# CS 231A Computer Vision (Fall 2011)
# Problem Set 3

**Master Set**

Due: Nov. $14^{\text{th}}$, 2011 (9:30am)

## 1 Probabilistic Recursion for Tracking (20 points)

In this problem you will derive a method for tracking a point of interest through a sequence of images. In this problem there exists a true location of an object or interest point that you want to estimate in each frame. We observe this true location through a noisy image. These observations will then be used in conjunction with prior knowledge of the general position where our object or interest point is known to lie. Formalizing this description we can write down the following random variables

$$x_k \in \mathbb{R}^d : \text{the ground truth location of our object or interest point at time } k$$
$$X_k = [x_1, \ldots, x_k]^T : \text{history of the ground truth location to time step } k$$
$$z_k \in \mathbb{R}^c : \text{our noisy position measurement at time k of } x_k$$
$$Z_k = [z_1, \ldots, z_k]^T : \text{history of our noisy position measurement of } x_k \text{ to time step } k$$

To gain our estimate for the position of the object or interest point at time step $k$ we would like to solve for $p(x_k|Z_k)$. In addition, our estimate needs to be computationally efficient, and should be easily updated for when we wish to estimate the position at $k+1$. Thus we will calculate our estimate using a probabilistic recursion. To be able to compute this recursion it can only depend on distributions which are stored in memory. The distributions we have stored in memory are

$$p(z_k|x_k) : \text{our measurement model}$$
$$p(x_k|x_{k-1}) : \text{transition model}$$
$$p(x_{k-1}|Z_{k-1}) : \text{the result of our recursion at the previous iteration}$$

(a) In this part of the problem we will assume that $Z_{k-1}$ and $z_k$ are conditionally independent given $x_k$. Using this assumption derive an expression for $p(x_k|Z_k)$ in terms of $p(z_k|x_k)$ and $p(x_k|Z_{k-1})$. This is conditioned solely on our ground truth location at time $k$ and measurements up to and including time $k-1$. Justify each step of your derivation (it should not be more than a few lines).

(b) The $p(x_k|Z_{k-1})$ term prevents our expression from being solely dependent on distributions that are stored in memory. We will now assume that $x_k$ and $Z_{k-1}$ are conditionally independent given $x_{k-1}$. Now using this assumption write our recursion as an expression which is solely dependent on distributions we have stored in memory. Justify your answer.

Remark: If both our measurement and transition model are Gaussian distributions, there exists a closed form solution for the recursion. In general, the recursion is approximated using numerical techniques known as Monte Carlo methods.

## Solution:

(a) Starting with the probability we wish to obtain

$$p(x_k|Z_k) \tag{1}$$
$$= p(x_k|Z_{k-1}, z_k) \tag{2}$$
$$= \frac{p(z_k|x_k, Z_{k-1})p(x_k|Z_{k-1})}{\displaystyle\int p(z_k|x_k, Z_{k-1})p(x_k|Z_{k-1})dx_k} \tag{3}$$
$$= \frac{p(z_k|x_k)p(x_k|Z_{k-1})}{\displaystyle\int p(z_k|x_k)p(x_k|Z_{k-1})dx_k} \tag{4}$$

where (3) follows from (2) after applying Bayes Rule, and (4) follows from (3) using the assumption that $Z_{k-1}$ and $z_k$ are conditionally independent given $x_k$.

(b) Starting with the expression we obtained from part (a) we have

$$\frac{p(z_k|x_k)p(x_k|Z_{k-1})}{\displaystyle\int p(z_k|x_k)p(x_k|Z_{k-1})dx_k} \tag{5}$$
$$= \frac{p(z_k|x_k)\displaystyle\int p(x_k|x_{k-1}, Z_{k-1})p(x_{k-1}|Z_{k-1})dx_{k-1}}{\displaystyle\int p(z_k|x_k)\int p(x_k|x_{k-1}, Z_{k-1})p(x_{k-1}|Z_{k-1})dx_{k-1}dx_k} \tag{6}$$
$$= \frac{p(z_k|x_k)\displaystyle\int p(x_k|x_{k-1})p(x_{k-1}|Z_{k-1})dx_{k-1}}{\displaystyle\int p(z_k|x_k)\int p(x_k|x_{k-1})p(x_{k-1}|Z_{k-1})dx_{k-1}dx_k} \tag{7}$$

where (6) follows from (5) using $p(a) = \int p(a|b)p(b)db$, (7) follows from (6) using the assumption that $x_k$ and $Z_{k-1}$ are conditionally independent given $x_{k-1}$.

