**SIFT IMAGE STITCHING OF INFINITE INPUT IMAGES**

By John Hacker

**INTRODUCTION**

When taking a picture, people usually look at the preview of the image before pressing the button to finalize the captured picture. If upon looking at the preview, the desired subject is not completely in view or otherwise unpleasantly framed, the camera-holder will move the camera until an acceptable view of the subject is achieved. In some instances, such a view may not be possible. Imagine the case where the subject is so large that getting it seen within the limits of the camera’s field of view degrades the resolution of the subject or it becomes out of focus. Additionally, if the subject is particularly large, there may not be a space where the camera-holder can physically stand to get the entire subject in view, caused by a wall or other such obstacle.

A solution to such a problem includes taking multiple pictures of the subject and using a computer software program to stitch the images together. This would require finding features between two images and combining the images at the points where they overlap. This paper will discuss how I used the Scale-Invariant Feature Transform (SIFT) to find and match key features to stitch an unlimited number of images together.

There are several examples where image stitching is already being used by similar robotic systems. One famous instance has been done on smartphones for years: panoramic photos. Many modern smartphones have the capability to allow their users to move their phones while capturing images at a designated interval to string together an interpreted panorama. Also, some Android phones, today, can create a photosphere, where the user can combine pictures of their surroundings and create a three-dimensional experience on their smartphone. Another example is digital maps. Large images of the globe, such as Google Earth, are created by stitching together images taken by satellites over a long period of time.

**METHOD**

SIFT is a method created by David G. Lowe to extract features of an image using edge-orientation histograms around Harris corners. The main benefit of using SIFT over just finding Harris corners or other such keypoint-finders is that the resulting keypoints are invariant to both rotation and scale. This means that two corners may be matched despite being different sizes and/or orientations. Additionally, SIFT can efficiently find many robust features across a wide spectrum of images. Descriptors are associated with each final keypoint to allow for better matching. The four steps of SIFT are scale-space extrema detection, keypoint localization, orientation assignment, and keypoint description.

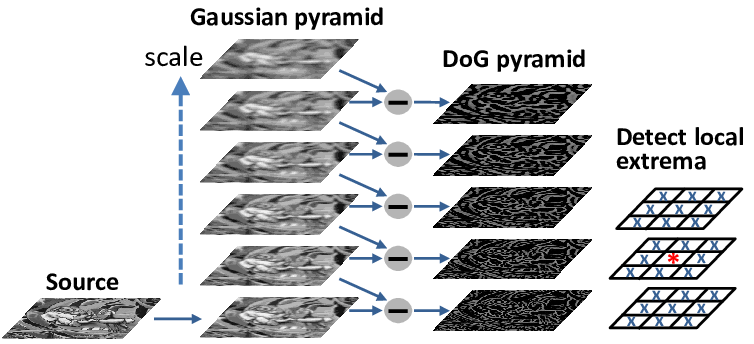


Figure 1: Keypoint Detection and Localization

The first step of SIFT is scale-space extrema detection. This seeks to find points image which can be found over multiple scales and image locations. These points are the edges and corners that best describe the image. As demonstrated in Figure 1, the first step is to create a Gaussian pyramid so that keypoints can be searched for at several intervals of the scale. Each level up the pyramid reduces the size and smooths the image. Adjacent levels of the pyramid are subtracted from each other to form a Difference of Gaussian (DoG), which approximates a Laplacian of Gaussian kernel. The DoG pyramid highlights local extrema in the image where edges and corners exist.

Secondly, keypoint localization is required to filter out bad detected edges. This part of the algorithm will look to reject flat edges with low contrast and poorly localized edges. After extracting maxima and minima from the DoG pyramid, compare each extracted point to its eight neighbors in the current image and nine neighbors in all the scales above and below it in the pyramid. Keypoints are disregarded if they do not fit nearby data.

The next step is the orientation assignment. To do this, a histogram of gradient directions is created within a region around the keypoint at a selected scale. Histogram values are weighted by the Gaussian function so that the most central values are the largest influencers. The orientation of each keypoint will be reassigned according to the maximum value in the gradient histogram around it.

Now that the position, scale, and orientation of each keypoint is known, it is time for the final step to SIFT: keypoint description. First, a normalized region for the keypoints must be created, which helps make SIFT more invariant to contrast and illumination. To do this, the window around each keypoint needs to be rotated to standard orientation and then scale the window based on the level of scale where the keypoint was found. Then, gradient magnitude and orientation of point in the window is calculated and weighted by the Gaussian function. Finally, a set of vectors can be formed to describe how the points look from different angles.

