک پایه غیر سلب متصل آویزان است.

تقریباً تمامی دوربین‌های فیلمبرداری امروزی امکان حذف لرزش دست از فیلم گرفته شده را دارند. اما اگر این حرکت به همان صورتی که در بالا توصیف شد دامنه وسیعی داشته باشد، یا به عبارتی آونگی باشد، الگوریتم‌ها و یا تجهیزات سخت‌افزاری این دوربین‌ها دیگر قادر به حذف خرابی‌های ناشی از لرزش نیستند اکثر رویکردهای موجود بر ویژگی حرکت کم و خطی لرزش دست تمرکز دارند و حرکت های آونگی را پوشش نمی دهند. برای داشتن فیلمی با کیفیت خوب با هر نوع حرکت دوربین، نیاز به بررسی حرکت آونگی دوربین و ویژگی‌های آن است. در واقع اگر بخواهیم با همان شیوه های سنتی مکانیزم های حذف نویز حرکتی را برای حرکت های آونگی انجام دهیم داده های زیادی را از دست می دهیم بعنوان مثال فریم هایی که در گوشه های حرکت آونگی بوده اند را از دست می دهیم که راه حل در نظر گرفته شده ایجاد و پیاده سازی یک موزاییک ویدیو از ویدیوی در حال نوسان بوده است.

در این پروژه تلاش شده است روند پیاده سازی یک موزاییک ویدیو به صورت مرحله به مرحله با استفاده از کتابخانه های opencv به زبان c++ با الگوریتم بهینه ای که بتواند بی درنگ[1] کار کند ارایه شود.

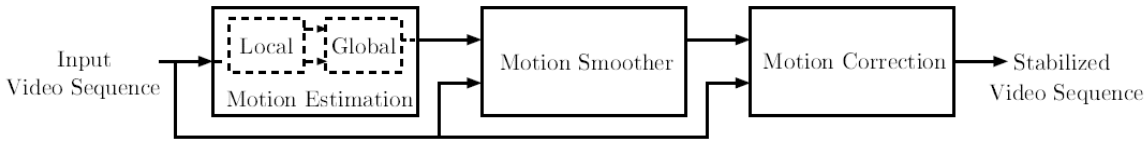
[1] Real-time

فصل 2

۱ موزاییک ویدیو

۱.۱ روش های سنتی برای کاهش لرزش :

اکثر الگوریتم‌های کاهش تاثیر نوسان سنتی که فقط در زمینه لرزش های جزیی ناشی از دست تمرکز کرده اند به صورت زیر می باشد که از سه قسمت اصلی تشکیل شده‌اند: تخمین حرکت[۱]، نرم کردن حرکت[۲]، اصلاح حرکت[۳].



شکل 1- نمودار بلوکی الگوریتم‌های کاهش تاثیر نوسان

این الگوریتم ها برای حرکت نوسانی دوربین کافی نیستند تمرکز اصلی این مقاله بر روی نحوه پیاده سازی موزاییک ویدیو و دنبال کردن و قرار نهادن هر فریم در جای مناسب است. که ساختار کلی در قسمت های آتی بیان می‌شود.

۱.۲ ویدیو موزاییک چیست ؟

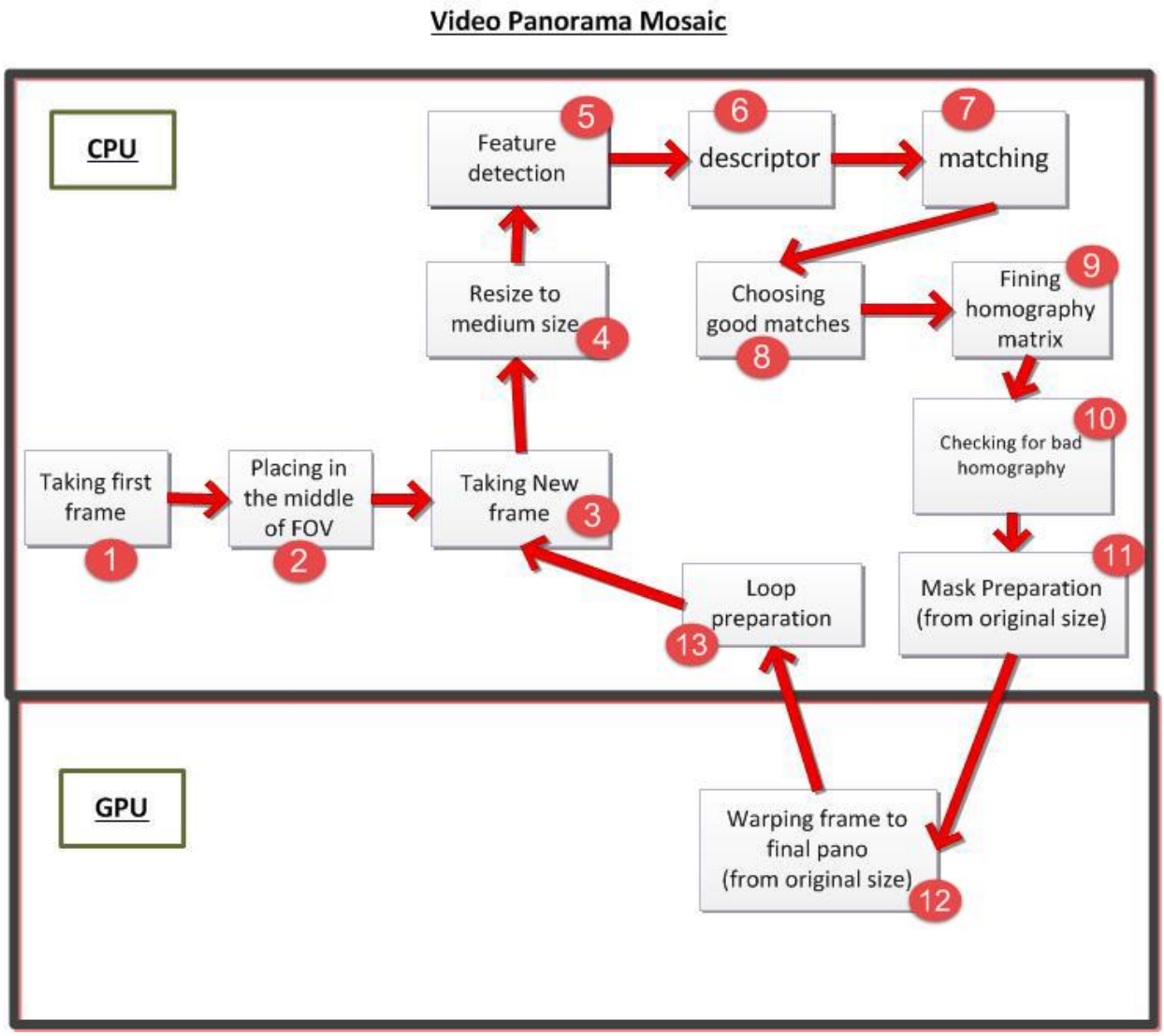
ویدیو موزاییک یا همان موزاییک ویدیو روشی است که بیشتر برای ویدیو های UAV (هواپیما ها یا بالن و سایر وسایل هوایی و نقشه برداری) که برای بررسی منطقه بکار می رود پیاده سازی می شود بدین گونه که هر عکس(فریم) گرفته شده تشخیص داده می شود که برای کدام قست از نقشه است و با فیلم گرفتن از منطقه نقشه ای بصورت کلی از منطقه داده می شود که با گرفتن هر عکس نقشه منطقه کاملتر می شود.