## Grading Guideline:

Total (20 points)

(a) (10 points)

(4 points) applying Bayes rule to the equation
(4 points) leveraging conditional independence correctly
(2 points) correct final solution

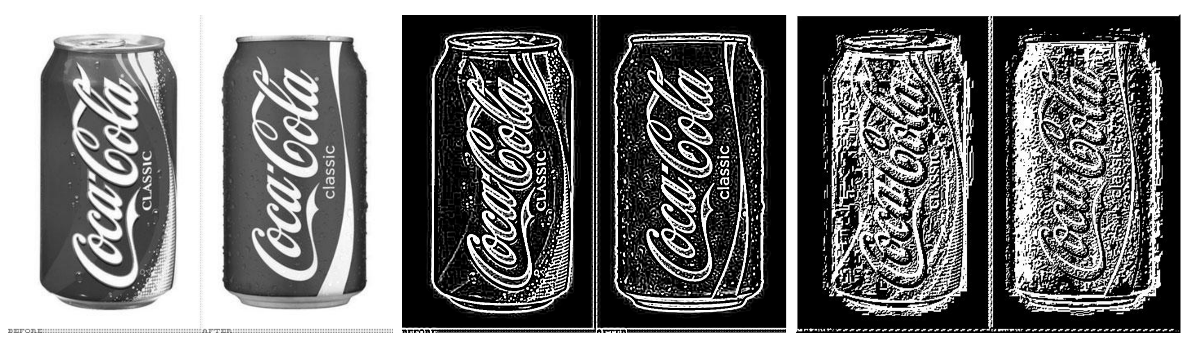
(b) (10 points)
(4 points) applying marginalization correctly
(4 points) leveraging conditional independence correctly
(2 points) correct final solution

## 2 Sifting through SIFT (30 points)

SIFT has become a household name for anyone involved with computer vision. However, it is often naively or incorrectly applied when somebody simply downloads a compiled version from the internet and applies it to their task. The purpose of this problem is to illuminate subtleties of SIFT that can be overlooked, while also giving practical experience to the description and matching methods developed by Lowe et al.

(a) It is desirable to locate image features that are stable across image instances and noise variation. Interest points are related to distinct changes in neighboring pixel values, but choosing the correct operator was the focus of many papers. For comparison, consider Fig. 1 which contains an input image, a difference of gaussians (DoG) of the input, and a gradient of the input. If you wanted to detect interest points that produce a stable signal across image variation and random noise, which operator would you use? Specifically, write down the mathematical formulation for both approaches (DoG and gradient) and identify properties that would support your decision. *hint: think about what these kernels look like in the frequency domain.*

(b) A large part of SIFT's immense popularity (10,000+ citations) is its excellent marriage of a keypoint detector and a descriptor. Choosing the keypoints involves many heuristics such as a spatial sampling, removing edge points, and contrast thresholding (described in sections 3 and 4 of the Lowe et al. 2004), whereas the descriptor is calculated in a straightforward manner. In this problem, we will give you a keypoint detector and ask you to create the descriptor. Please download `PS3Prob2.zip` from the course webpage to complete the rest of this problem.



(a) Original Image (b) Difference of Gaussian of Image (c) Gradient of Image

Figure 1: Different metrics for stable interest points of a given image

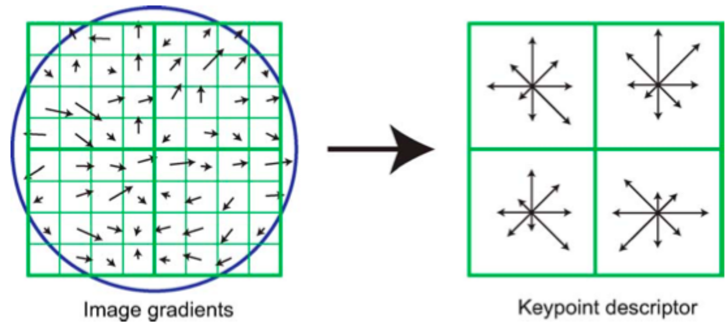



Figure 2: Diagram of keypoint descriptor formation. The blue circle is used to indicate the presence of a gaussian centered at the keypoint.

