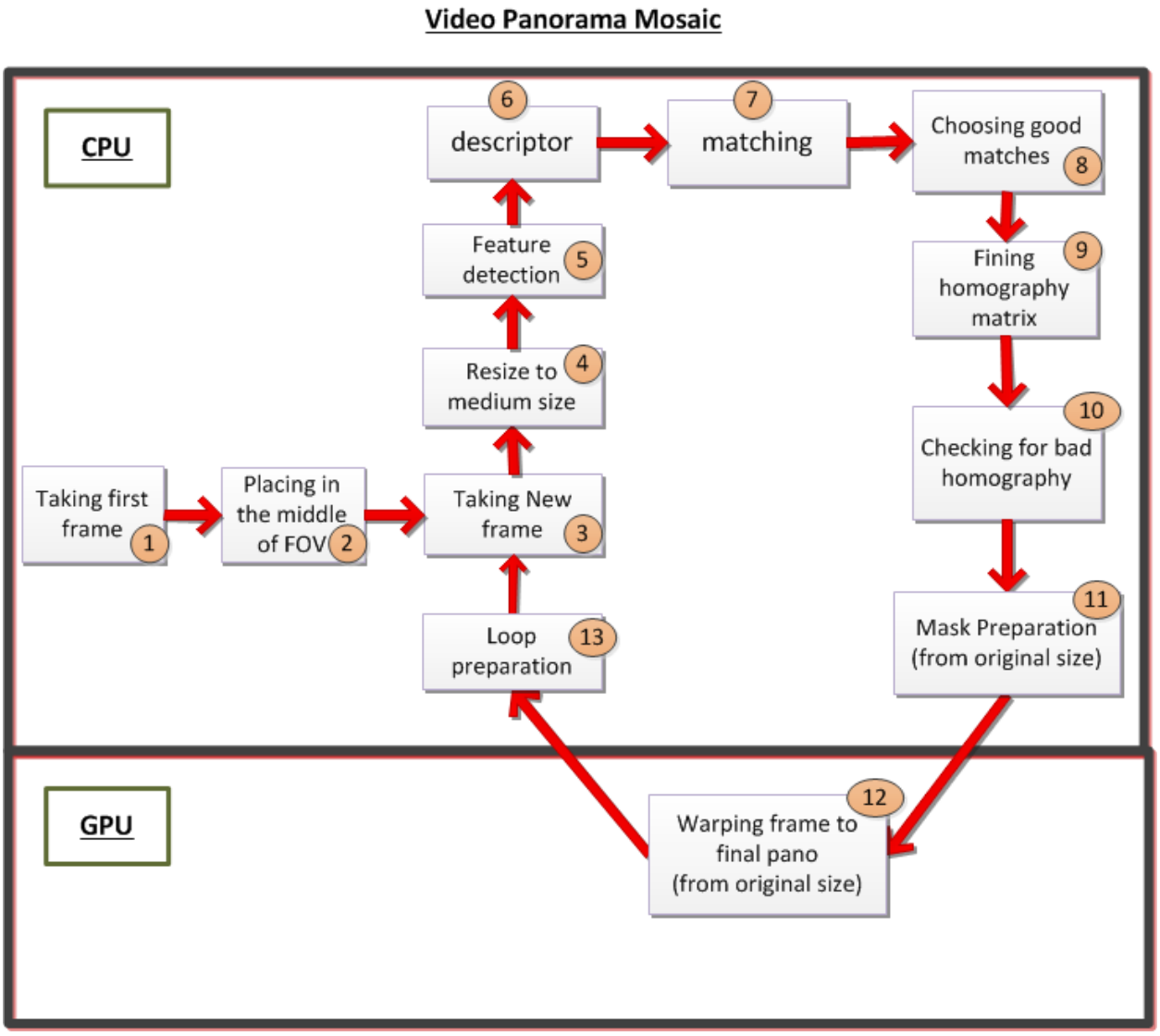
Video Mosaic panorama

The pipeline below is the main process which I have implemented:

For better result we can use GPU for whole process.



Note: All timing comparisons are in Debug mode. They will run much faster in Release mode. Results for Release Mode are at the end.

Here is some explanation of the parts:

1) **Taking first frame :**

We took the first frame and gray scale it then find its features and descriptor.

