Review for Final

Erik G. Learned-Miller
Department of Computer Science
University of Massachusetts, Amherst
Amherst, MA 01003

April 30, 2014

Abstract

This document reviews material you will need for the final. In addition, you should also cover all of the material for Exam 1 from the previous review sheet.

1 SIFT features

Review the slides on SIFT features using the link on the course web site (Nov. 19 lecture).

The key high-level ideas of the SIFT descriptor are

- First, we find points of interest, known as "keypoints" in the image. These will be *local extrema* (minima or maxima) of the Difference-of-Gaussian image pyramid (discussed below).
- Some of these local extrema points are thrown out because they are unstable. This means that a very small change to the image (changing one pixel by one brightness value, for example), may change whether the point is a local extremum. Points in smooth regions of the image and along "ridges" (like images of folding curtains) are typically the types of points that are thrown out.
- After a keypoint is found (it is at a local extremum and it is not unstable), its "scale" is defined to be the level of the image pyramid in which it was found. If it was found in the blurriest level of the image pyramid, it will have a large scale. If it was found in the sharpest level of the image pyramid, it will have a small scale.
- Then a keypoint *orientation* is assigned. This is the, or one of the, dominant orientations in a patch around the keypoint. The dominant orientation is found by looking at a histogram of the gradient orientations in the patch, and picking the orientations with the most values in the histogram.
- Finally, with keypoints that have scales and orientations, we put a set of 4x4 bins down at the given orientation and scale, and build sixteen histograms of local gradient magnitudes. Since each histogram has 8 bins, this will give us a total of 4x4x8 values for on SIFT *descriptor*.

You should understand all of the above points about SIFT keypoints and descriptors. Now, here are some additional details you need to understand to apply SIFT features.

1.0.1 Scale space and difference-of-Gaussian scale space

The scale space of an image is a sequence of successively more blurred copies of an image. By starting with an image I and letting this be layer 1, which we can call I_1 of the scale space, we can form layer 2 of scale space by filtering the image with a Gaussian kernel, which we will write like this:

$$I_2 = I_1 \otimes f_{Gauss}.$$

You can blur more or less depending upon the spread of your Gaussian kernel, but you can just think of using the kernel given above. In general,

$$I_{k+1} = I_k \otimes f_{Gauss}.$$

The set of images $I_1, I_2, ..., I_K$ is called a Gaussian scale space of K levels. The difference-of-Gaussian scale space is formed by taking differences of successive images in the Gaussian scale space:

$$D_1 = I_2 - I_1,$$

and more generally,

$$D_k = I_{k+1} - I_k.$$

The set of images $D_1, D_2, ..., D_{K-1}$ is called a difference-of-Gaussian scale space of $K - 1$ levels.

1.0.2 Local extrema

A pixel in the stack of images $D_1, D_2, ..., D_{K-1}$ is a *local extremum* if it is larger than its 26 neighbors or smaller than its 26 neighbors.

1.0.3 Magnitude and Orientation of a gradient

Using the partial derivative filters described in our previous work on filtering, you can compute the partial derivatives of an image at each point, and hence form the gradient of the image at each point. You can compute the magnitude of the gradient simply by using the vector magnitude formula. Assuming the magntiude is not zero, you can compute the gradient orientation as

$$\theta = \arctan(\frac{\partial I}{\partial y}, \frac{\partial I}{\partial x}).$$

That is, θ is the angle whose tangent is the y-partial over the x-partial. If you don't understand the arctangent function, you need to look it up on your own.

1.0.4 SIFT uses

SIFT was specifically defined to create descriptors for image points such that if the same object were seen again, the descriptor for points on that object would be highly similar to the previous view of that object.

2 RANSAC

Given a bunch of SIFT keypoints and descriptors in one image, and a bunch of SIFT keypoints in another image, we can draw lines between descriptors that are very similar to each other (Euclidean distance betwen two SIFT vectors is less than some ϵ). However, some of these matches will be incorrect. The purpose of RANSAC is to find a transformation that puts as many points from one image into correpondence with the other image as possible. I ask you to refer to the slides on the course web site to review RANSAC.

You should be generally familiar with how to use RANSAC with SIFT features to build an image panorama, as discussed in class.