(i) The keypoint descriptor of SIFT is created by first computing the gradient magnitude and orientation at each image sample point in a region around the keypoint location, then collecting the results into orientation histograms. Please refer to Section 6.1 Lowe's 2004 Paper for details of implementation. Download `PS3Prob2.zip` from the course webpage and examine `sift_test.m`. The interest point detector is given to you, and it is your task to create the 128-d (4×4 array of histograms with 8 orientation bins per histogram) feature vector descriptor for each keypoint.

Follow the step by step instructions in `mySIFTdescriptor.m` to implement your SIFT descriptor and test your code using the given `sift_test.m` with the test image `elmo.jpg`. Print and turn in your `mySIFTdescriptor.m` and the plot of your result SIFT descriptors. The expected output is shown in figure 3.

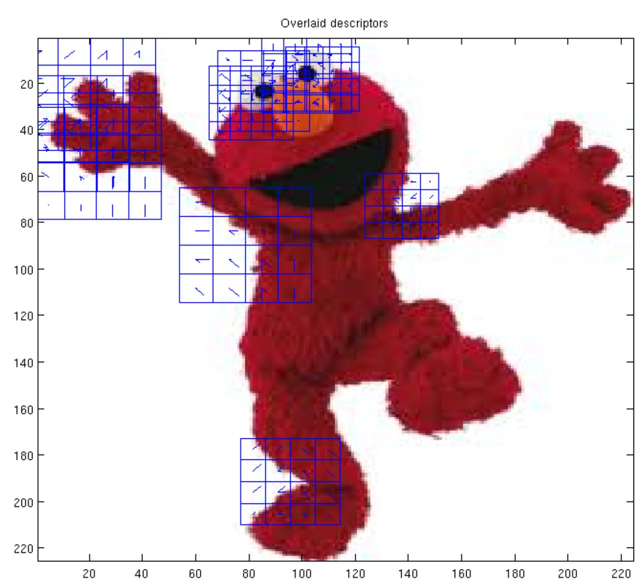



Figure 3: Result image of SIFT descriptors.

(ii) Consider the situation where you are trying to detect a rigid object that has a consistent orientation across object instances (Refer to Fig. 1 for a practical example). What change to the orientation assignment of the original SIFT algorithm could improve detection of the specified object? *hint: what invariances does SIFT have and what are the tradeoffs?*

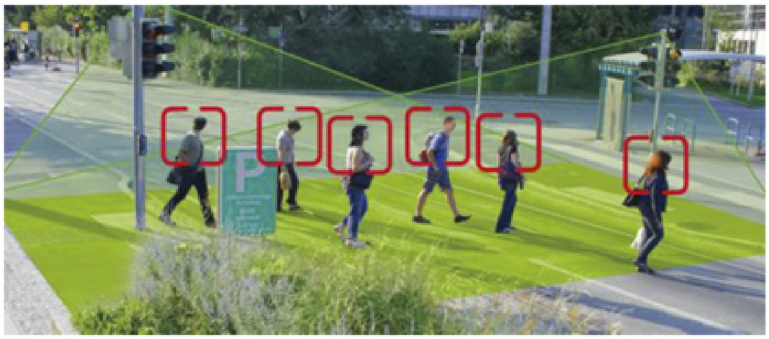


Figure 4: In the example problem of detecting pedestrians with a stationary camera, it is a safe assumption that they will always be of similar orientation.

**Solution:**

(a) The gaussian looks the same in time as in frequency. Thus the Difference of Guassians is the difference of two lowpass filters. It would provide more stable interest points because it captures both the edge nature of the interest points and is able to filter out some of the noise that might cause instability. This filter effect can be seen by looking at the Fourier Transform of the gaussian kernel, which is a guassian itself, and noticing that this kernel will suppress high frequency noise. On the other hand, the gradient will amplify high frequency noise.

