IMAGE-GUIDED TOURS: FAST-APPROXIMATED SIFT WITH U-SURF FEATURES

Eric Chu, Erin Hsu, Sandy Yu
Department of Electrical Engineering
Stanford University
{echu508, erinhsu, snowy}@stanford.edu

Abstract **– In this paper, a feature-detecting algorithm to identify paintings from mobile phone camera images is presented. The algorithm is capable of identifying paintings from noisy camera-phone images. Even with typical blurring, rotation, and a small amount of perspective distortion that would occur in a controlled museum setting, by training on a set of standard painting images, the paintings can be correctly identified. With calculation-intensive portions of the code in C, the algorithm runs efficiently as well as accurately. All 99 test images were detected correctly, with an average runtime of 0.9228 seconds on SCIEN machines.**

I. INTRODUCTION

The prevalence of cellular phones with built-in cameras has stimulated a growing interest in the application of image recognition on mobile phones. Various applications are already in use, such as visual and bar code scanning technologies [1], image-based search engines [2], location disclosing photo recognition [3], and many more that are currently in development.

The EE368 final project this quarter is the application of image recognition on mobile phones in a museum setting. Given a set of training images, consisting of several perspectives of 33 paintings in a gallery, and an image to be matched (which shall be referred to as the test image), the name of the painting is to be returned. In a real-world scenario, this kind of system could replace the numbered audio commentaries prevalent in many museums today and give visitors additional information via the Internet or another source.

The training images consist of one painting without its frame, and some background, but it is not a difficult task to seek out the painting, since the background is a controlled white environment. Cell phone camera images tend to be poorer than regular cameras, due to low quality lenses, an increased likelihood of camera shake, and a lack of flash on many models. Noise, motion blur, low contrast, and poor lighting often plague these images. For this project, the museum provides consistent lighting in each image, so the main difficulties are noise, blur, perspective, and scaling.

Several algorithms have been established for feature detection, including Scale-Invariant Feature Detection (SIFT) [4][5][6] and Speeded Up Robust Features (SURF) [7]. Both algorithms create an image pyramid, filter the pyramids appropriately to detect points of interest, quantify the features, then compare with features of another image to find matches. The algorithm described is based on this same structure but deviates from both SIFT and SURF in different areas.

II. ALGORITHM DESIGN

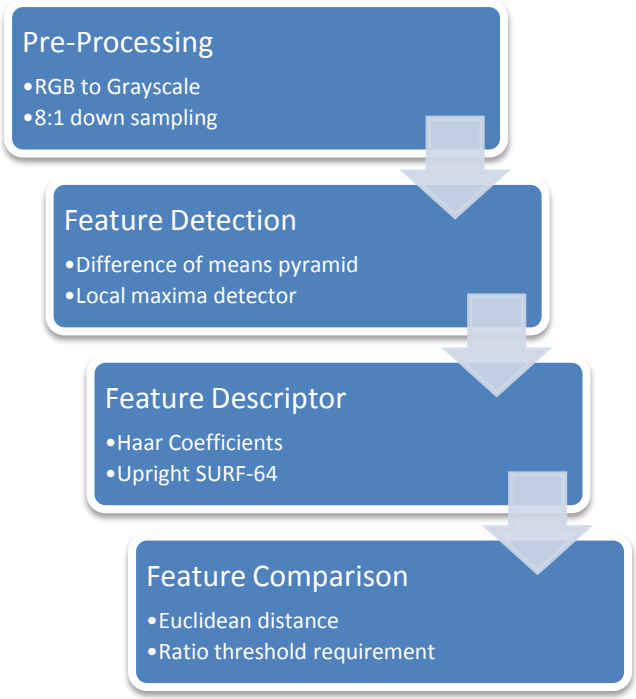

Figure 1: Algorithm flowchart

The algorithm applied to this project was developed based on studying and experimenting with SIFT and SURF processes as well as other topics from course material. After much testing, the final algorithm developed utilizes a combination of methods that are best suited to match paintings in a relatively consistent environment, while also optimizing for speed. The major steps of the algorithm involve pre-processing the test images, finding relevant features, quantifying features with local descriptors, and comparing these features with the training paintings to correctly identify the test images, as shown in Figure 1.