2) **Placing in the middle of FOV :**

Then we place the first frame into the middle of our field of view (FOV).

3) **Taking new frame**: we took new frame and make it gray scale for further process.

4) **Resize(medium size) :**

Our frames could have high resolution. Higher resolution mostly has higher features leads to more process time. We can higher threshold but it's not good at all. Better way is too resize them and make them smaller. It could lead to robustness in comparison with threshold changing.

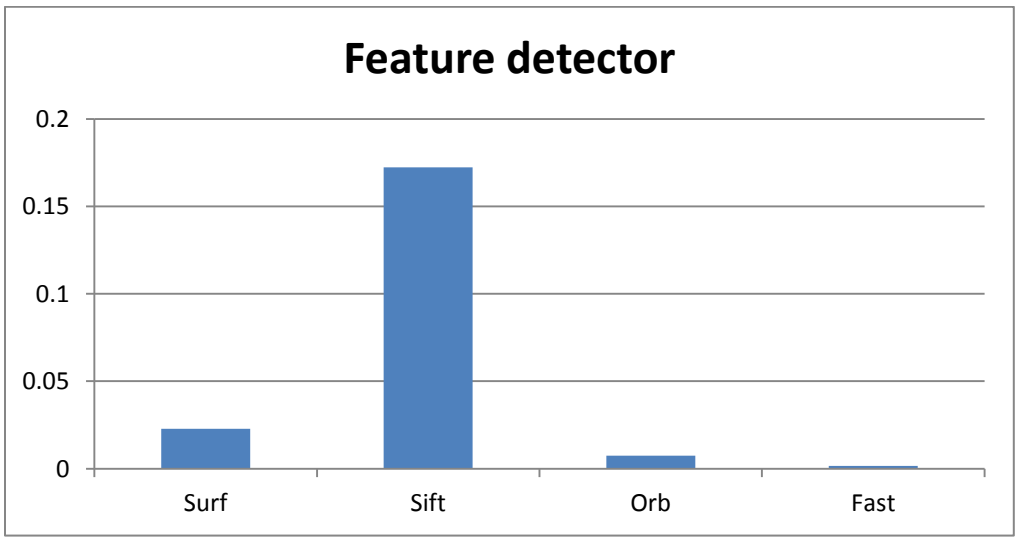
5) **Feature detection :**

About 200 features are enough for our process we do not need more because time is an important factor.

There are some feature detections already defined in opencv library. Some are faster but not good

For frame 48:

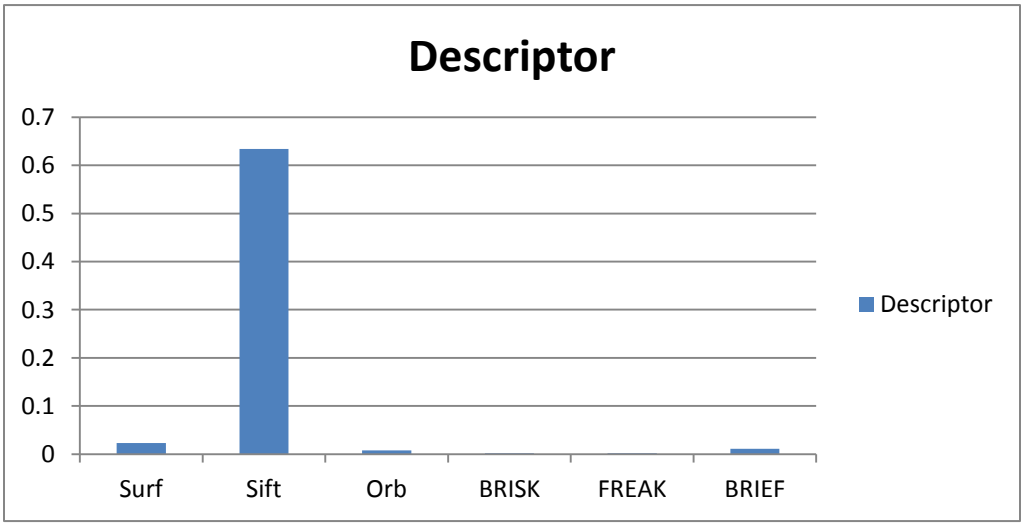
	Surf	Sift	Orb	Fast
Threshold	130	200	200	16
feature	105	101	118	178
time	0.0228395	0.172254	0.0074782	0.00161182