1- Motion Estimation
2- Motion Smoother
3- Motion Correction

شکل 2- نمونه ای از ویدیو موزاییک پیاده سازی شده با فریم بر ثانیه 15

فصل 3

۲ مراحل موزاییک سازی



شکل 3- نمودار بلوکی ساخت ویدیو موزاییک

۲.۱ گرفتن اولین فریم [1]

در این مرحله فریم اول فیلم را می گیریم. این فریم میتواند مستقیم از فایل ویدیو باشد یا به صورت زنده از دوربین وصل شده ضبط گردد.

[1] Taking first frame

۲.۲ جایگذاری در وسط میدان دید [1]

در این قسمت فریم اولیه را در وسط موزاییک قرار می دهیم.

۲.۳ گرفتن فریم جدید [2]

در این مرحله فریم جدید را می گیریم. همانند مرحله اول این فریم می تواند مستقیم از فایل ویدیو باشد یا به صورت زنده از دوربین وصل شده ضبط گردد.

۲.۴ تغییر سایز به اندازه متوسط [3]

بدلیل اینکه الگوریتم باید با سرعت بالایی اجرا شود و با توجه به اینکه اگر رزولوشن تصویر بالا رود مرحله پردازش بشدت کند می شود بهتر است سایز تصویر را برای ادامه مراحل کوچک کرده و در نهایت برای انتقال به موزاییک ویدیو آن فریم را با سایز اصلی قرار دهیم.

۲.۵ تشخیص ویژگی [4]

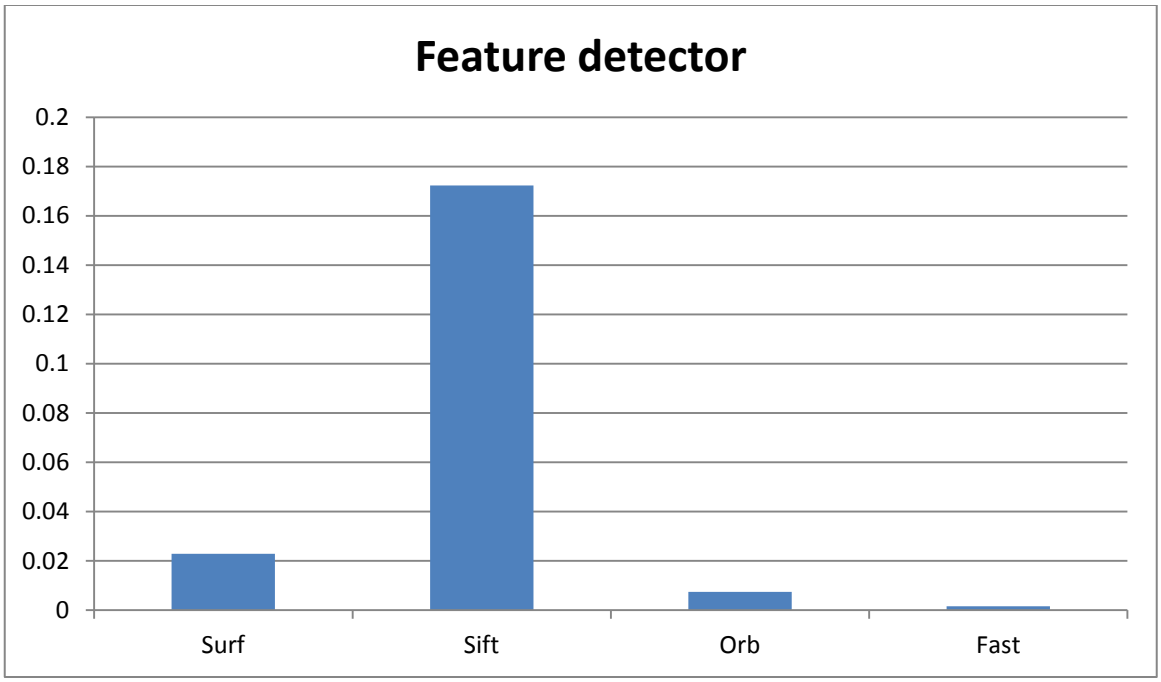
در این مرحله ویژگی های تصویر با توجه به الگوریتم های از پیش نوشته شده در کتابخانه های Opencv بدست می آوریم . 200 ویژگی(پیکسل های مهم مانند گوشه ها و ...) در عکس کفایت می کند.