(b) There are two parts to this problem:
(i)

```
% initialize your output
gray_img = rgb2gray(img);

% parameters
num_angles = 8;
num_bins = 4;
num_samples = num_bins * num_bins;
patchSize = 16;

% initialize descriptors to zero
descriptors = zeros(size(keypoints,1), num_samples * num_angles);

% for all patches
for i=1:size(keypoints,1)
```

```
H = fspecial('gaussian',10, keypoints(i,3));
scaled_img = imfilter(gray_img,H,'replicate');

G_X = [-1 0 1];
G_Y = [-1 0 1]';
I_X = filter2(G_X, scaled_img, 'same'); % vertical edges
I_Y = filter2(G_Y, scaled_img, 'same'); % horizontal edges
I_mag = sqrt(I_X.^2 + I_Y.^2); % gradient magnitude
I_theta = mod(atan2(I_Y,I_X) - keypoints(i,4),2*pi); %rotation invariance!
I_theta(find(isnan(I_theta))) = 0;
angle_step = 2 * pi / num_angles;
angles = 0:angle_step:2*pi;

% find coordinates of sample points (bin centers)
sample_x_t = round(keypoints(i,1));
sample_y_t = round(keypoints(i,2));

% find window of pixels that contributes to this descriptor
x_lo = sample_x_t - patchSize/2;
x_hi = sample_x_t + patchSize/2 - 1;
y_lo = sample_y_t - patchSize/2;
y_hi = sample_y_t + patchSize/2 - 1;

try
  patch_mag = I_mag(y_lo:y_hi,x_lo:x_hi) .* ...
      fspecial('gaussian',patchSize, patchSize/2);
  patch_ang = I_theta(y_lo:y_hi,x_lo:x_hi);
catch
  continue; %skip the point that is too close to the image boundary
end
feature = [];

for x = 1:4:patchSize-3
  for y = 1:4:patchSize-3
    smallhist = zeros(1,8); %the histgram for one 4x4 region

    %find the correct angle bin and increment it by the magnitude
    %loop around all points inside each 4x4 section
    for x_off = 0:3
      for y_off = 0:3
        if patch_mag(y+y_off,x+x_off) > 0
          %If the magnitude is zero, then there will be no
          %addition to the orientation histogram
          for a = 1:8 %across the angles
            if a == 8 %the angle bin must be the last one
              smallhist(a) = smallhist(a) + patch_mag(y+y_off,x+x_off);
              break;
```

```
              end
              if patch_ang(y+y_off,x+x_off)<angles(a+1)
                smallhist(a) = smallhist(a) + patch_mag(y+y_off,x+x_off);
                break;
              end
            end
          end
        end
      end

      feature = [feature smallhist];
    end
  end
  %The computed feature is now the completed descriptor for that
  %particular interest point!

  descriptors(i,:) = feature;
end

%Normalization step:

% find indices of descriptors to be normalized (those whose norm is larger than 1)
tmp = sqrt(sum(descriptors.^2, 2));
normalize_ind = find(tmp > 1);

descriptors_norm = descriptors(normalize_ind,:);
descriptors_norm = descriptors_norm ./ ...
    repmat(tmp(normalize_ind,:), [1 size(descriptors,2)]);

% suppress large gradients
descriptors_norm(find(descriptors_norm > 0.2)) = 0.2;

% finally, renormalize to unit length
tmp = sqrt(sum(descriptors_norm.^2, 2));
descriptors_norm = descriptors_norm ./ repmat(tmp, [1 size(descriptors,2)]);

descriptors(normalize_ind,:) = descriptors_norm;
```

(ii) In order to make the descriptors more distinctive, you could remove the rotation invariance in SIFT by skipping the step in which you offset the descriptor gradients by the orientation of the keypoint. In the case of noisy samples (such as the pedestrian case), it would be useful to use your prior knowledge of the image structure - that is, the people are always upright - to discard potentially incorrect feature matches.

