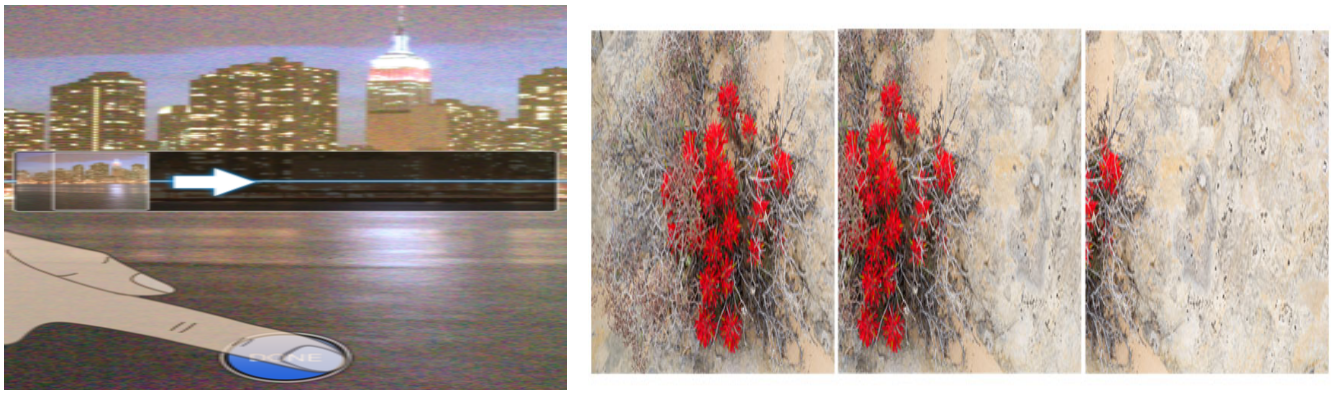
COMPUTER VISUALIZATION FINAL REPORT (EX, §4)
Varun Boodram
81221594

1 Motivation

One of the more interesting developments in digital photography is the nowadays common panoramic photography function built into many cameras and smartphones. Panoramic photography produces images with elongated fields of view, which allows for an increased sense of depth and immersion into the photograph.

Despite the modernization of cameras, however, the techniques used to compose a panoramic photograph have remained largely unchanged since the first panoramic photos of the 1800s. On the iPhone4, for example, panoramic photos are taken by manually panning the camera, as shown in the figure below[1]. The latest Nikon CoolPix series requires that the user manually allign part of the previous image to the image in the viewfinder[2].



(a) iPhone Panoramic Photography

(b) Individual Sections in Panoramic Photography

Figure 1: Panoramic Photography Techniques on Modern Cameras

Both of these techniques have the disadvantage that they require manual alignment; consequently, they can result in images of reduced quality, due to uneven panning or to image misallignment. Computational photography offers an attractive alternative. Manual alignment can be replaced by automatic alignment, based on the presence of "corners".

A corner point can be identified by looking at a small window of the image. If shifting the window slightly in any direction results in

- a large change in intensity, this has identified a corner
- no change in intensity in a single direction, this has identified an edge
- no change in intensity in any direction, this has identified a "flat region"

[1]Image URL: `http://www.wikihow.com/Take-Panorama-Photos-with-an-iPhone`
[2]Image URL: `http://www.nikonusa.com/en/Learn-And-Explore/Article/gp09aubf/panoramas.html!`

Naturally, the next step after identifing good features is to match them between images. If it is possible to match corners between images automatically, the need for manual allignment between images quickly becomes redundant. Implementing appropriate algorithms to acheive this motivates this report

2 Goal of this report

In this report I attempt to conduct the exercise in §4 of the computer visualization class handouts. This exercise requires us to

- Implement an Interest Point Detector
- Implement an Interest Point Descriptor
- Develop an algorithm to plot an ROC Curve

These tasks are addressed respectively in §3, §4, and §5 of this report

3 Implementation of an Interest Point Detector

An interest point is simply a region of expressive texture. This is a point at which the direction of the boundary of an object changes drastically. This could be a geometic corner, or the intersection of two lines.