A. Pre-Processing

The first step in matching two images is often to balance the color, brightness, or contrast. Color balancing was not implemented because of the well-controlled lighting in the environment, and also because the selected feature descriptor is invariant to changes in intensity. Because of the standard museum setting and lighting, changes in brightness between images were not significant, so that was also not used. Finally, contrast-balancing was also applied at one point but also did not considerably affect accuracy.

Therefore, the pre-processing in this algorithm consisted only of converting the RGB images to intensity (grayscale) images and an 8:1 down sampling to speed up the runtime of the algorithm. Color information was discarded because of its unreliability in most computer vision settings, and while the LAB colorspace could have been used to simulate color constancy in human vision, the increased runtime was unappealing, and the same task could be conducted on only the intensity image. Because the input images are of extremely high resolution, down sampling is used to decrease the number of pixels used to construct the image pyramids and thereby provide a significant decrease in processing time. Prior to down sampling, a low pass averaging filter was applied to the grayscale image in order to prevent aliasing when sampling, as demonstrated in Figure 2. This filter was chosen to precisely zero-out the zoneplate frequencies at $\pi/8$ and significantly attenuate higher frequencies.

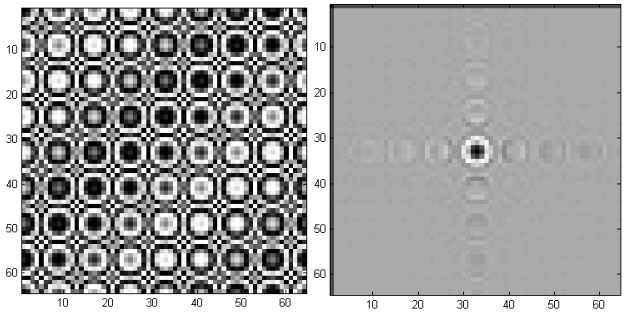

Figure 2: The image on the left shows the aliasing that occurs when down sampling 8:1 with no filter. The image on the right is the result with a filter applied.

B. Feature Detection

Once the images have been pre-processed, the next step is to find relevant features to use to match paintings. These features are often corners or edges in the image but should be invariant to scale and stable, meaning that these features should be consistently detected even in a different setting. Because the Harris corner detector is not scale invariant, it was not a good candidate for feature detection. Rather, SIFT and SURF features, which are scale-invariant, were decided on as the features of interest. SIFT and SURF algorithms have slightly different ways of detecting features. SIFT builds an image pyramid, filtering each layer with Gaussians of increasing sigma values and taking the difference. This difference of Gaussians approximates a Laplacian pyramid, as discussed in class and in [5]. Since image pyramids are used in this multi-resolution image, the Gaussians of different scale can be made using a constant filter size.

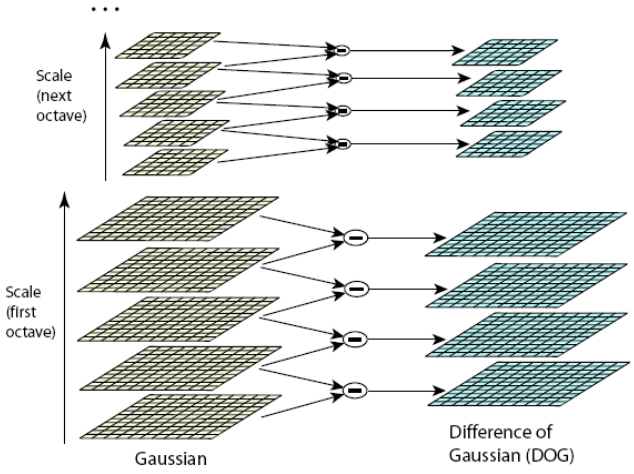

Figure 3: Visualization of Difference of Means concept
Source: (Lowe, Distinctive Image Features from Scale-Invariant Keypoints, 2004)

On the other hand, SURF creates an "image pyramid" without 2:1 down sampling for higher levels in the pyramid, resulting in images of the same resolution. Hence, it can be better described as an image "stack" instead of a pyramid. SURF then filters

the stack using a box filter approximation of second-order Gaussian partial derivatives. To filter higher layers, the filter sizes are successively increased. This method is possible due to the use of integral images [8] in the calculations of the partial derivatives of varying scale, since integral images allow the computation of rectangular box filters in near constant time.

