Lonnie Princehouse (ljp37)
CS 6670 Spring 2011
Project 1

Note: OpenOffice gives me a mysterious error when I try to export this as HTML. So here's a PDF instead.

Custom Feature Descriptor

My custom feature descriptor consists of a MOPS descriptor along with data derived from a hue histogram generated by the pixels sampled in the MOPS patch. This is an attempt to inform the MOPS descriptor with color information, but I can't say it was particularly successful.

First, 32x32 pixels are sampled from the MOPS patch area of the original image, and their color values converted from RGB to HSV. Saturation and value are discarded, and a histogram is built from the hue values. There are 8 buckets in the histogram; I also tried 16, but it doesn't seem to make much difference.

A hue descriptor is then computed as the difference of histogram buckets separated by distance 1, distance 2, etc. Further differences are divided by two.

```
circular_difference(dist) = { hist[(i+dist)%n] – hist[i] * 2-(dist-1) | 0 <= i < nbuckets}
hue_descriptor = circular_difference(1) + circular_difference(2) + circular_difference(3)
(where + is vector concatenation)
```

The reason for using this differencing scheme instead of just the histogram is that comparing descriptors by taking the squared difference of buckets in the histogram would miss the relationship between adjacent buckets; a descriptor with a high value in histogram_bucket[1] should be closer to a descriptor with a high value in histogram_bucket[2] than histogram_bucket[5], because these adjacent buckets represent nearby hues.

Design Choices

My MOPS descriptor gaussian blurs the image before sampling the MOPS patch; this is an attempt to reduce aliasing by eliminating the highest frequencies.

Sub-pixel sampling with linear interpolation is used for MOPS and my custom descriptor. This was easier to code than a fancier interpolation, but is a dramatic improvement over no interpolation. For example, here's a rotated image using no interpolation versus my linear interpolation for sub-pixel sampling:

Features.cpp contains a global config structure with various options, including whether to use ANMS or a basic threshold on Harris values to cull unimportant features. The default is ANMS, which will start with a small feature radius and gradually expand it until the number of features that are local maxima within their radius does not exceed an ANMS upper bound. The default is <= 150 features.

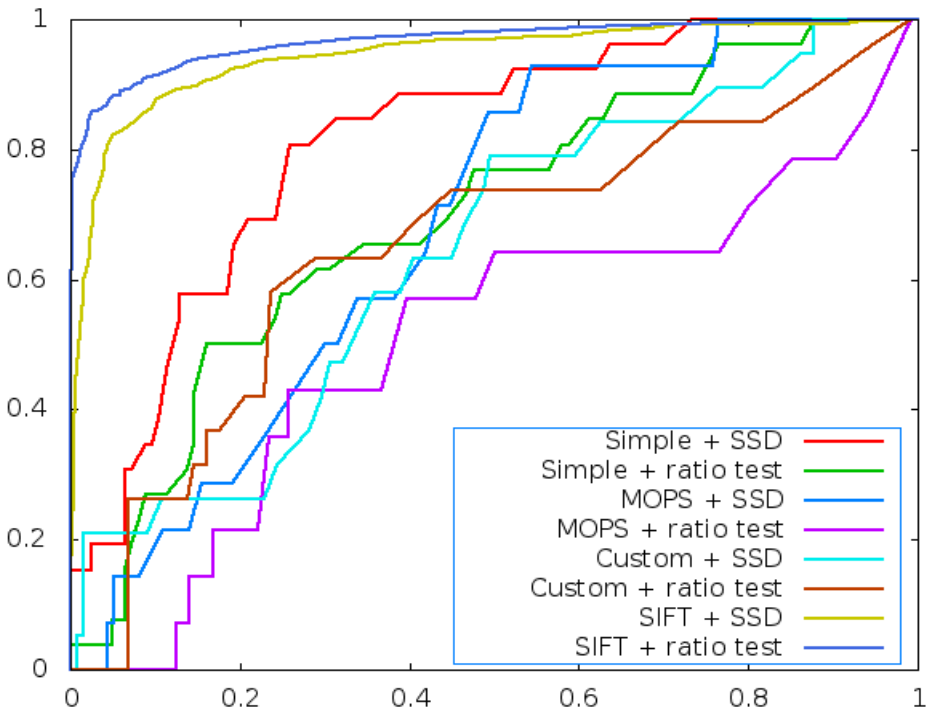
I ended up writing a limited-functionality matrix class that is sufficient for 2D affine transformations. The * operator is overloaded, which makes the various transforms needed for the MOPS operator rather elegant in code:

```
// move feature to the origin
Matrix translate = Matrix::translation(-f.x, -f.y);

// orient according to feature angle
Matrix rotate = Matrix::rotation(-f.angleRadians);

// translate so that corner of descriptor patch  is at the origin
Matrix translate2 = Matrix::translation(patch_size/2, patch_size/2);

// scale to patch coordinates (MOPS_DESCRIPTOR_WINDOW_SIZE)
Matrix scale = Matrix::scale(((double)config.MOPS_DESCRIPTOR_WINDOW_SIZE) / patch_size);

Matrix patch_coordinates = scale * translate2 * rotate * translate;
Matrix image_coordinates = patch_coordinates.inverse();

for(int u = 0; u < config.SIMPLE_DESCRIPTOR_WINDOW_SIZE; u++) {
  for(int v = 0; v < config.SIMPLE_DESCRIPTOR_WINDOW_SIZE; v++) {
      double x = u, y = v;
      image_coordinates.affine_transform(x,y);
      data.data[u][v] = sample_subpixel(image, x, y, 0);
  }
}
```