The implementation of the Harris Corner Detector Method is easily achived via the `cornermetric` function built into the MATLAB image processing toolbox. In this section of this report, I compare my own implementation to the example implementation that can be found in the documentation for `cornermetric`. I applied both implementations to the following image that I took in my home country:

Figure 2: Count Lopinot's Estate House, Trinidad and Tobago

The image was chosen not only because of the presence of geometric corners in the background, but also because of the rich detail in the forground, which should have lead to the identification of several corner features.

However, as this image consistently crashed my MATLAB code, possibly due to its large size, I chose instead to perform my analysis using the following image from a database of standard images for image processing.

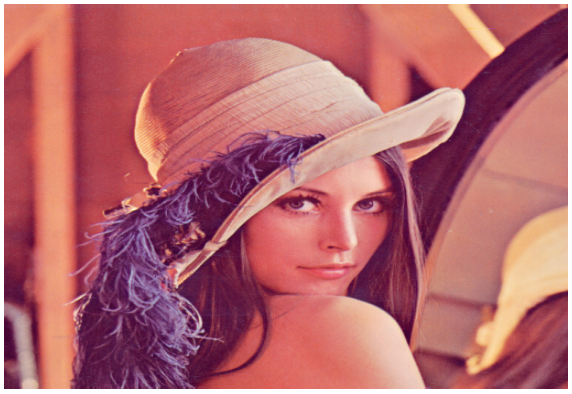


Figure 3: Image used (Obtained from: `http://www.imageprocessingplace.com`)

In this analysis, I compare the number of corners detected by the respective implementations, and to what extent they overlap.

3.1 Summary of the Harris Corner Detector method

The Harris Detector uses the sum of squared differences as its measure of the similarity of between two image patches. If $I(x, y)$ is the intensity of an image patch, and $I(x + u, y + v)$ is its shifted intensity, then

$$E_{\text{WSSD}}(u, v) = \sum_{x,y} w(x, y) \left[I(x + u, y + v) - I(x, y) \right]^2 ,$$

where w is a uniform or Gaussian window function. Smaller values imply greater image similarity. The first–order Taylor Expansion of the above expression is

$$\begin{aligned}
E_{\text{WSSD}}(u, v) &= \sum_{x,y} w(x, y) \left[\cancel{I(x, y)} + uI_x + vI_y - \cancel{I(x, y)} \right]^2 \\
&= \sum_{x,y} w(x, y) \left[\begin{pmatrix} u & v \end{pmatrix} \begin{pmatrix} I_x \\ I_y \end{pmatrix} \right]^2 \\
&= \sum_{x,y} w(x, y) \left[\begin{pmatrix} u & v \end{pmatrix} \begin{pmatrix} I_x \\ I_y \end{pmatrix} \begin{pmatrix} I_x & I_y \end{pmatrix} \begin{pmatrix} u \\ v \end{pmatrix} \right] \\
&= \begin{pmatrix} u & v \end{pmatrix} \left[\sum_{x,y} w(x, y) \begin{pmatrix} I_x I_x & I_x I_y \\ I_x I_y & I_y I_y \end{pmatrix} \right] \begin{pmatrix} u \\ v \end{pmatrix} \\
&= \begin{pmatrix} u & v \end{pmatrix} M \begin{pmatrix} u \\ v \end{pmatrix}
\end{aligned}$$

$E_{\text{WSSD}}(u, v)$ is the equation of an elipse, and the eigenvalues of the covariance matrix, λ_1, λ_2 describe the directions of fastest and slowest change. Together the eigenvalues give a measure of cornerness.

- If both $\lambda 1$ and λ_2 are close to 0, then that region of the image is "flat," meaning that there are no strong corners in that region of the image

- If either λ is significanly larger than the other, then there is a horizontal or vertical edge in that region of the image
- If both λs are high in value, then there is a corner in that region of the image

"High" is of course, relative, but better corners are obtained from higher thresholds

3.2 Results and Discussion

The following image shows the results of the two implementations of the Harris Edge Detector. The images in the top row represent the results obtained via the MATLAB function `cornermetric(I)`, while the bottom row are the results of my own implementation.

