Project Report on Computer Vision : Spring 2018

Team : LazyLasers
Subhasis Chand, Raghu Ram, Shrey Agarwal

April 14, 2018

This report is submitted as part of Computer Vision Course, Spring 2018. The objective is to implement the paper "Speeding up Homography Estimation in Mobile devices" and check its validity in various use cases. We have also developed an application which converts a video to a panoramic scene. So the over all tasks completed in this project are

- Implementation of Geometric constraints for Speeding up Homography in OpenCV-3.1.0
- Time comparison with and without Geometric Constraints using SIFT and SURF features
- Implementation of above two points in RaspberryPi 3B and check timings
- Panorama creation using Homography
- Panoramic view generation from a video

1 Introduction to the idea

The homography matrix is mainly estimated between the two image pairs in order to find out the relative orientation between both the images.It is also used in many applications like bot navigation,Finding the camera movements etc.,Single homographies are usually estimated finding the correspondences of a set of discriminant features using Random Sampling method.The scheme followed in our implementation is SIFT and SURF for feature description followed by RANSAC for Homography estimation.It iteratively takes the random subsets from the set of correspondences and fits a model to them.The quality of model is given by the number of inliers.An inlier here is a sample whose distance from the model is below a given threshold.

Geometric constraint in Homography estimation:
The set of points in the plane are visible only when two necessary conditions hold:
i)Points must be in front of the camera
ii)The plane containing the points must face towards the camera.

By considering that the above constraints are satisfied we can proceed towards model estimation by RANSAC which saves a lot of computational effort on the processor.This can be mathematically explained as.If I_0 and I_1 be two camera images of the plane pi.P_0 and P_1 are the visible key-points on image I_0 and I_1.let (p_0,p_1) be a tuple that maps an element each from P_0 to P_1 by a tentative correspondence.

- Let $p_i \in P_i$ be the visible point in Image I_i.Since it is visible, there exists a positive scale λ_{p_i}, such that $p_i = \lambda_{p_i} P_i$, $\lambda_{p_i} > 0$. Where P_i belongs to R^3 is the location of the projected point p_i in the reference frame of the camera i.

- The plane induced by correspondences should be visible in both the images I_o and I_1. That is both the camera centers must be located on the same side of the plane. This is given by the following conditions $n_0^T C_0 + d_0$ is similar upto positive scale $n_1^T C_1 + d_1$

So the geometric constraint obtained is $(b_0 \times c_0)^T a_0$ **is similar up to a positive scale** $(b_1 \times c_1)^T a_1$.

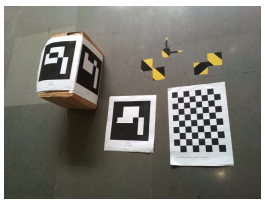
2 Implement Geometric constraints for Speeding up Homography in OpenCV-3.1.0

We implemented the paper "Speeding up Homography Estimation in Mobile devices" on OpenCV-3.1.0. The paper suggests to implement the check on geometric constraints in CvModelEstimator2::runRANSAC in OpenCV2.4, however in OpenCV-3.1.0 it has been moved to RANSACPointRegistrator− >run(). However after our implementation is completed we noticed that this feature has been already included in OpenCV-3.1.0 in HomographyEstimatorCallback− > checkSubset(). So we disabled the original implementation to check our results. However surprisingly enough the original implementation did not help reducing time in homography estimation. So we decided to test the timings without any constraints, with original implementation and with our implementation. The code implementation is provided in "ptsetreg.cpp" which is to be placed inside OpenCV-3.1.0/modules/calib3d/src/ while building OpenCV. The code changes can be found under the tag "Subhasis" in the above file.

3 Time Comparisions with and without Geometric Constraints using both SIFT and SURF features

After the changes in OpenCV internal code we tested its effects on homography estimation. We tested it on three pairs of images. For each pair we recorded the time taken for homography estimation and complete feature matching process using SIFT and SURF in three different scenarios e.g. without any geometric constraints, with original implementation in OpenCV and with our implementation. The code can be found in "homography_ransac_for_comparision.py". This can be easily observed from the table that our implementation reduces the time for estimation of homography by half. So we reached the goal of Speeding of Homography Estimation. The time comparison is listed on Table.1.

(a) floor 1



(b) floor 2

(c) clg 1

(d) clg 2

(e) hill 1

(f) hill 2

Figure 1: Figures used for generating the time comparision table

Table 1: Comparison of time (in mSec) for Homo graphy Estimation and Feature Matching using SIFT and SURF Features under three different constraints