**Grading Guideline:**

Total (30 points)
(a) (5 points)
    (3 points) state DoG if difference of two lowpass filters so more stable
    (2 points) state gradient amplifies noise
(b) (25 points)
    (i) (19 points)
        (5 points) Correct code for computing `I_mag`, `I_theta`
        (4 points) Correct code for computing `x_lo` `x_hi`, `y_lo`, `y_hi`
        (3 points) Correct code for determining 4x4 bins
        (4 points) Correct code for getting angle histogram
        (3 points) Correct code for descriptor generation
    (ii) (6 points)
        (2 points) Mentioned removing rotation invariance
        (2 points) Mentioned skipping orientation offset
        (2 points) Mentioned reason - people always upright

## 3 Single Object Recognition Via SIFT (30 points)

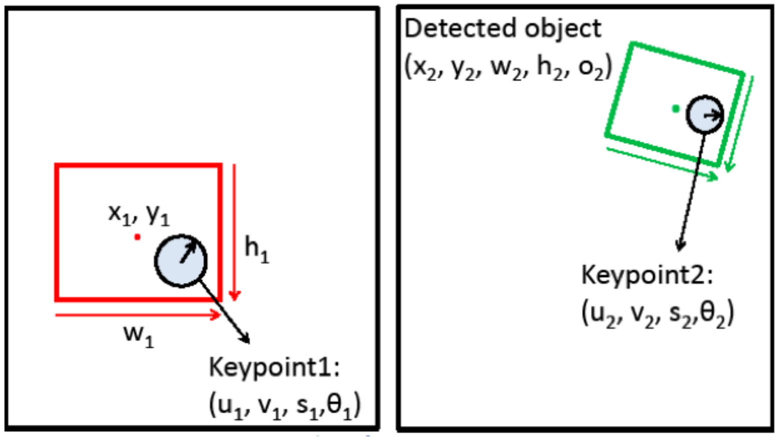
In his 2004 SIFT paper, David Lowe demonstrates impressive object recognition results even in situations of affine variance and occlusion. In this problem, we will explore a similar approach for recognizing and locating a given object from a set of test images. It might be useful to familiarize yourself with sections 7.1 and 7.2 of the paper, which can be found on the course website under the reading. The code and data necessary to solve this problem can be found in `PS3Prob3.zip` on the course webpage.


Figure 5: Sample Output, showing training image and keypoint correspondences.

(a) Given the descriptor $g$ of a keypoint in an image and a set of keypoint descriptors from another image $f_1 ... f_n$, write the algorithm and equations to determine which keypoint in $f_1 ... f_n$ (if any) matches $g$. Implement this matching algorithm in the given function `matchKeypoints.m` and test its performance using the `matchObject.m` skeleton code. Load the data in `PS3_Prob3.mat` and run the system with the following line:

```
>>matchObject(stopim{1}, sift_desc{1}, keypt{1}, obj_bbox, stopim{3}, ...
 sift_desc{3}, keypt{3});
```

Note that the SIFT keypoints and descriptors are given to you in `PS3_Prob3.mat` file. Your result should match the sample output in Fig. 5. [Turn in your code and a sample image similar to Fig. 5]




(a) Image 1 (b) Image 2

Figure 6: Two sample images for part (b)

(b) Now given an object in an image, we want to explore how to find the same object in another image by matching the keypoints across these two images.

(i) A keypoint is specified by its coordinates, scale and orientaion $(u, v, s, \theta)$. Suppose that you have matched a keypoint in the bounding box of an object in the first image to a keypoint in a second image, as shown in figure 6. Given the keypoint pair and the red bounding box in Image 1, which is specified by its center coordinates, width and height $(x_1, y_1, w_1, h_1)$, find the predicted green bounding box of the same object in Image 2. Define the center position, width, height and relative orientation $(x_2, y_2, w_2, h_2, o_2)$ of the predicted bounding box. Assume that the relation between a bounding box and a keypoint in it holds across rotation, translation and scale.

(ii) Once you have defined the five features of the new bounding box in terms of the two keypoint features and the original bounding box, briefly describe how you would utilize the Hough transform to determine the best bounding box in Image 2 given $n$ correspondences.

(c) Implement the function `getObjectRegion.m` to recover the position, scale, and orientation of the objects (via calculating the bounding boxes) in the test images. You can use a coarse Hough transform by setting the number of bins for each dimension equal to 4. Use the line in (a) to test your code and change all the 3's to 2, 4, 5 to test on different images. If you are not able to localize the objects (this could happen in two of the test images), explain what makes these cases difficult. [Turn in your `getObjectRegion.m` and matching result images.]

**Solution:**

(a) Find the two nearest neighbors in the second image using Euclidean distance. If the distance to the nearest neighbor is less than t times the distance to the second nearest neighbor (e.g, for $t = 0.8$), then the points are considered to match.