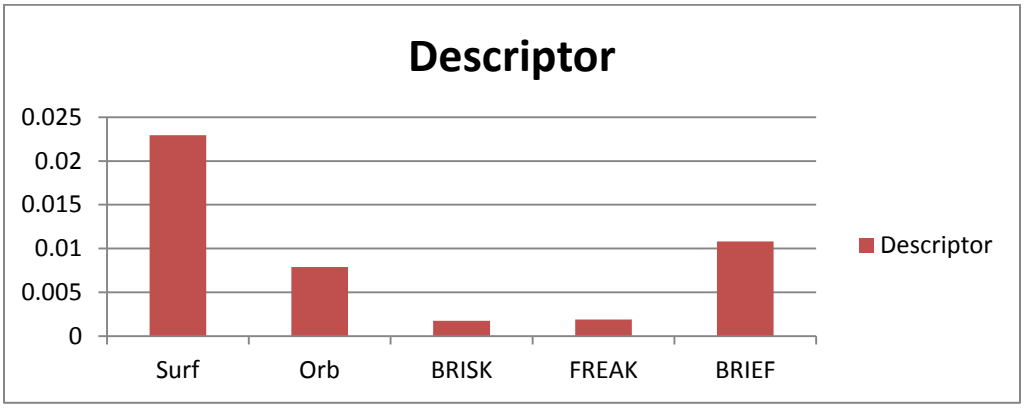
6) **Descriptor:** after finding feature points in the frame we need a descriptor in order to match them.

Here are some descriptors already defined in the OpenCV library:

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



Here is result without sift to be more accurate:



7) **Matching**: There is some matching I used BruteForce matcher. There is some Flannbased matcher which I didn't used till now.

Matcher (frame 48)	BruteForce matcher	FlannBased matcher
time	0.00383481	0.150021

8) **Choosing good matches**: Its normal if we have some mismatches, so I picked only matches which their distance is less than 3*min distance of whole matches. This will cause mostly robustness but sometimes it will cause more errors too

9) **Finding homography matrix:** The main part is to find a 3*3 Perspective Projection Matrix or a 3*2 Affine Matrix between frame (n-1) and frame (n) using RANSAC[1] method.

For placing these images together we need to multiply the transition matrices together and find transition matrix between frame n and frame one.

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

Then we need an offset to move the image to the center of the FOV.by pre multiplying this matrix:

[1] Random sample consensus

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

This offset matrix only needs to be premultiplied at first.