3 Transforms or transformations

3.1 Basic transforms

This subsection describes the following simple transformations in two dimensions:

- translation
- rotation
- scaling
- shearing

All of these except translation can be implemented by multiplying a coordinate vector

$$\mathbf{x} = \begin{bmatrix} x \\ y \end{bmatrix},$$

by a 2x2 matrix T. This is written

$$\mathbf{x}' = T\mathbf{x}.$$

All of them, including translation, can be implemented by using *homogeneous coordinates*, in which we append a 1 to the end of the coordinate vector,

$$\mathbf{x} = \begin{bmatrix} x \\ y \\ 1 \end{bmatrix},$$

and use a 3x3 matrix T. This is still written as

$$\mathbf{x}' = T\mathbf{x}.$$

For the final exam, please focus your attention the homogeneous forms of the transformations. That is, focus on the 3x3 versions of matrices, not the 2x2 versions. It should be clear from context whether T is 3x3 or 2x2. We now describe these basic transformations.

3.1.1 Shifting (or translation)

Preserves orientation and all the properties of rigid transformations. Given a coordinate vector $\mathbf{x}$ that is being transformed to a new position $\mathbf{x}'$, this can be writeen as

$$\mathbf{x}' = \mathbf{x} + \begin{bmatrix} t_x \\ t_y \end{bmatrix},$$

where t_x and t_y are the horizontal and vertical displacements. When using **homogeneous coordinates**, this can be written as a matrix operation as

$$\mathbf{x}' = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} \mathbf{x},$$

where $\mathbf{x}$ has now had a 1 appended to the end.

3.1.2 Rotation

Preserves areas (or volumes if in 3-D) and all the properties of similarity transforms. This can be written as

$$\mathbf{x}' = \begin{bmatrix} \cos(\theta) & \sin(\theta) & 0 \\ -\sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix} \mathbf{x}$$

in homogeneous coordinates, where θ is the angle of rotation. Note that this operation will rotate the image around the point $(0, 0)$ rather than around the center of the image.

3.1.3 Scaling (Uniform)

A scaling matrix T is written as

$$T = \begin{bmatrix} s & 0 & 0 \\ 0 & s & 0 \\ 0 & 0 & 1 \end{bmatrix},$$

where s is the amount of scaling. If s is greater than one, the image will be magnified. If $0 < s < 1$, the image will be shrunken.

3.1.4 Non-uniform Scaling

In non-uniform scaling, the horizontal and vertical directions are "stretched" by different amounts. The scaling matrix T is written as

$$T = \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix},$$

where s_x is the amount of scaling in the horizontal direction and s_y is the amount of scaling in the vertical direction.

3.1.5 Shearing

A shearing matrix T is written as either

$$T = \begin{bmatrix} 1 & sh_x & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix},$$

for shearing horizontally, or

$$T = \begin{bmatrix} 1 & 0 & 0 \\ sh_y & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix},$$

for shearing vertically. Shearing in the x direction "tilts" the image to the right if the shearing value is positive and to the left if the shearing value is negative.

3.2 Larger families of transformations

In this subsection, we discuss the following families of transformations:

- translations
- rigid transformations
- similarity transformations
- affine transformations
- perspective transformations (homographies)

The transformations in each family are a subset of the transformations below them. For example, the family of rigid transformations is a subset of the family of similarity transformations, and also a subset of affine transformations.

3.2.1 Rigid. (translation + rotation)

Rigid transformations are any combination of translations and rotations. They can be done to a "rigid" object without deforming it or stretching it. They are also known as **Euclidean** transformations. Rigid transformations in 2 dimensions preserve areas (or volumes if in 3-D) and all the properties of similarity transforms.

3.2.2 Similarity (rigid + scale)

. Similarity transformations are any combination of translations, uniform scalings, and rotations. Hence, they can also be described as any combination of a rigid transformation and a uniform scaling. They preserve angles, straightness of lines, and parallelness of lines.

3.2.3 Affine transformations (Similarity + shear).

Affine transformations include any combination of translations, rotations, scaling (including non-uniform) and shearing. Any 3x3 matrix represents an affine transformation.