The algorithm implemented uses the SURF method of an image stack with full resolution on each level. However, instead of a Gaussian second-order partial derivative filter, a simple box filter is applied to compute mean images. Again, to improve algorithm speed, integral images [8] are used with box filters of different sizes to calculate the means of each image in the pyramid as in [9]. These filtered images are then subtracted from each other and normalized with the factor presented in [9]. Thus, a difference of means pyramid, which approximates a difference of Gaussians pyramid is built. This concept of taking the difference of images is illustrated in Figure 3. Note that this algorithm also does not perform the 2:1 down sampling at higher levels, because it uses the integral image to compute the averages.

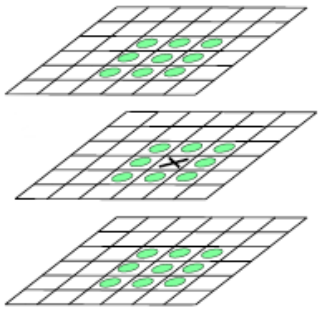

Figure 4: Comparison of point with 26 neighbors in 3x3 area
Source: (Lowe, Distinctive Image Features from Scale-Invariant Keypoints, 2004)

Once the stack has been built, a local maxima and minima detector is applied to identify keypoints by comparing all 26 neighboring elements in a 3x3 matrix, as indicated in Figure 4. A pixel is compared with its 8 neighbors and the 9 in the level above and below it; only the largest or smallest values are retained. Boundary cases are ignored, and "edge-like" features as defined by David Lowe [5] are also eliminated. This is done by taking differences of neighboring sample points to compute a 2x2 Hessian matrix to check that the ratio of principal curvatures is below some threshold *r*:

$$\mathbf{H} = \begin{bmatrix} D_{xx} & D_{xy} \\ D_{xy} & D_{yy} \end{bmatrix} \quad \frac{\mathrm{Tr}(\mathbf{H})^2}{\mathrm{Det}(\mathbf{H})} < \frac{(r+1)^2}{r}$$

Each feature location and radius is passed on to the next segment of the algorithm.

Figure 5 shows a rough comparison of the features detected by this algorithm (in green) which approximates SIFT features compared to the actual features detected by an implementation of SIFT (in red) [6]. Note that while this algorithm reliably detects features of small scale (or size) it does not find relevant larger features as well as SIFT.

Figure 5: Comparison of approximated SIFT features (green) to actual SIFT features (red).

C. Quantifying Feature Descriptors

Once the points of interest are identified, they must each have distinct quantitative representations for matching purposes. SIFT and SURF both have slightly varying methods to achieve this goal. SIFT breaks apart a window around a feature into 4x4 sub-blocks and calculates an orientation histogram. The magnitudes of vectors weight 8 bins of orientations, and the sum within each bin is the feature descriptor. A faster version of SIFT uses integral histograms to approximate the orientation histograms, speeding up calculations [9][10]. SURF also has similar 4x4 sub-blocks but uses Haar coefficients to represent the feature. Both algorithms assign a "canonical" orientation by finding the strongest orientation within the window and angling the window in that direction. Both algorithms also normalize the feature vectors to allow for intensity invariance. Moreover, Lowe's SIFT implementation allows the reduplication of keypoints that have more than one, strong canonical orientation, providing more stability in the detected features.

After lengthy testing, the feature descriptor applied in this algorithm is similar to the SURF method. The Haar Transform was performed on a window, broken into 4x4 sub-blocks, and using the sum of Haar coeffients and their absolute values for each sub-block. The size of the Haar filter selected is relative to the scale of the feature detected. This implementation is exactly "Upright SURF" (U-SURF) presented in [7], in which orientation is not taken into account. Because the calculation of Haar coefficients involves a box filter split into ones and negative ones, the integral image was again used. Though orientation was applied in several other algorithm attempts, it was not necessary in the SURF implementation because of SURF's inherently high performance, and because integral images only provide fast computation of rectangular regions—including orientation information would have required an approximation of rotated Haar filters.