از آنجا که سرعت پردازش از اصلی ترین معیار برای بی درنگ بودن محسوب میشود مقایسه ای بین جستجوگر های مختلف را برای داده های خودم بدست آوردم

	SurfFeatureDetector	SiftFeatureDetector	OrbFeatureDetector	FastFeatureDetector
Threshold	130	200	200	16
Features	105	101	118	178
Time(s)	0.0228395	0.172254	0.0074782	0.00161182

جدول 1 – مقایسه زمانی بین جستجوگرهای ویژگی

[1] Placing in the middle of Field of View
[2] Taking new frame
[3] Resize to medium size
[4] Feature detection



شکل 4 – مقایسه زمانی بین جستجوگرهای ویژگی

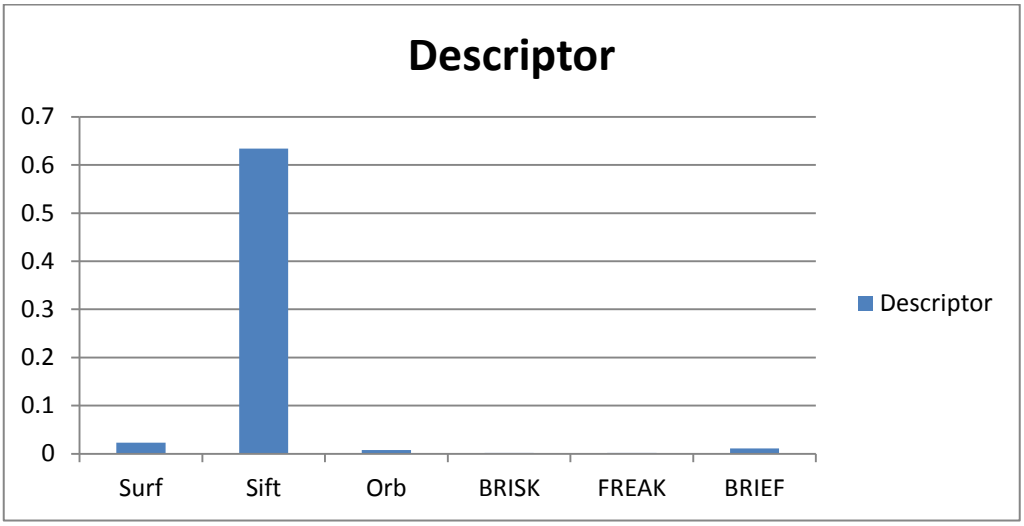
۲.۶ دسکریپتورها[1]

این قسمت پیکسل های مهمی را که در قسمت قبل مشخص شده بود را بررسی می کند و ویژگی منحصر بفرد و مقاوم را با توجه به همسایه های آن پیکسل بدست می آورد مثلا شدت رنگ[2] پخش شده در پیکسل های همسایه. بنابراین تمامی ویژگی های تصویر[3] که در مرحله قبل بدست آمد با بررسی همسایه آن ویژگی به آنها ویژگی های منحصر بفرد تخصیص داده می شود تا برای مرحله مطابقت سازی[4] آماده شوند.

[1] Descriptors
[2] Intensity
[3] features
[4] matching

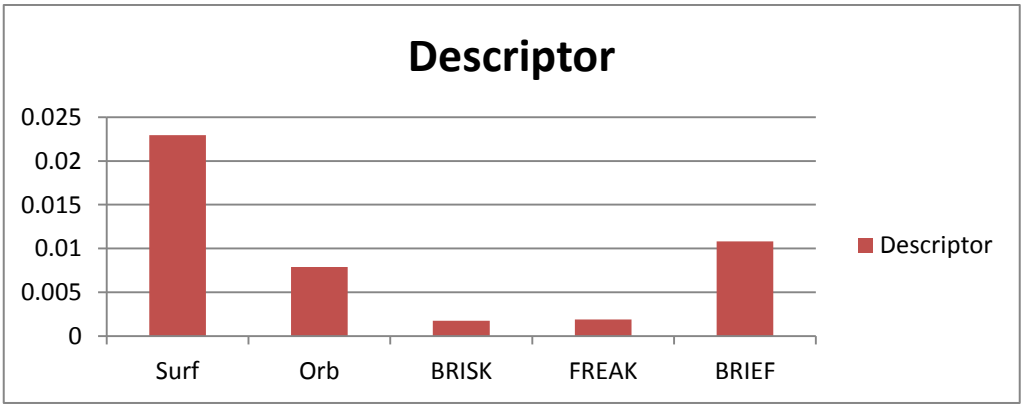
	Surf Descriptor Extractor	Sift Descriptor Extractor	Orb Descriptor Extractor	BRISK	FREAK	Brief Descriptor Extractor
time	0.0229385	0.633913	0.00790002	0.0017437	0.00190175	0.0108122

جدول 2 – مقایسه زمانی بین دسکریپتور ها



شکل 5– مقایسه زمانی بین دسکریپتور ها

برای مقایسه دقیق تر نمودار زیر بدون در نظر گرفتن الگوریتم Sift رسم شده است :



شکل 6– مقایسه زمانی بین دسکریپتور ها بدون Sift

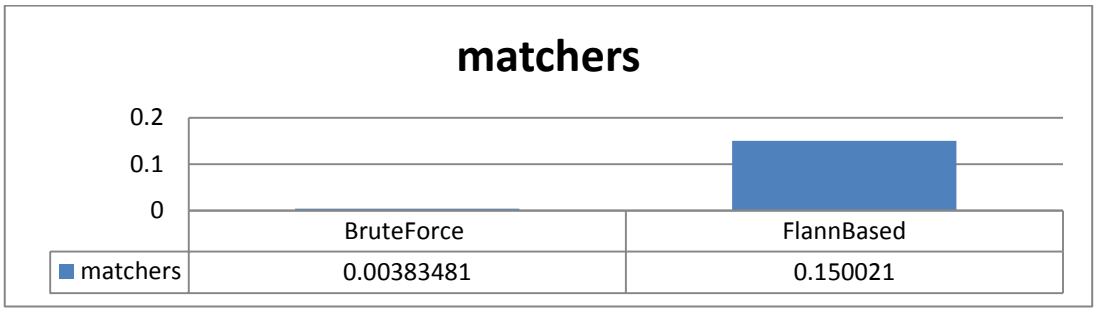
۲.۷ تطبیق[1]

در این مرحله ویژگی های منحصر به فرد فریم جدید با ویژگی های منحصر به فرد فریم قبلی، دو به دو مقایسه شده و با هم مطابقت داده می شوند.

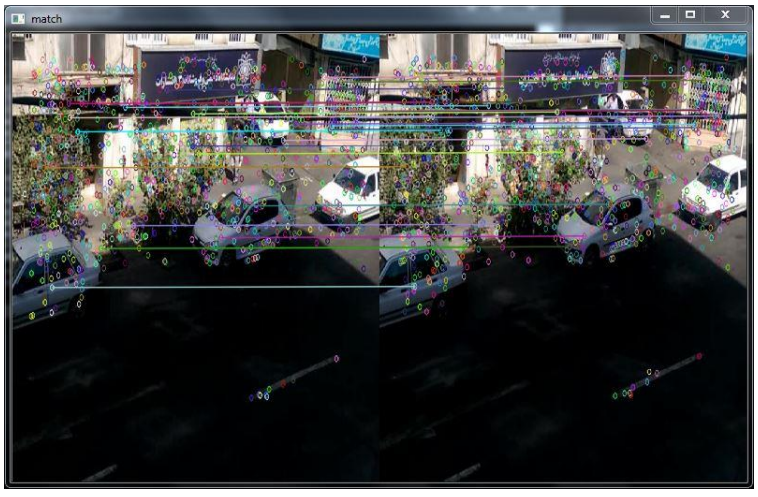
کتابخانه OPENCV 2.4 دو الگوریتم مختلف برای اینکار دارد.