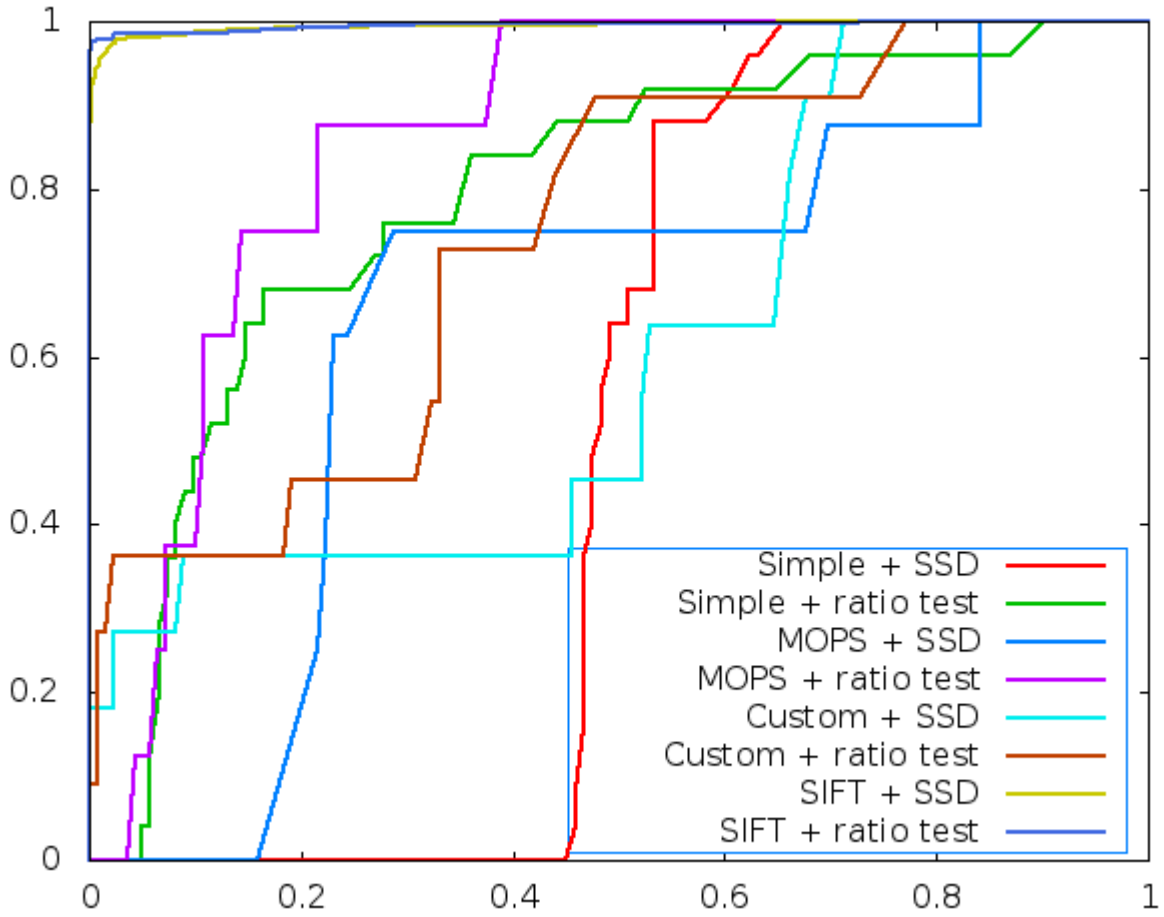
But it really would have been nice to be able to use one of the many open source linear algebra libraries for C++ instead. All of my code is in features.cpp, with the exception of the rgb-to-hsv function in rgb_hsv.h which I shamelessly copied and pasted (attribution in the header file).