D. Comparing Features Between Paintings

Each test painting is compared to the set of training images, and within each image comparison, every feature descriptor of the painting is compared quantitatively with every descriptor in the training image. Features match when they are closest to each other in Euclidean distance. A dot product between the two descriptor vectors is used to approximate Euclidean distance for small angles and because of Matlab's efficiency in calculating them. Once a feature descriptor of the painting is compared with all descriptors in the training image, the dot products are sorted. If the ratio of the two largest dot products is above a certain threshold, the two features are deemed a match. This process is repeated between every descriptor in the painting and all the descriptors of every training image, so it is the bottleneck of the algorithm in terms of speed.

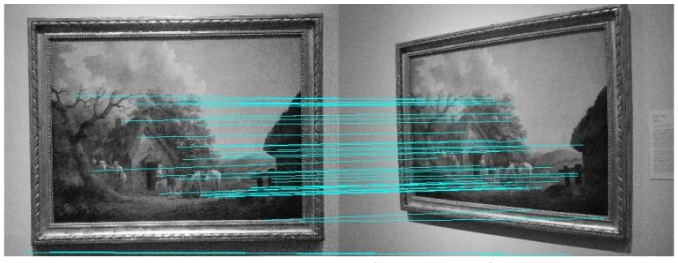

Figure 6: Feature matching example

After all the matches are counted (see Figure 6), the training image with the largest number of matches is designated the painting match, and the name of the painting is returned. Because SURF has a fairly low false-positive rate, a threshold was used to speed up the matching process so that if a particular painting returned over 15 feature matches, it was highly likely to be the correct painting and further comparisons to other paintings were deemed unnecessary.

III. Results

The bulk of the testing of the algorithm was spent optimizing the feature descriptor. Initially, a simple Discrete Cosine Transform (DCT) was used; one implementation divided it into 4x4 sub-blocks and used those coefficients, while the other represented the feature with the first 49 coefficients with the highest energy concentration and some high-frequency, detail information. The 4x4 sub-block implementation was slightly more accurate but had a slower running time. Overall, the DCT was relatively accurate, but would usually miss about 3 out of 99 test images.

The SIFT descriptor was another attempted implementation. An orientation histogram was calculated, and the sum of each bin was used to define the descriptors. A method to find the strongest orientation within each window was employed but had minimal improvement. Overall, SIFT did not work as well as expected, with a slightly lower accuracy than the DCT algorithm and a longer processing time, as seen below in Table 1.

Descriptor	DCT	SIFT	SURF
Largest # Matches	261	163	244
Smallest # Matches	9	5	5
Avg processing time*	2.7856 s	4.5290 s	2.3744 s
Accuracy	96/99	95/99	99/99
Avg 'SNR'**	14.44063	11.16999	16.67469
Avg 'Error Margin'***	0.856218	0.864507	0.907540

* When run on AMD 2.00 GHz, 1.00 GB RAM processor

** SNR is the ratio of (largest # matches – second largest # matches)/(second largest # matches) for feature matches when determining an image match and gives a rough idea of the robustness of the descriptor and its strength in describing the feature

*** Error Margin is the ratio between the (largest # matches – second largest # matches)/(largest # matches) for feature matches when determining an image match and gives a rough idea of how effective the descriptor is and its immunity to false-positives

Table 1: Comparison of three different descriptor algorithms

The final descriptor algorithm implemented and the one selected was U-SURF as described on the

previous page. The best performing SURF algorithm used a descriptor vector of size 64. Smaller sizes had decreased accuracy and larger vectors were slower. This Upright SURF-64 algorithm was the most accurate and robust of the three tested, with higher SNR and Error Margin calculations, as shown in Table 1. It was also able to classify all 99 test images correctly in the fastest amount of time. On the SCIEN machines, this algorithm had an average processing time of 0.9228 seconds per image. Observing the histogram of the number of feature matches for each image match shows the matching pattern of the algorithm. Although the number of matches is concentrated in the smaller numbers in Figure 7, the peak is reasonable, between 25 and 50. Figure 8 gives a better look at the smaller number of matches. These graphs can be compared with those in Figure 9 of the two other implementations.