Matcher (frame 48)	BruteForce matcher	FlannBasedMatcher
time	0.00383481	0.150021

جدول 3 - مقایسه زمانی بین matchers



شکل 7- مقایسه زمانی بین matchers



شکل 8- تطبیق بین دو فریم. نقاط دایره ای ویژگی ها هستند و خطوط ویژگی های تطبیق داده شده هستند

[1] Matching

۲.۸ انتخاب تطبیق های خوب[1]

حال تا اینجا فهمیدیم کدام پیکسل ویژگی در فریم قبلی به کدام پیکسل ویژگی در فریم جدید مشابه است. باید در نظر بگیریم همواره مقداری خطا در تمامی الگوریتم ها ممکن است باشد. برای اینکه خطا را مینیمم کنیم باید با استفاده از الگویتم هایی، تطابق های اشتباه را جدا کنیم. برای اینکار از یک راه ابتکاری استفاده کردم. به این صورت که فاصله پیکسلی بین تمامی ویژگی های تطبیق داده شده را بدست آوردم. بعنوان مثال ویژگی اول 10 پیکسل جا به جا شده، ویژگی دوم 15 پیکسل جابه جا شده و... . حال مینیمم این فاصله ها را بدست می آوریم. اگر فاصله ویژگی ها از 5 برابر مینیمم بیشتر بود آن تطبیق را حساب نمی کنیم و همینطور فقط 50 فاصله کمتر را بعنوان تطبیق های خوب در نظر می گیریم.

۲.۹ پیدا کردن ماتریس انتقال[2]

دو نوع از ماتریس های انتقال پیاده سازی شده است که نتایج آن در انتها آورده شده است.

۲.۹.۱ تبدیل افاین[3]

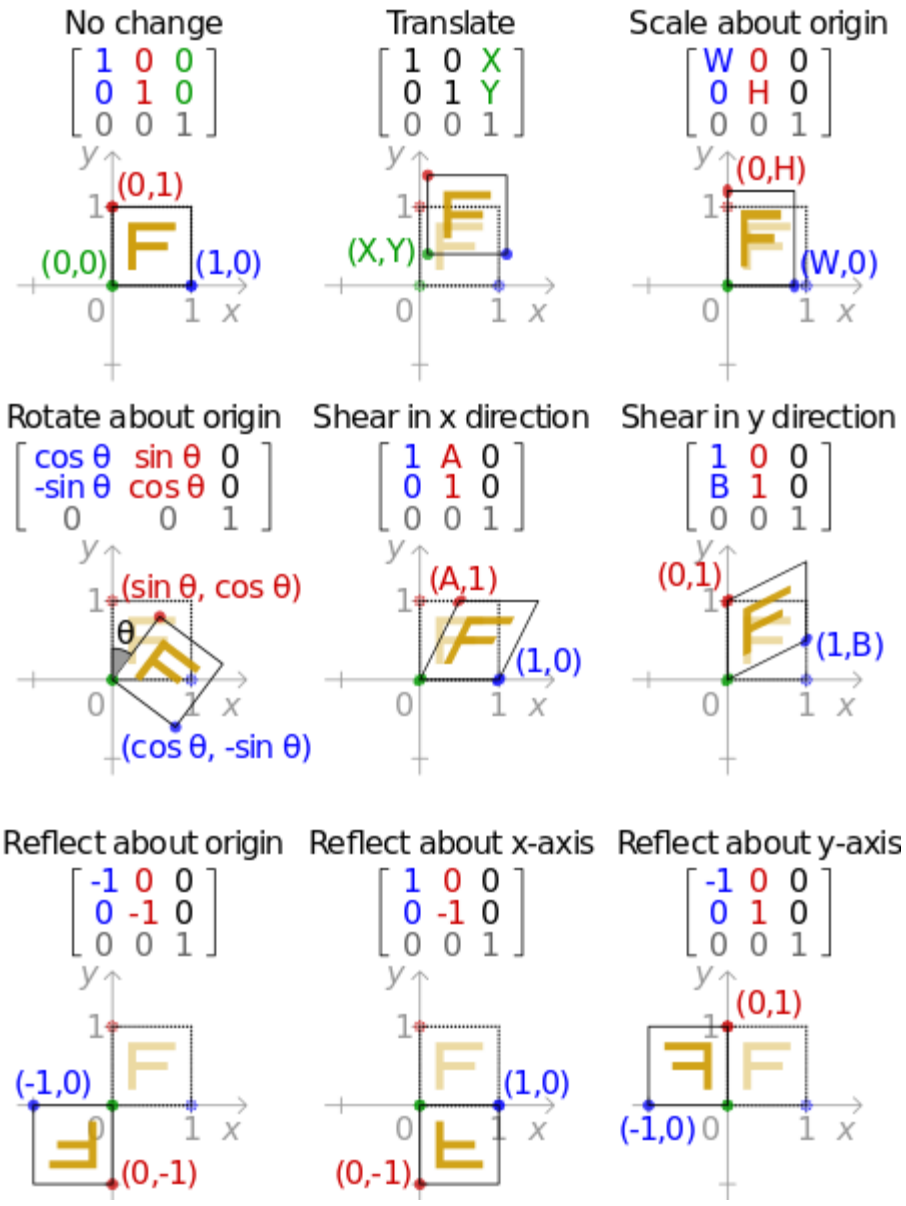
این تبدیل دارای 6 پارامتر می باشد که در این مدل تبدیل بین دو پیکسل را می‌توان به صورت زیر نشان داد:

$$\begin{pmatrix} x \\ y \\ 1 \end{pmatrix} = \begin{pmatrix} h11 & h12 & h13 \\ h21 & h22 & h23 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix}$$

$(h13,\ h23)$ جابجایی خطی تصویر و $(h11,\ h12,\ h21,\ h22)$ میزان چرخش، تغییر مقیاس و کشیدگی تصویر را تعیین می‌کنند.