My script, written in Python 3.6.8, uses an OpenCV Contrib (version 3.4.2.16) function to perform the SIFT algorithm, which returns the keypoints and descriptors. Then, I use the descriptors to find the homograph of the keypoints via the RANSAC function. With the resulting homograph matrix, I warp the perspective of the first (right) image to match the second (left) image. Then, the warped right image is appended to the left image. Thus, a stitched image of the two pictures is created.

For added utility and complexity, I developed a solution that stitches together more than two images. This is accomplished by iteratively and compounding stitching images together. In other words, the first two images are stitched together; then that stitched image is stitched with the third picture, and so on. These images need to be in a separate directory, which is designated by a customizable variable within the code. A plot showing each step of the compounding stitching is shown along with each stitched image being saved to another directory.

**DISCUSSION**

The operations of this system can be described as a mild success. It can perform the desired task of stitched together more than two images where overlap occurs. Figure 2 shows an example of stitching together four images of one wall of my bedroom. The results are adequately successful at combining the first two images in the sequence with only minor discrepancies segments lining up. Differences in lighting cause a pretty clear distinction between the two halves of the image, but the common overlap is well identified and stitched. It does a great job of warping the right image to match the perspective of the left image, which I find particularly impressive. This is clear evidence that the descriptors are doing a good job of describing the orientation and scale of keypoints.

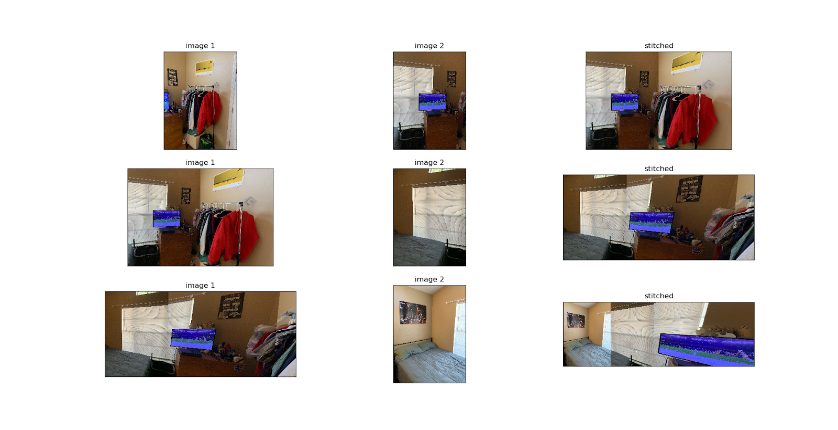


Figure 2: Example Output of Stitching Three Images

Some more problems arise when the third image is stitched on--shown in the second row of Figure 2. For starters, a fairly large portion of the right image ends up being cropped out to match the orientation and size of the new picture. Additionally, the television screen is beginning to be stretched because the perspective warp believes that is how it would look given its descriptors. Again, differences in illumination are clear in the stitched image, but the features are still well identified.

Finally, the last row of Figure 2 shows some major issues with my system for stitching infinite input images. Another large portion of the right image is cropped, and the television screen is stretched to a comedic level. The bottom of the window and bed are well lined up in this stitching, but the top of the window is clearly jagged across the border.

With more time to evaluate, there are some examples that show more flaws and successes in the system. A couple of such flaws are odd-looking human faces where stitching occurred down the middle of them and warped perspectives resulting in an image smaller than the defined dimensions.

**CONCLUSION**

There are several ideas that could be implemented to improve the results of my robotic system. The easiest is to make the input data more appropriate for the function of the system. The example shown in Figure 2 was made with input from taking pictures while standing in a single spot and rotating to capture the entire wall. The way this system works, images on the far right will be warped once for each image to the left of it. This means that the rightmost image used for Figure 2 was warped three times to match the perspective of each image to its left, while the leftmost image was never warped. This causes a wicked-looking stitched result. Also, illumination, contrast, and focus in the images should be more consistent for the sake of making the stitched images more consistent.

To change the script itself, one idea to centered on the middle image, instead of the leftmost image. This should make for more appealing results. Also, the script requires the images to be ordered alphabetically from right to left. One idea could be to make the script more robust to images being ordered in different ways. Also, the script requires a specific set of versions of Python, OpenCV, and OpenCV Contrib. In the future, the script could be modified to handle different versions of these software packages.