3.3 Combining transformations to achieve desired goals

In this section, instead of just referring to transformation matrices as T, we will use additional notation to help clarify the role of the transformation. For example R, R_1, R_2, etc., will be used to describe rotation matrices.

To produce a transformation matrix that achieves a sequence of operations, such as rotation, scaling, and translation, one may simply multiply the constituent matrices together, as in:

$$T_{total} = T \times S \times R,$$

where R is a rotation matrix, S is a scaling matrix, and T is a translation matrix. Notice that conceptually, the rotation matrix R is applied to the coordinate vector **x first**. Why? To understand, consider how we would apply these operations one at a time:

$$T_{total} = T \times (S \times (R \times \mathbf{x}))).$$

Notice that, because matrices are applied on the left, the rightmost matrix R is the first one to operate on $\mathbf{x}$. Thus, to write these three operations as a single matrix, we use the associative property of matrix multiplication to conclude that

$$T_{total} = T \times (S \times (R \times \mathbf{x}))) \quad (1)$$

$$= \quad T_{total} = (T \times S \times R) \times \mathbf{x}. \quad (2)$$

Thus, in this case, the total transformation is $T \times S \times R$.

3.3.1 Rotating or scaling around a point that is not the origin

Probably the single most important application of the idea of composing matrix operations together is in rotating or scaling around a point that is not the origin. Typically, the origin of an image is in the corner of an image. But usually, when we want to rotate an image, we want to rotate about a point that is not the corner. Typically, we want to rotate an image around its center, whose coordinates we can call (c_x, c_y). In order to do this, we must first transform every image coordinate to that the middle of the image is at the origin. We do this by *translating* the image an amount equal to the negative of the center coordinates, as in

$$T_{toOrigin} = \begin{bmatrix} 1 & 0 & -c_x \\ 0 & 1 & -c_y \\ 0 & 0 & 1 \end{bmatrix}.$$

We can return an image to its previous position by applying the opposite translation:

$$T_{fromOrigin} = \begin{bmatrix} 1 & 0 & c_x \\ 0 & 1 & c_y \\ 0 & 0 & 1 \end{bmatrix}.$$

By putting these transformations together, with the operation we want first on the **right**, we can produce a rotation of the image about its center, rather than about the point $(0, 0)$:

$$T_{total} = T_{fromOrigin} \times R \times T_{toOrigin}.$$

3.4 Transforming images pixel by pixel

You should understand the programs we discussed in class to rotate images using the "forward" transformation technique and the "inverse" transformation

technique. The next two pages contain two matlab functions called rotate1 and rotate2. In particular, you should understand why rotate2.m was necessary to "fill the holes" created by rotate1.m.

```
function rim=rotate1(im,angle)
r=(angle/360)*2*pi;
rotmat = [cos(r) sin(r); -sin(r) cos(r)];
[rows,cols]=size(im);
xcoords = 1:cols;
ycoords = 1:rows;
xcarray = repmat(xcoords,[rows 1]);
ycarray = repmat(ycoords,[1 cols]);
xcv = xcarray(:);
ycv = ycarray(:);
pairs = [xcv'; ycv'];
newpairs = rotmat*pairs;
newpairs = round(newpairs);
newpairs(newpairs<1)=1;
newpairs(newpairs>rows)=rows;
rim=zeros(rows,cols);
for i=1:size(newpairs,2)
     rim(newpairs(1,i),newpairs(2,i))=im(pairs(1,i),pairs(2,i));
end
```

```
function rim=rotate2(im,angle)
r=(angle/360)*2*pi;
rotmat = [cos(r) sin(r); -sin(r) cos(r)];
invmat = rotmat ^ (-1);
[rows,cols]=size(im);
xcoords = 1:cols;
ycoords = 1:rows;
xcarray = repmat(xcoords,[rows 1]);
ycarray = repmat(ycoords,[1 cols]);
xcv = xcarray(:);
ycv = ycarray(:);
pairs = [xcv'; ycv'];
oldpairs = invmat*pairs;
oldpairs = round(oldpairs);
oldpairs(oldpairs<1)=1;
oldpairs(oldpairs>rows)=rows;
rim=zeros(rows,cols);
for i=1:size(pairs,2)
     rim(pairs(1,i),pairs(2,i))=im(oldpairs(1,i),oldpairs(2,i));
end
```

4 Alignment

- Name the 3 basic methods of alignment. (Answer: Exhaustive search over a set of transformations, keypoint methods (like SIFT with RANSAC), and gradient descent).