با داشتن 3 نقطه از تصویر اول که یک مثلث را تشکیل بدهند و با داشتن یک مثلث دیگر در تصویر دوم میتوان ماتریس تبدیل را با روش های بازگشتی یافت. در این تبدیل خطوطی که موازیند موازی باقی می مانند.

[1] Choosing good matches
[2] Finding homography matrix (translation matrix)
[3] Affine transformation



شکل 9- نمونه ای از ماتریس های تبدیل افین

۲.۹.۲ تبدیل پرسپکتیو[1]:

این تبدیل همانند تبدیل بالا می باشد با این تفاوت که دارای 8 پارامتر است. بدین گونه که در راستای z هم می توان عملیات چرخش و ... را انجام داد. این تبدیل به واقعیت نزدیکتر است. برای محاسبه

[1] Perspective transform

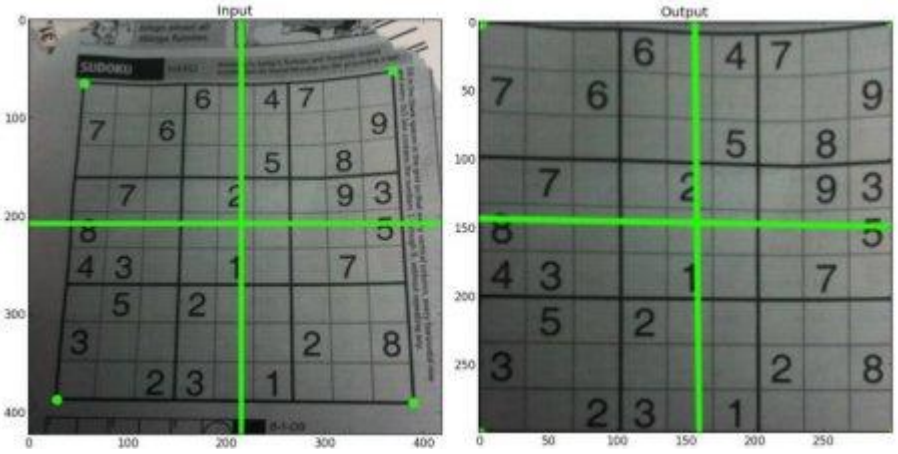
این تبدیل باید 4 نقطه که هیچ 3 تایی از آن ها همخط نیستند هم از تصویر اول هم از تصویر دوم داشت تا بتوان تبدیل آنرا محاسبه کرد.

$$X' = a_1 X + a_2 Y + a_3 Z \ , \ Y' = a_4 X + a_5 Y + a_6 Z \ , \ Z' = a_7 X + a_8 Y + a_9 Z$$

$$focal\ length = 1 \ , \quad x' = \frac{X'}{Z'} \qquad y' = \frac{Y'}{Z'}$$

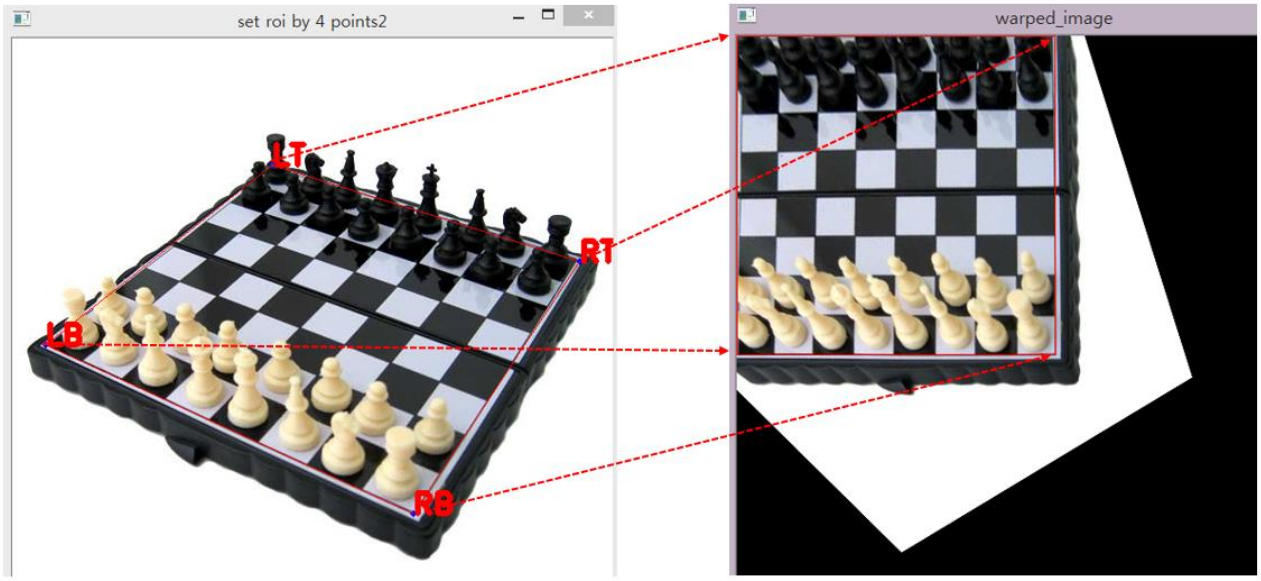
$$x' = \frac{a_1 X + a_2 Y + a_3 Z}{a_7 X + a_8 Y + a_9 Z} \ , \quad y' = \frac{a_4 X + a_5 Y + a_6 Z}{a_7 X + a_8 Y + a_9 Z}$$

$$x' = \frac{a_1 x + a_2 y + a_3}{a_7 x + a_8 y + a_9} \ , \quad y' = \frac{a_4 x + a_5 y + a_6}{a_7 x + a_8 y + a_9}, \qquad a_9 = 1$$



شکل 10- مثالی از تبدیل پرسپکتیو (1)

شکل 11- مثالی از تبدیل پرسپکتیو (2)



شکل 12- مثالی از تبدیل پرسپکتیو(3)

۲.۹.۳ الگوریتم RANSAC [1]

بدلیل اینکه تعداد زیادی ویژگی در عکس ها بدست آوردیم و اگر از تبدیل افاین استفاده می کنیم فقط 3 نقطه از هر دو عکس و اگر از تبدیل پرسپکتیو استفاده می‌کنیم 4 نقطه کافیست باید بعضی ازاین نقاط را در نظر نگیریم. این الگوریتم بهترین نقاط را انتخاب میکند و باعث می‌شود جوابی مقاوم[۲] داشته باشیم. در واقع اگر نقطه ای هم اشتباه تطبیق داده شد خطا بسیار کم است.