H2to1= H_{offset}*H2to1

H3to1= H3to2*H2to1

…

Hnto1= H (nton-1) *…*H2to1

	Affine	Perspective
TIME	0.0135252	0.0137963

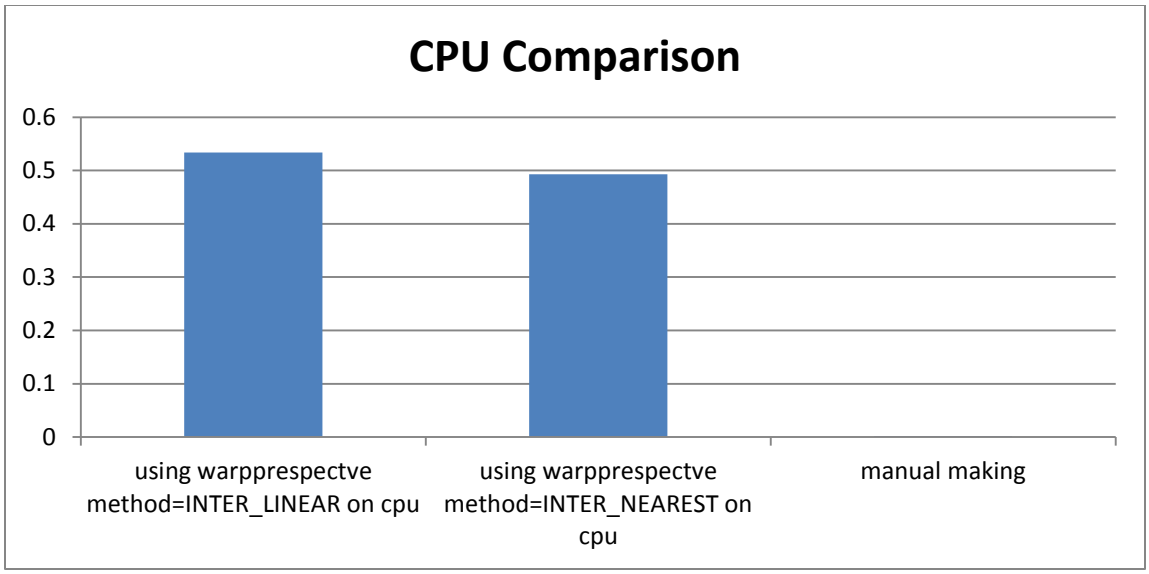
10) Checking for bad homography: Our data is video file so the homography matrix is not changing suddenly if there is sudden It is error and can cause to output failure, so I calculate norm 2 of new homography matrix minus last homography matrix , and then I calculated average of these norms till n-1 frame in each frame. If norm in frame n is more than average norms till frame n-1, it is an error then I used frame n-1 homography for frame n, It will cause some small errors but better than losing the whole output, if the norm is less than 10*average norm then it's ok and I update the Average with the new norm achieved.
The threshold is optional based on the data

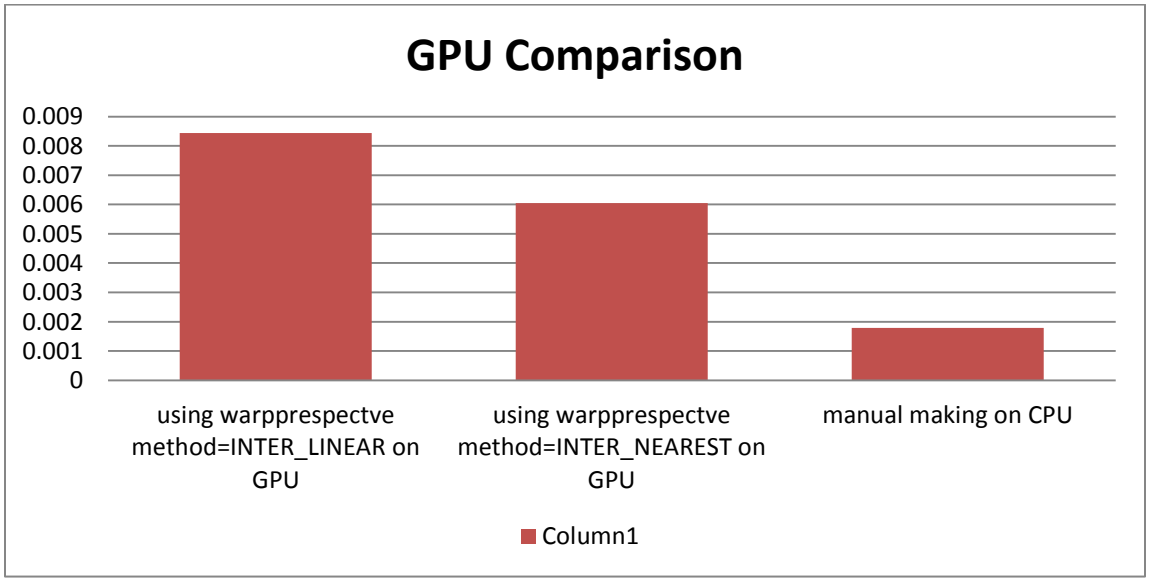
11) Mask preparation:

We need a region of interest for the final image to copy the wrapped image into it. One way for making mask is applying the homogaphy matrix to a white filled rectangle same size as frame (we apply homography matrix to a white image same size to our frame). The bad thing is if our FOV has high resolution (usually it is) it will take so much time for CPU (for me about **0.533502** second for a 3840*1200 image).

Here is what I did in a creative way:
I made a black image same size as final panorama then I calculated the corners of the warped frame using `perspectiveTransform` function, then fill the shape with white color with `fillConvexPoly` function .So by making the mask manually it took 0.00178886 second for a 3840*1200 image. This way is even better than using Gpu. Gpu will take 0.00604753 second.