Performance

1. ROC curves

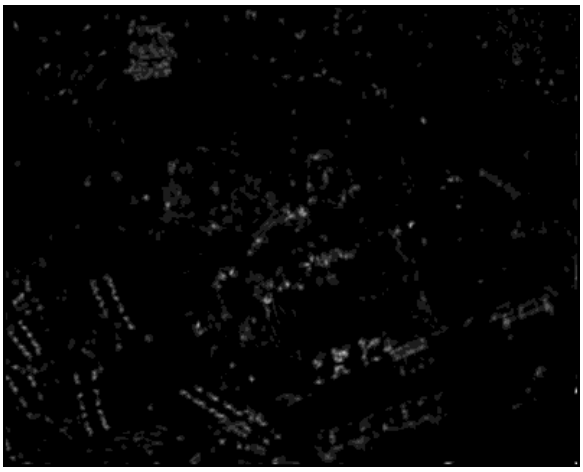


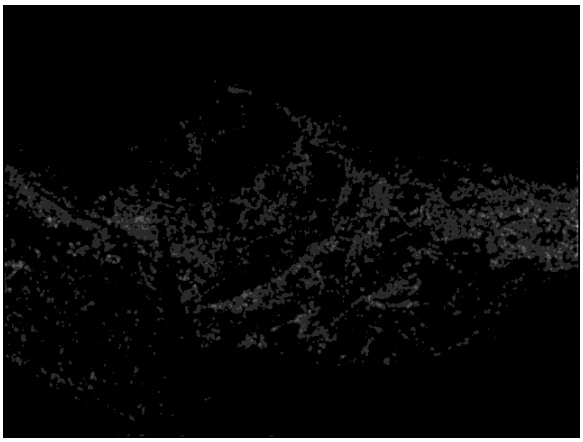
ROC curves for graf img1, img2. Sadly, MOPS is worse than simple 5x5?



ROC curves for yosemite benchmark. MOPS redeems itself.

2. Harris operator images, shown side-by-side with original grayscale image. Harris images are normalized.



Bikes benchmark average AUC

Descriptor	SSD	Ratio
Simple 5x5	0.36	0.53
MOPS	0.6	0.61
Custom	0.6	0.62

Graf benchmark average AUC

Descriptor	SSD	Ratio
Simple 5x5	0.56	0.51
MOPS	0.59	0.54
Custom	0.514	0.53

Leuven benchmark average AUC

Descriptor	SSD	Ratio
Simple 5x5	0.26	0.59
MOPS	0.51	0.49
Custom	0.52	0.56

Wall benchmark average AUC

Descriptor	SSD	Ratio
Simple 5x5	0.52	0.53
MOPS	0.6	0.56
Custom	0.52	0.6

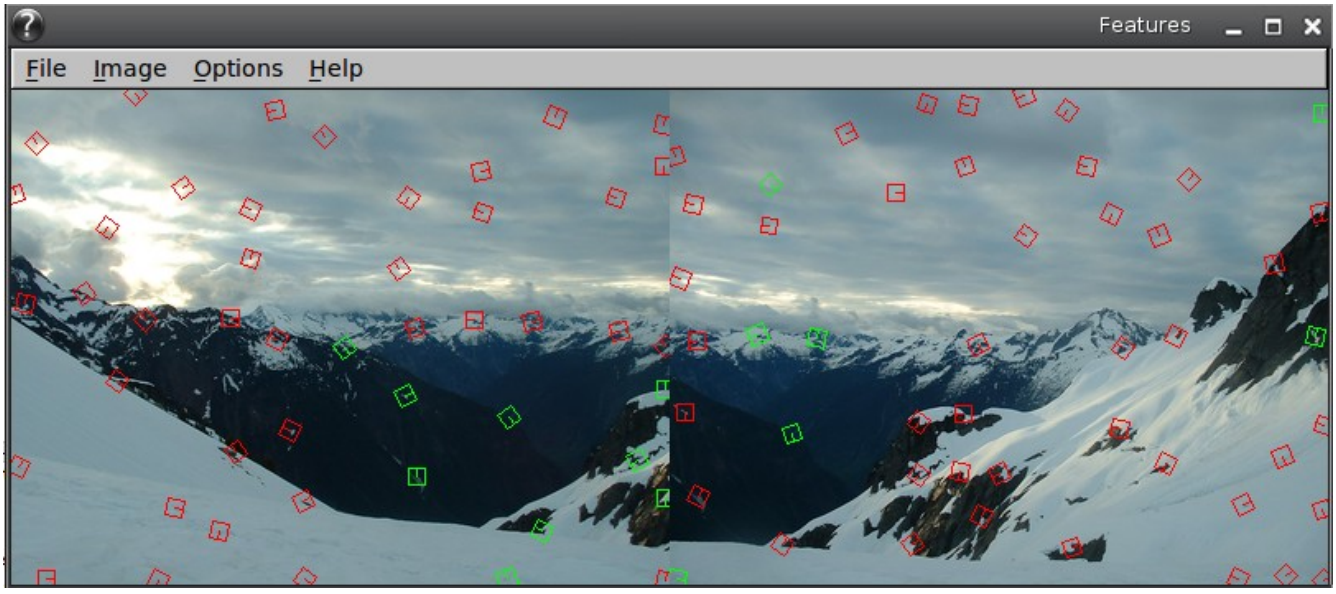
Strengths and Weaknesses

Custom descriptor doesn't work very well.

C++ is intensely frustrating, especially if you want to write efficient code. I wasted hours in a special hell devised of auto_ptr and references.

My Own Images

Here's a picture of MOPS + ratio test getting a matching **spectacularly wrong** on a couple of pictures I took for a panorama near Mt. Baker, WA. The nicely distributed features are due to ANMS, set to 50 features for this example. To the naked eye, it's clear that a lot of features do align, but their angles are not consistent from picture to picture. Maybe I should have gaussian blurred the x- and y- derivatives to make them a little less sensitive when trying to compute the gradient...



Extra Credit

Implemented ANMS. See features.cpp :: computeLocalMaxima for details.