Figure 4: The results of two implementations of the Harris Corner Detector. The top row uses the MATLAB function `cornermetric`, while the bottom row is an implementation that does not use this function

The most glaring difference between the two results is the number of cornerpoints detected: the number of corners detected by the `cornermetric` implementation is 5437 versus only 936 by my implementation. This may have been due to some image preprocessing inbuilt into the MATLAB fuction, or the contrast added to the implementation, or possibly, because the mask that I chose for y implementation was not wide enough. The Gaussian filter used in the second implementation also had its parameters chosen more–or–less randomly, and this may have also had an effect on the number of corner points that were detected.

While the results acheived in the second implementation are somewhat disappointing, a significant number of the corners in the second implementation do overlap with those of the first, suggesting that some adjustment to the parameters of the above code could reproduce all of the corners detected in the top row.

4 Implementation of an Interest Point Descriptor

The feature points located in the previous section only identify the pixels at which corners occur. In order to match these pixels to features in another image computationally, we require more information about that point. This is where descriptors become important.

4.1 Summary of the SIFT Feature Descriptor

The state of the art descriptor is the Scale Invariant feature Transform (SIFT), which is able to match features across images invariant to scale, illumination, rotation or viewpoint. The algorithm is extrordiarily involved, but breaks down into six basic steps[3]

1. Generate a Scale Space: This is a two-step process. First, an image is progressively blurred by repreated convolution with the Gaussian operator. Progressively blurred images such as this are called an octave. The original image is then halved, and the resulting image is blurred the same number of times. This is usually repeated to create 4 octaves of 5 blurred images

2. Calculate the DoG: In each octave, we caluclate the difference between successive pairs of images. The resulting images are one fewer than the length of an octave

3. Detect minima: In this step, excepting the first and final image in each octave, we cycle through each pixel. and compare it to the nine surrounding pixels on the same image, as well as the ten pixels above and below. If the pixel being examined is the greatest or the least of the 26 surrounding pixels, it is labeled an extrema

4. Apply the Harris Corner Detector to remove edges

5. Orient the Keypoints: For each keypoint, the orientation and magnitude are calculated. The dominant orientation becomes the orientation of the keypoint

6. Generate SIFT Features: We consider a 16×16 window areound the keypoint, which is further subdivided into sixteen 4×4 windows. In each 4×4 window, the gradient and magnitue is recalulated, and penalized for the distance away from the pixel in question.

[3]based on `http://www.cs.bilkent.edu.tr/ duygulu/Courses/CS554/Notes/LocalFeatures.pdf`

4.2 Results and Discussion

The six steps outlined above were implemented in MATLAB. The fourth step was implemented using the inbuilt function `cornermetric`. To visualize the results, we attempted to overlay indicators of orientation and magnitute. This was achived via the open–source code developed by Andrea Vidaldi, avalaible at `http://www.vlfeat.org/ vedaldi/code/sift.html`. The following figure presents the results of the implementation, showing 100 SIFT features. The size of each circle corresponds to the magnitude, while the line inscribed in the circle shows the direction

(a) Original Image

(b) SIFT Feature Descriptors of Magnitude and Orientation

Figure 5: Results of SIFT Feature Descriptor Implementation

While the implementation clearly obtains SIFT features, the primary difficulty in evaluating these results is the lack of a benchmark image, such as the one generated by `cornermetric` in the previous implementation. This makes it extremely difficult to say if the implementation successfully obtained good SIFT Descriptors. One way to circumvent this problem would be to reapply the implementation to the image under, say, an affine transformation, and determine how well the features matched in each image. This is the subject of the final section of this report.

5 Development of an ROC-Plotting Algorithm

A good detector would be one that would be able to detect most or all true interest points, invariant to geometric and photometric transformations. Matches between descriptors between images I_1 and I_2 are either going to be true or false. The plot of the ratio of the true positive rate (=#true positives/#positives) to the false positive rate (=#false posives/#positives) traces out the Receiver Operator Characteristic curve.

Such a curve could be easily determined if we had a truth ground set availiable (i.e.: if we knew what transformations had derived the transformed image from the original one, we would know exactly where each pixel in the origianl had been mapped to in the second image).

Applying a matching criterion to the two images, such as the Sum of Squared Differences between a feature in the original and a candidate feature in the transformed image would generate the data necessary for plotting this curve.

However, this procedure could easily be critizised for depending on a truth ground set, as this would not normally be available for any pair of images, in general.