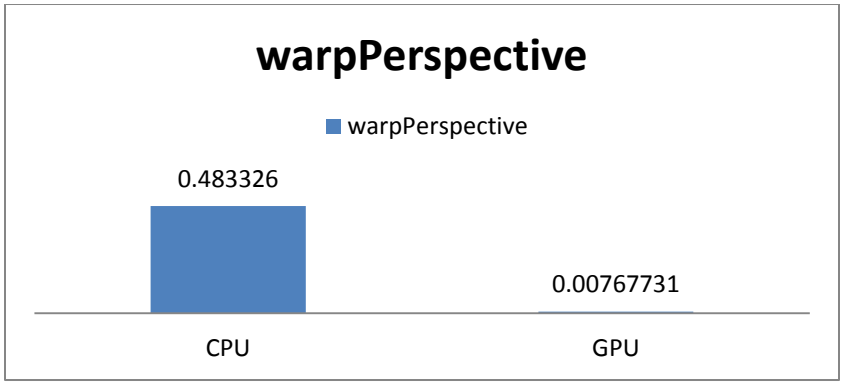
	WarpPrespective Inter_Linear	WarpPrespective Inter_Nearest	My method
CPU	0.533502	0.493054	0.00178886
GPU	0.00843678	0.00604753	



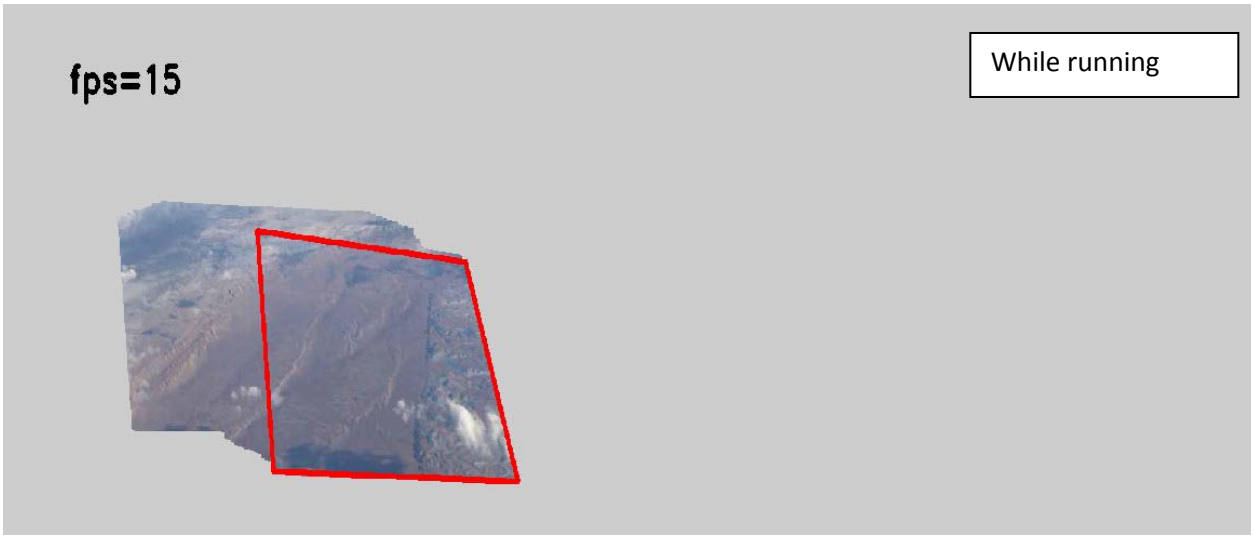


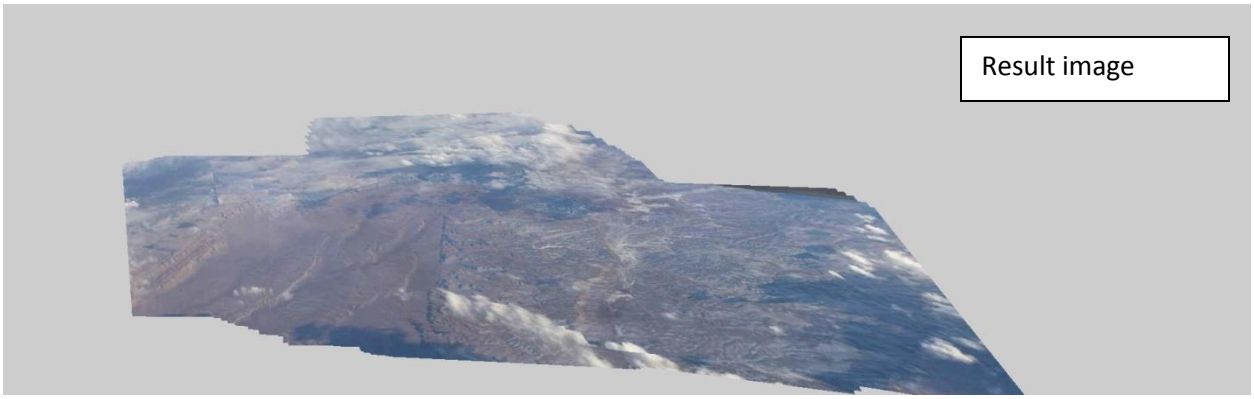
For example the mask of frame 50 is image below.

12) Warping image to final pano: now we need to apply homography matrix to the image, then copy it using mask to our final image. The time consuming for CPU depends on the output resolution .for a 3840*1200 resolution 0.483326 sec needed. So I used GPU for this part. Time for GPU is 0.00767731.



13) Loop preparation: At the end of loop we need to copy current frames , its features , descriptor and some other things into last frame so no need to calculate these again. This process took 6.36312*10^-5