۲.۹.۴ محاسبه ماتریس انتقال کلی:

تا اینجا با تعاریف ماتریس انتقال آشنا شدیم. برای موزاییک ویدیو نیاز به یک عکس مرجع داریم تا بقیه عکس ها با آن عکس سنجیده شود و ماتریس انتقال همه نسبت به یک عکس مشترک بین همه سنجیده شود. برای همین اولین عکسی را که در مرکز صفحه قرار دادیم به عنوان عکس مرجع انتخاب می‌کنیم. از ویژگی های ماتریس های تبدیل در زیر آمده شده است :

ماتریس تبدیل عکس A به C = ماتریس تبدیل عکس A به B * ماتریس تبدیل عکس B به C

[1] Random sample consensus
[2] Robust

بنابراین از این ویژگی استفاده کرده و بعد از محاسبه ماتریس تبدیل دو فریم پشت سر هم، آن ماتریس تبدیل را در ماتریس تبدیل قبلی ها ضرب می‌کنیم تا ماتریس تبدیل نسبت به عکس اول بدست آید:

Transition_n_to_one = Transition _n_to_n-1 * Transition_n-1_to_n-2 *... * Transition_2_to_1

و برای عکس اول هم باید به آن یک آفستی داد تا به مرکز تصویر منتقل شود :

$$\begin{pmatrix} 1 & 0 & \text{offset}_x \\ 0 & 1 & \text{offset}_y \\ 0 & 0 & 1 \end{pmatrix} \times \begin{pmatrix} R_{11} & R_{12} & t_x \\ R_{21} & R_{22} & t_y \\ P & P & 1 \end{pmatrix}$$

این ماتریس آفست (ماتریس سمت چپ در بالا) فقط یکبار در ماتریس هموگرافی در ابتدا پیش ضرب می‌شود تا مبدا کل تصاویر را مرکز FOV کند.

در واقع محاسبات ما بصورت زیر می شود:

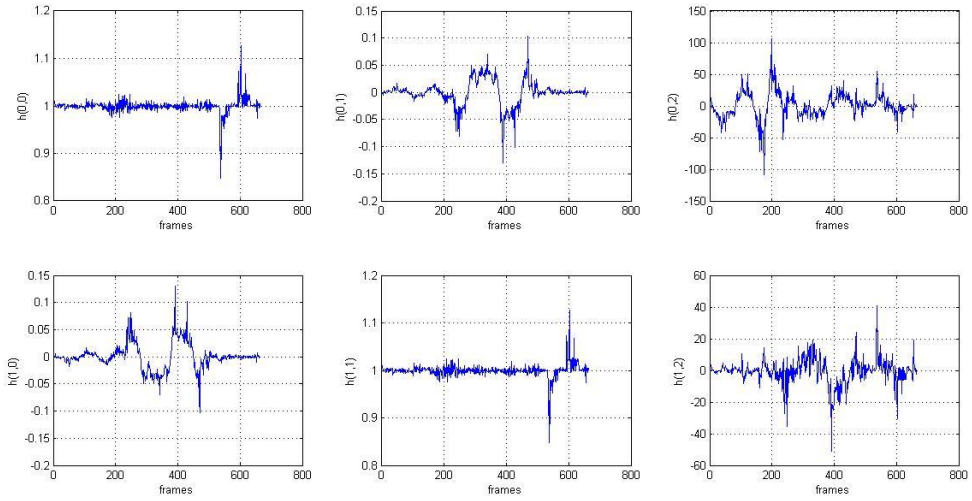
H2to1 = H_{offset} * H2to1
H3to1 = H3to2 * H2to1
...
Hnto1 = H(nton-1) *...*H2to1

توابعی که ماتریس تبدیل بین دو تصویر را بدست می آورند:

برای تبدیل افاین : `estimateRigidTransform`

برای تبدیل پرسپکتیو : `findHomography`

شکل زیر تغیرات 6 پارامتر تبدیل افین برای دیتای خودم است:



شکل 13- تغییرات پارامتر های ماتریس افاین

همانطور که مشاهده میشود تغییرات بسیار نویزی است که با فیلتر کالمن درآینده می‌توان آنرا بهبود بخشید.

۲.۱۰ بررسی کردن ماتریس انتقال[1]:

بدلیل اینکه حرکت دوربین تغییرات آنی ندارد و پیوسته است پس بنابراین تغییرات آنی در ماتریس انتقال نشان دهنده خطا است. بدلیل اینکه این ماترس ها بصورت متوالی در هم ضرب می شوند اگر یکی از آن‌ها اشتباه باشد کل سیستم را دچار خطا می‌کند حتی اگر تصاویر بعدی ماتریس‌هایشان درست محاسبه شده باشد. بنابراین الگوریتمی که بکار گرفتم بدین صورت بود که اگر همبستگی[2] ماتریس انتقال با ماتریس قبلی محاسبه شده بیشتر از عددی مشخص بود یا اندازه ماتریس انتقال(نرم 2) بیشتر از 10 برابر میانگین بود بنابراین این فریم را در نظر نمی‌گیریم و به فریم بعدی می رویم.

۲.۱۱ آماده سازی ROI[3]:

بعد از اینکه ماتریس انتقال را پیدا کردیم نیاز به ناحیه ای داریم که فریم جدید را در آن ناحیه کپی کنیم بدون اینکه تغییری در بقیه قسمت ای موزاییک ایجاد شود. درواقع باید ماسکی باینری بسازیم که

[1] Checking for bad transition matrix
[2] Correlation
[3] Mask preparation

پیکسل سفید به معنی اینکه عملیات روی آن پیکسل‌ها فقط انجام می شود و بر روی پیکسل های مشکی در صفحه موزاییک تغییری ایجاد نمی‌شود

شکل 14- نمونه ای از ماسک در فریم 50

متدهای قدیمی:

برای ساخت این ماسک معمولا یک صفحه سفید به اندازه فریم‌های گرفته شده در نظر می‌گیرند و ماتریس انتقال را بر روی تمامی پیکسل هایش به طوری که تصویر خروجی به اندازه سایز نهایی موزاییک شود اعمال می‌کنند (حتی در مثال‌های رسمی OpenCV برای پانوراما ساختن نیز اینکار را انجام داده است). باید توجه داشت اعمال ماتریس انتقال بر روی تمامی پیکسل‌ها کاری زمان بر است و قطعا باعث میشود که الگوریتم با سرعت پایین‌تری اعمال شود.