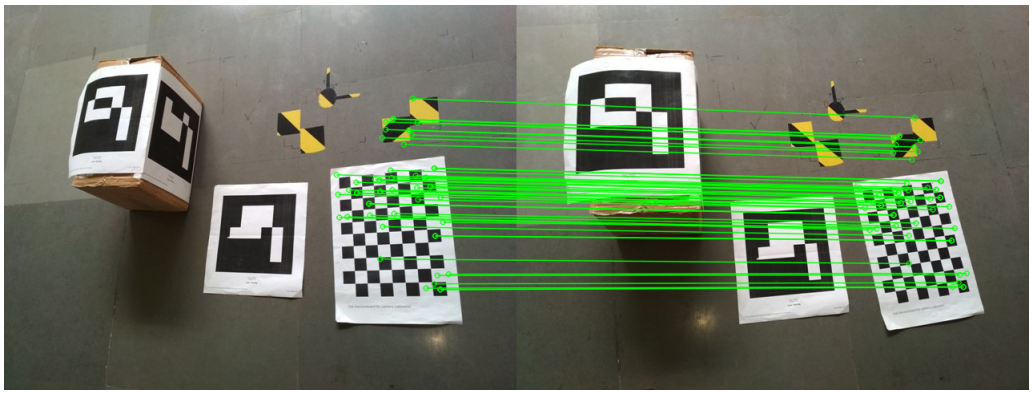
		SIFT			SURF		
		hill	floor	clg	hill	floor	clg
Without any constraints	Time for Homography	43.5	32.6	40.0	25.7	34.9	44.0
	Time for feature matching	1432.9	465.6	937.2	272.3	480.8	420.5
With Original Implementation	Time for Homography	44.6	33.2	41.3	26.0	35.0	45.3
	Time for feature matching	1434.5	467.4	937.1	269.1	475.0	414.3
With our Implementation	**Time for Homography**	**20.2**	**18.9**	**20.0**	**11.2**	**20.6**	**22.6**
	Time for feature matching	1412.8	459.2	918.7	254.1	462.7	394.7

4 Implementation on RaspberryPi

We installed OpenCV-3.1.0 on RaspberryPi 3B which is a SoC with 1.2GHz quad core processor and 1GB of ram. The time comparison was done with original geometric constraint implementation and our code implementation. It was observed that even if after speeding the homography, the feature matching takes a significant amount of time on RaspberryPi. Refer to Table.2 for the time comparison. An image of the Pi set up is provided in Figure.3.

Table 2: Time comparison on RaspberryPi

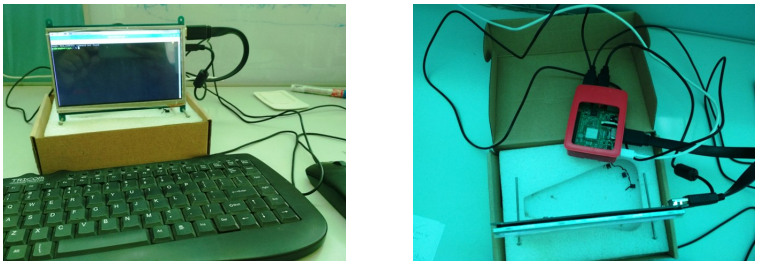
		SIFT			SURF		
		hill	floor	clg	hill	floor	clg
With Original Implementation	Time for Homography	1427	744.3	1180	747.6	860.7	1460.2
	Time for feature matching	42751	17202	28740	4799	3816	8670
With our Implementation	**Time for Homography**	**592**	**524.1**	**628.9**	**249.4**	**665.2**	**764**
	Time for feature matching	41675	16963	28250	4233	3662	7915



(a) Feature matching using SIFT

(b) Feature matching using SURF

Figure 2: Feature correspondence



(a) (b)

Figure 3: RaspberryPi Setup for Homography Estimation and Feature Matching

5 Panorama Creation using Homography

After successful implementation of Homography and Feature Matching we tried to generate panoromic images from pairs of images. "homography_ransac.py" was used to generate panoromic view from two images that was used for timing comparison. Refer to Figure.1 to see the original images. After stitching the below panoramic images were generated in Figure.4.

Then we took multiple images horizontally and stitched them together using homography matrix. Let the homography matrix between image one and image two is H_{12} and the homography matrix between image two and image three is H_{23}. Then the homography matrix between image one and

(a) Panorama generated using SIFT

(b) Generated using SIFT

(c) Generated using SURF

Figure 4: Stitching two images to generate a panoramic view and observe the difference between stitching using SIFT and SURF in (c)

image three is $H_{13} = H_{12}.H_{23}$ and so on. Thus panoramic view of our campus was generated which is shown in Figure.5.

Figure 5: Multi Image Panoramic view of our campus

6 Panoramic view generation from video

After some serious brain storming on further possibilities we decided to go for vertical stitching along with horizontal stitching to create a bigger panoramic image in all directions. We took nine images of Himalaya building in horizontal and vertical direction and stitched them to create one image (Figure 6). The resultant image is stitched properly, however it seems skewed due to the use of plane projection while wrapping.

Figure 6: Multi Image Panorama generation in horizontal and vertical directions

We thought of creating an application using RaspberryPi to create panoramic images in real time. But we didn't have portable battery or any power source to use the raspberryPi out side the lab. So instead we took a video of our Himalaya Building and created a panoramic image using homography transform. If the video source is replaced with a camera stream in real time, it will generate the same type of panorama image. Current mobile applications can build only horizontal panoramic image, but our program is able to handle the movement of camera in all directions. The video which is used to generate the panorama is provided in the resources directory of the git repository. The final result is shown in Figure 7. In this figure the first frame of the video can be easily observed in the right bottom corner. Then the camera moved up and then moved towards left. We captured image frames at regular interval to create the final image.

Figure 7: Panoramic view generation from a video