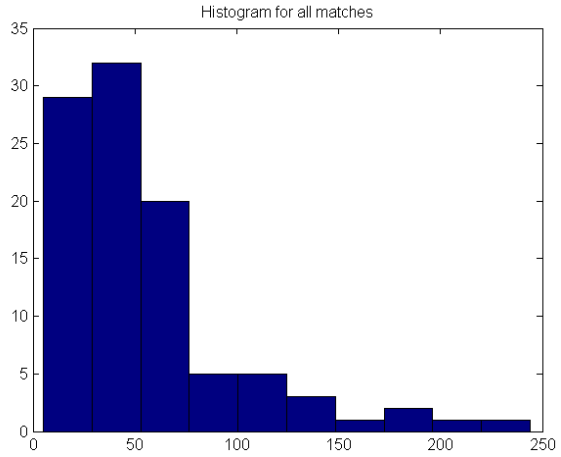

Figure 7: Histogram of number of feature matches for each image match for the selected SURF algorithm

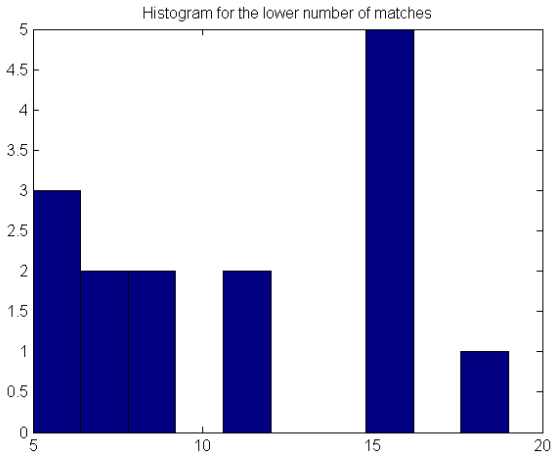

Figure 8: Detailed histogram of lowest number of feature matches for each image match with the selected SURF algorithm

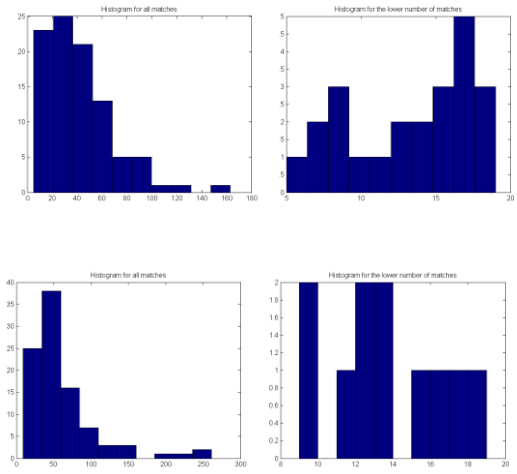

Figure 9: Histograms of SIFT and DCT algorithm implementations. Beginning from the upper left and going clockwise, the images are: SIFT histogram for all matches, SIFT histogram of lowest number of matches, DCT histogram for all matches, DCT histogram of lowest number of matches.

IV. Conclusion

In this project, an object recognition algorithm that matches images taken with a cell phone camera to images in a database was implemented to provide relevant information to the user. After experimenting with several algorithms including SIFT, SURF, and DCT coefficients, an implementation of Upright SURF with 64 features produced the best results: 100% accuracy on the given set of images and an average 0.9228 second runtime on SCIEN machines.

One source of error in the selected implementation was that the detected keypoints did not always reappear in another picture of the same painting from a different perspective. The keypoints were not as stable as in the SIFT implementation where the keypoints are more repeatable and invariant to scale. This phenomenon is particularly evident in the painting shown in Figure 10. The smaller size of the painting in addition to the blur greatly decreased the number of accurate keypoint recognitions and distorted the calculation of the Haar coefficients.

Also to note is that parameters such as the error threshold, distance ratio, window and Haar filter size, and number of features were optimized for this particular database but would not necessarily generalize for a different one. In other words, these

parameters should be calibrated to produce the best results for specific applications.