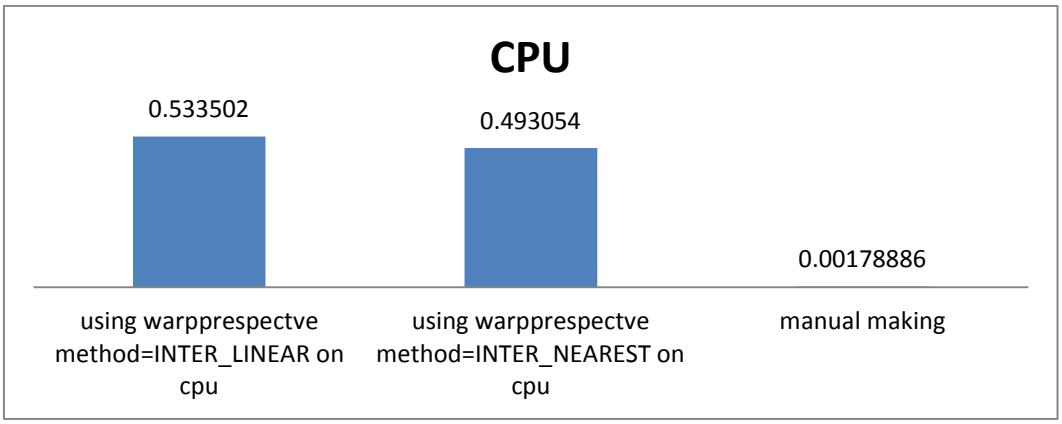
راه ابتکاری:

برای ساخت ماسک نیازی به اعمال ماتریس به همه پیکسل‌ها نیست. اعمال به 4 پیکسل گوشه تصویر سفید کافی است تا 4 گوشه ماسک بدست آید. سپس با استفاده از تابع fillConvexPoly نقاط داخل این 4 نقطه را با رنگ سفید پر می‌کنیم. انجام اینکار بسیار سریعتر از استفاده از روش های قدیمی حتی باپردازش روی کارت گرافیک است.

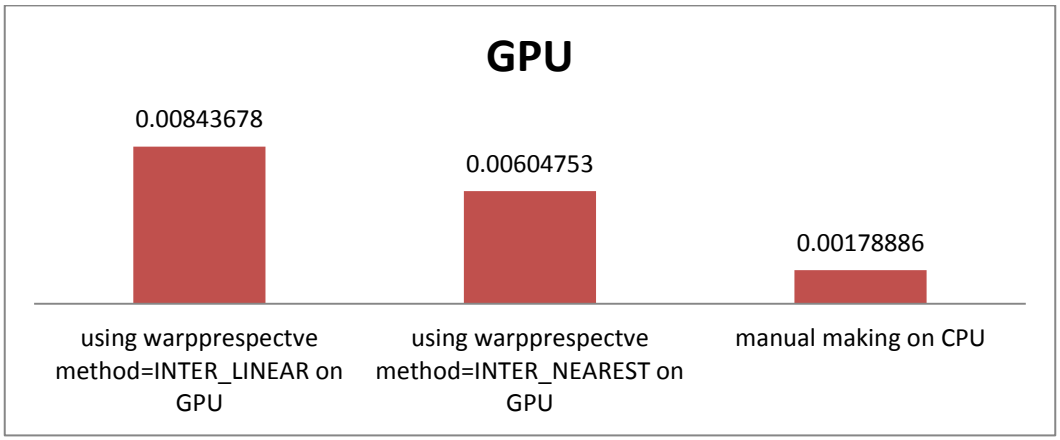
جداول و نمودار های زیر برای ساخت ماسک 3840*1200 پیکسلی انجام شده است:

	WarpPrespective Inter_Linear	WarpPrespective Inter_Nearest	My method
CPU (second)	0.533502	0.493054	0.00178886
GPU(second)	0.00843678	0.00604753	

جدول 4 – مقایسه زمانی روش های ساخت ماسک



شکل 15- مقایسه زمانی روش های ساخت ماسک در **cpu**



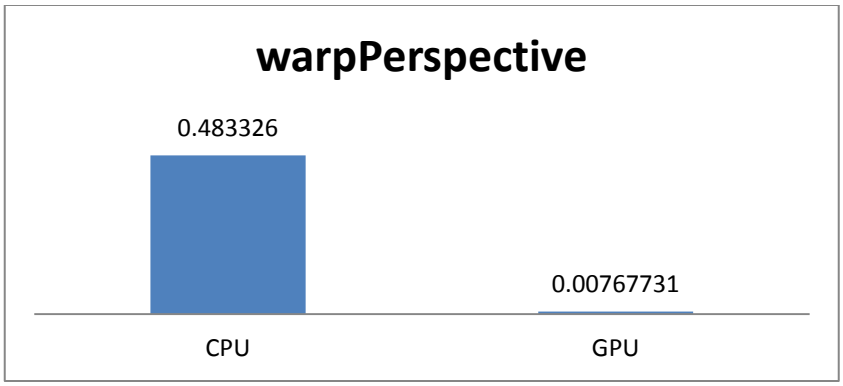
شکل 16-مقایسه زمانی ساخت ماسک در کارت گرافیک

همانطور که مشاهده می شود زمان بسیار کمتری برای ساخت ماسک در این روش صرف می شود.

۲.۱۲ اعمال فریم جدید به موزاییک نهایی:[1]

بعد از ساخت ماسک حال باید ماتریس انقال را به تصویر اعمال کنیم و سپس خروجی آن را با استفاده از ماسک به تصویر نهایی اعمال کنیم. همانطور که در بالا اشاره شد مرحله ی اعمال ماتریس بر تصویر اگر سایز تصویر بالا باشد بشدت زمان محاسبه بالا می رود بنابراین این قسمت از پردازش باید بر روی کارت گرافیک اجرا شود.

[1] Warping image to final mosaic



شکل 17- مقایسه زمانی بین اجرای در CPU و GPU

۲.۱۳ آماده سازی حلقه :[1]

در انتهای حلقه باید فریم جدید و ویژگی ها و دسکریپتورهایش و بعضی موارد جزیی لازم برای اجرای حلقه در دور جدید را به جای فریم و ویژگی هایش و دسکریپتورهای فریم قبلی قرار دهیم. مرحله بعدی دوباره گرفتن فریم جدید است و این پروسه تکرار می شود.