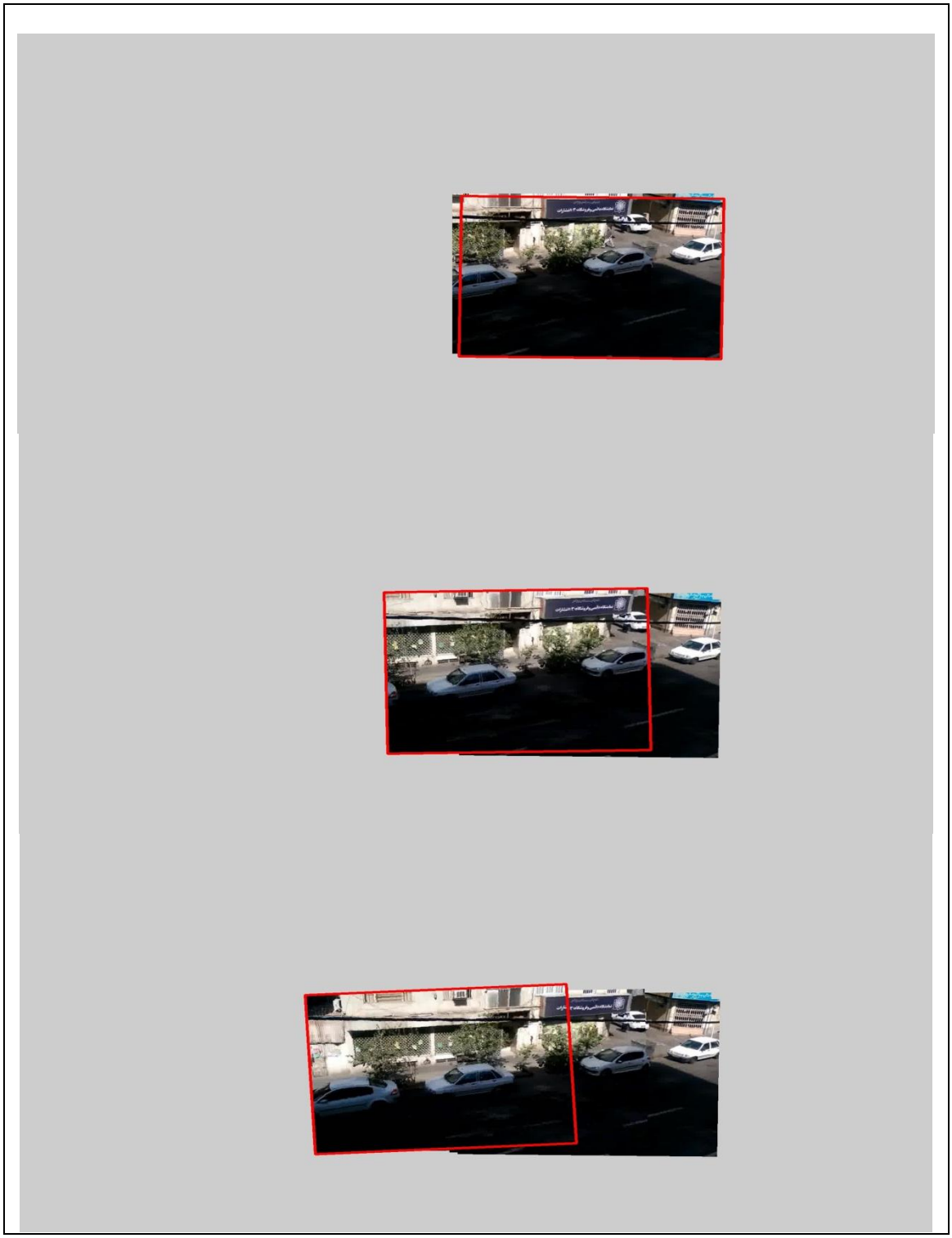


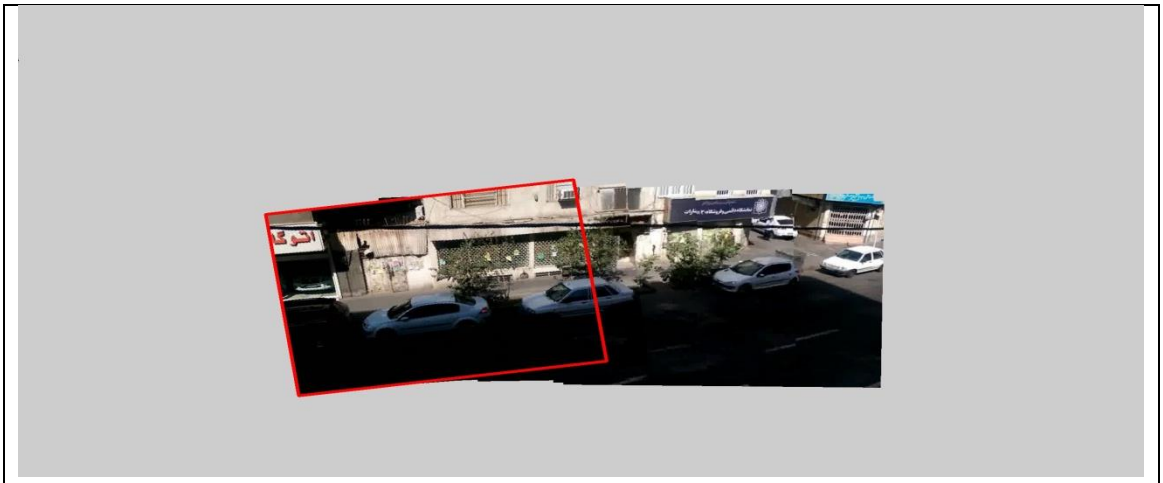
The video has 320*240 pixels. Field of view has 1920*600 pixels

Doing with:

- FastFeatureDetector (13)
- ORB DESCRIPTOR
- BFMATCHER (METHOD=NORM_HAMMING, CROSS MATCHING=TRUE)

Another Result:





Result:

Exact results in Release mode Tests:

	Frames	Time	FPS
With Consideration of Webcam/hard delays	664	32.7757	**20.2**
Without Consideration of Webcam/hard delays	664	29.693	**22.3**

Result :

Input resolution is 640*480 and Output resolution is 2560*1920

Doing with:

- FastFeatureDetector with 300 features
- BRISK DESCRIPTOR
- BFMATCHER (METHOD=NORM_L2, CROSS MATCHING=TRUE)

Tested on:

- MSI Core I7
- CPU frequency=1.94873 MHZ
- Nvidia GT540
- OpenCv 2.411, Visual Studio2013, Windows7, 64 bit

Reference:

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

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

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

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

9. *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.

10.[Online] http://6.869.csail.mit.edu/fa12/lectures/lecture13ransac/lecture13ransac.pdf