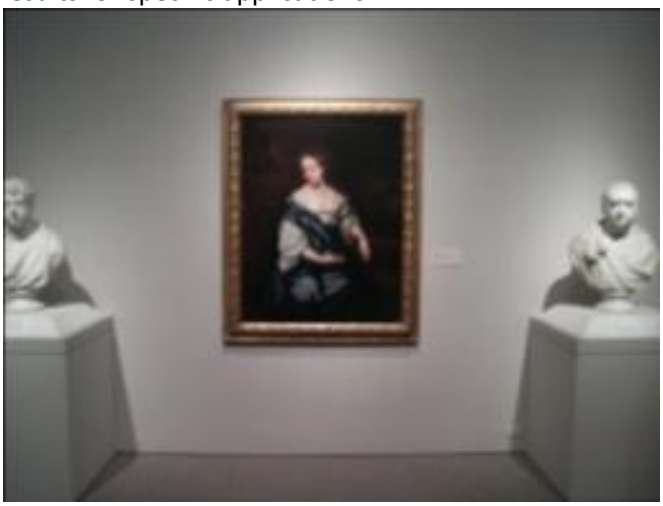

Figure 10: One of the harder paintings to match, due to its smaller scale and blurred features.

The selected implementation can be improved if the accuracy of detecting and matching keypoints is increased. This can be achieved with a more robust feature detector such as SIFT features. The SIFT implementation used in this project did not produce the same results as the Vedaldi implementation [6], and this is another point of improvement that could be made to ensure accurate comparison of the different algorithms that were used. It is possible that the implementation used in this project was faulty, or that the detected keypoints from the approximated SIFT algorithm were not as accurate as actual SIFT keypoints. Finally, further improvement of the implementation would have been to include simple orientation information (in the 0°, 90°, 180°, and 270° direction) to increase the stability of the keypoints, while still allowing the use of rotated Haar, box filters and integral images to calculate the Haar coefficients.

The selected algorithm performed well in terms of both runtime and accuracy, even with images taken under low-light, noisy, and vantage point differences, with some problems as highlighted in Figure 10. Nonetheless, it can be used in mobile technology that will allow for wider application of mobile devices.

V. References

[1] Rohs, M. G. (2004). Using Camera-Equipped Mobile Phones for Interacting with Real-World Objects. *Advances in Pervasive Computing Proceedings* , 265-271.

[2] Yeh, T., Grauman, K., Tollmar, K., & Darrell, T. (2005). A Picture is Worth a Thousand Keywords: Image-Based Object Search on a Mobile Platform. *Conference on Human Factors in Computing Systems.*

[3] Randerson, J. (2004, April 10). *Photo recognition software gives location.* Retrieved 5 27, 2007, from NewScientist.com: http://www.newscientist.com/article.ns?id=dn4857

[4] Lowe, D. (1999). Object recognition from local scale-invariant features. *ICCV* .

[5] Lowe, D. (2004). Distinctive Image Features from Scale-Invariant Keypoints. *International Journal of Computer Vision* , 91-110.

[6] Vedaldi, A. (n.d.). An Implementation of SIFT Detector and Descriptor. University of California Los Angeles.

[7] Bay, H., Tuytelaars, T., & Van Gool, L. (2006). SURF: Speeded Up Robust Features. *9th European Conference on Computer Vision* .

[8] Viola, P. J. (2001). Rapid object detection using a boosted cascade of simple features. *CVPR* , 511-518.

[9] Grabner, M., Grabner, H., & Bischof, H. (2006). Fast Approximated SIFT *. *ACCV* , 918-927.

[10] Porikli, F. (2005). Integral Histogram: A Fast Way to Extract Histograms in Cartesian Spaces. *IEEE Computer Society Conference on Computer Vision and Pattern Recognition* , 829-836.

VI. Work Log

Eric Chu	Algorithm research Pre-processing code Difference of Means code Wrapper code Code integration Debug/Test Project report
Erin Hsu	Algorithm research Local maximum code Orientation code Code integration Debug/Test Project report
Sandy Yu	Algorithm research Feature descriptor code Feature descriptor algorithm testing Debug/Test Project report