[1] Loop preparation

فصل 4

۳ روند پروژه

۳.۱ مشکلات بوجود آمده در حین انجام پروژه :

بدلیل ارایه شدن نسخه جدید کتابخانه های ورژن OpenCV 3.0.0 در طی هفته‌ی جاری به این نسخه رجوع کردم اما هنوز این نسخه دارای مشکلات و سوال های بی جوابی است که نیازمند آپدیت‌های بیشتر است. طبق گفته‌های مربوط به این نسخه در پردازش GPU دارای سرعت بیشتری نسبت به نسخه‌های قدیمی است.

کتابخانه‌های opencv برای استفاده از الگوریتم های مربوط به Surf وGPU نیاز به کامپایل مجدد دارد. کامپایل کردن این کتابخانه نیازمند داشتن نرم افزارها و کامپایلرهای مختلف و ... است که ممکن است به مشکلات زیادی در طی کامپایل برخوردم. برای کامپایل از Visual Studio 2013 64 bit استفاده کردم.

در استفاده از کلاس stitching کتابخونه opencv بهیچ وجه پیشنهاد نمی شود. در استفاده از این کلاس جواب های بسیار دقیق و خوبی نتیجه گرفتم اما سرعت پردازش بسیار پایینی داشت در نهایت با انجام الگوریتم های بهینه سازی و موازی سازی اجرا و ... توانستم 15 تا عکس 630*470 را در 20 ثانیه به تصویر پانوراما تبدیل کنم که سرعت بسیار پایینی دارد و مناسب این مقاله نیست و مجبور شدم از ابتدا شروع کنم.

۳.۲ نتایج بدست آمده:

تمامی پردازش ها و مقایسه‌ها بر روی سیستمی با مشخصات زیر در مد دیباگ بدست آمده و تست شده است :

Msi core i7 **Cpu clock: 1948769Hz** **Ram : 8GB** **Graphic: Nvidia GT540**	**Visual studio 2013 64 bit** **Windows 7 64bit** **Opencv 2.411**

با استفاده از الگوریتم بالا با استفاده از Affine transformation و brisk descriptor و BrouteForce matcher الگوریتم بالا برای ویدیو 680*480 و خروجی تصویر موزاییک 2560*1200 اجرا شده است و الگوریتم دارای سرعت 21 فریم بر ثانیه بوده است (تصاویر زیر در مد دیباگ اجرا شده و سرعت کمتری نسبت به مد release دارند .

fps=10
frame=4

fps=11
frame=5

fps=11
frame=6

نتیجه خروجی:

شکل 18- نتیجه fastdetector با fps=19 با مدل prespective

شکل 19 - نتیجه orb feature detection با fps= 22 با مدل AFFINE

شکل 20 – نتیجه surf feature detection با fps = 13 با مدل AFFINE

۳.۳ مشکلات الگوریتم و راه حل های پیشنهادی در آینده :

در بدست آوردن ماتریس تبدیل اگر خطا ناچیز هم باشد وقتی ماتریس ها در هم ضرب می شوند خطاها در دراز مدت بیشتر خود را نشان می دهند و تبدیل به اعداد بزرگی می شوند. در آینده می توان برای بهبود این خطاها با استفاده از بردارهای حرکتی[1] و فیلترکالمن دقت را بالاتر برد یا از موزاییکی که قبلا بدست آمده استفاده کرد.

[1] Optical flow

فصل 5

۴ نتیجه گیری

با توجه به نتایج ارائه شده در قسمت 2-4 می‌توان گفت که الگوریتم انتخاب شده تا حد بسیار خوبی درست عمل کرده و نتیجه به دست آمده قابل قبول می‌باشد.

اما در جاهایی که دوباره قرار است فریمی را روی موزاییک قرار دهد که قبلا فریمی آنجا بوده است شاهد خطا هستیم که باید راهکارهایی برای این گونه موارد اندیشید مانند استفاده از فیلترکالمن یا زیاد کردن ویژگی های درست و تطبیق داده شده و حذف تطبیق‌های اشتباه. از دیگر کار هایی که می توان انجام داد blending و جبرای سازی[1] برای موزاییک ویدیو است که تصویری یکدست داشته باشیم البته اضافه کردن این موارد از سرعت پردازش می کاهد و ممکن است باعث ازدست رفتن بی درنگ بودن شود.

[1] compensation

۵ مراجع

1. **Adrian Kaehler, Gary Bradski.** *Learning OpenCV.* s.l. : O'Reilly Media, 2013.

2. **Gonzalez, Rafael C.** *Digital Image Processing using MATLAB.* 2003.

3. *Automatic Panoramic Image Stitching using Invariant Features.* **Matthew Brown, David G. Lowe.** s.l. : International Journal of Computer Vision, 2007.

4. *Probabilistic video stabilization using Kalman filtering and mosaicking.* **Andrew Litvin, Janusz Konrad, William C. Karl.** CA : Proceedings of SPIE Conference on Electronic Imaging, 2003.

5. [Online] http://www.stackoverflow.com.

6. [Online] http://www.opencv.org.

7. *Real-time construction and visualisation of drift-free video mosaics unconstrained camera motion.* Mateusz Brzeszcz, Toby P. Breckon : The Journal of Engineering, 2015.

8. *Real-time Spherical Mosaicing Using Whole Image Alignment.* **Steven Lovegrove, Andrew Davison.** s.l. : ECCV 2010, 2010.

9. *Near-Optimal Mosaic Selection for Rotating and Zooming Video Cameras.* **Nazim Ashraf, Imran N. Junejo, and Hassan Foroosh.** 2007 : Computer Vision – ACCV.

10. **Dr.MubarakShah.** *Computer Vision Class Videos.* UCF.

11. *Real-time Mosaicing from Unconstrained Video Imagery.* **Mateusz Brzeszcz, Toby P. Breckon, Ken Wahren.** s.l. : 26th International Conference on Unmanned Air Vehicle Systems, 2011.

UNIVERSITY OF TEHRAN

College of Engineering

School of Electrical and Computer Engineering

Title:

Real-time Video Mosaicing for Fast Camera Motions

By:
Ali Jahani Amiri

Supervisor:
Dr. Hadi Moradi

A thesis submitted to the Graduate Studies Office
In partial fulfillment of the requirements for
The degree of B.Sc
In
Electrical Engineering
Control department

September 2015