- Describe why exhaustive search is impractical in many situations, especially for complex families of transformations, like similarity transformations.

- Describe tracking as an alignment problem. (Answer: Let I be an image patch showing the item we want to track in frame T. Now, let J be frame $T+1$, and $J_{\mathbf{x}}$ be a patch of J whose center is at coordinates $\mathbf{x}$. The goal is to find the patch in J which best matches I according to some comparison function $f(I, J_{\mathbf{x}})$. This best match will be the new position of the tracker.

- How does gradient descent tracking work? Answer: Find the gradient of the function $f(I, \cdot)$ described above with respect to your transformation parameters. For example, if your set of allowable transformations is the *rigid transformations* (x-translation, y-translation, and rotation), then you need to find the derivatives of f with respect to each of your transformation parameters:
$$\frac{\partial f}{\partial t_x}, \frac{\partial f}{\partial t_y}, \frac{\partial f}{\partial \theta}.$$
In practice, this can be done by simply looking at how the function f changes when you make small changes to each parameter. For example, the derivative of f with respect to x-translation can be found by just shifting the image patch $J_{\mathbf{x}}$ one pixel to the right and looking at the change in f.

- In gradient descent matching, after we have computed the gradient of the matching function f with respect to the transformation parameters, how can we ensure that we do not take "too big of a step" in the direction of the gradient? Answer: Normalize the gradient vector to unit length. If you don't know how to do this, look up how to normalize a vector to unit length.

- The final step of gradient descent alignmen t is to change each parameter by the corresponding component of the normalized gradient vector. For example, if the normal gradient vector above had components [3/13 4/13 12/13], we would add 3/13 to the x-component of the transformation, 4/13 to the y-component of the transformation, and 12/13 to the angle of the rotation matrix

- **coordinate descent**. Another version of gradient descent is called "coordinate descent". Know how it works. Answer: Instead of computing a full gradient, we just see whether we can improve the matching function f by changing each parameter in turn. For example, we can see whether

shifting the image patch one pixel to the right or left improves the value of the comparison function f. In coordinate descent, we choose the new value of the parameter that improves the function f the most. If making the parameter larger or smaller makes the function worse, we leave the value the same during that iteration.

5 Face Detection and Face Recognition

Suppose you read an article with a quotation from the CEO of a new face recognition company that says, "Our algorithm achieves 99.9% in face recognition." Why is this a meaningless statement? Consider the following factors:

- What is the number of people who are considered as potential identities of a face?
- Is the face under even lighting, or arbitrary lighting?
- Is the subject asked to pose for the picture, or are the pictures candid?
- How many training examples are given for each face?

These are just some of the reasons a single number is meaningless. Be prepared to discuss other reasons.

Discuss why face recognition might not yet be practically applied when looking for terrorists in an airport. How is the deployment of face recognition technology dependent upon the costs of the errors that might be made? I discussed these issues in class.

5.1 LIkelihoods for face verification

In the face verification problem, we are trying to decide whether two faces are "same" or "different". In class, I defined a distance function between two patches, one from each of a pair of images. You should understand the following about the basic version of the classifier discussed in class:

- Using training data for the "same" class, meaning pairs of images that show the same person, we measured the frequency of various values of the distance function between patches. This allowed us to produce a *likelihood function* for each patch of "same" images:

$$p(distance|''same'').$$

- Using training data for the "different" class, meaning pairs of images that show two different peopel, we measure the frequency of various values of the distance function between patchs. This allows us to produce a likelihood function for each patch of different images:

$$p(distance|''different'').$$

- When we see a new pair of images and consider a particular patch location, we can produce a likelihood that these pair of patches come from the same person, or from different people by computing the distance between the patches and plugging it into the functions above.

- Finally, we can combine the evidence from all the pairs of patches of a new image pair, by assuming that the distances are statistically independent, and computing:

$$\prod_{i=1}^{N} p(distance_i|''same'')$$

and

$$\prod_{i=1}^{N} p(distance_i|''different'')$$

We then choose the label ("same" or "different") with highest total likelihood.