(b) (i) For a single correspondence we can use the following geometric relations to calculate the bounding box in the second orientation.

$$
\begin{aligned}
o_2 &= \theta_2 - \theta_1 \\
w_2 &= w_1 * (s_2/s_1) \\
h_2 &= h_1 * (s_2/s_1) \\
x_2 &= u_2 + (s_2/s_1) * [\cos(o_2) * (x_1 - u_1) - \sin(o_2) * (y_1 - v_1)] \\
y_2 &= v_2 + (s_2/s_1) * [\sin(o_2) * (x_1 - u_1) + \cos(o_2) * (y_1 - v_1)]
\end{aligned}
$$

Note: it is also possible to define the center by computing the distance and relative orientation of the descriptor to the object center.

(ii) To extend this approach to $n$ correspondences we would need to use a hough transform (or some voting scheme) in 5 dimensional bins to determine a good fit for the 5 parameter of the bounding box in the second image.

(c) See code. The difficulties are the strong slant of the stop signs (SIFT is not invariant to out of plane rotation) and the multiple signs (Lowe's idea of using relative distances of first and second nearest neighbors as threshold assumes that there is only one valid match in the scene). In image 5, there are two signs, potentially causing the second problem mentioned above. To get full credit, there needed to be mechanisms to handle outliers (incorrect matches), such as Hough voting with a threshold on the number of votes required.


Figure 7: Issues with high affine variation


Figure 8: Clean Result

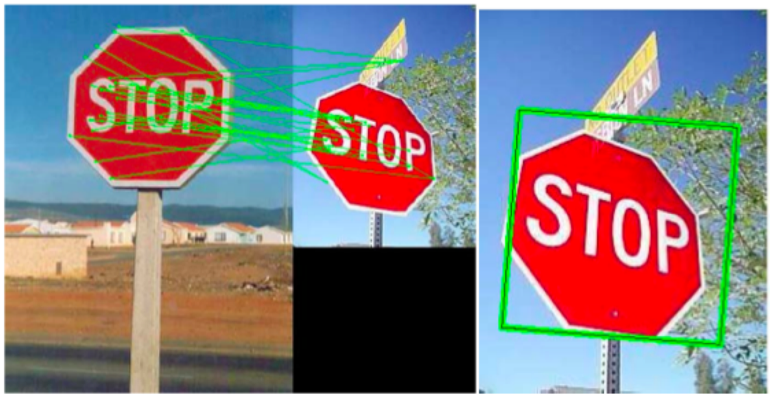



Figure 9: Clean Result

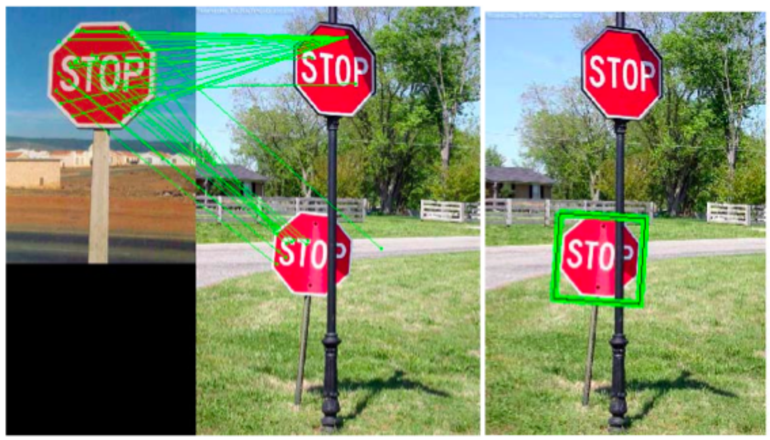



Figure 10: Difficulties with multiple/spurious matchest

```
function matches = matchKeypoints(desc1, desc2, thresh)

n1 = size(desc1,2);
matches = zeros(n1, 2);
n2 = size(desc2,2);
for k1 = 1:n1
    dist = zeros(n2,1);
    for k2 = 1:n2
        dist(k2) = sqrt(sum((desc1(:, k1)-desc2(:,k2)).^2));
    end
    [sval, sind] = sort(dist);
    if sval(1)/sval(2)<thresh
        matches(k1,1) = k1;
        matches(k1,2) = sind(1);
    end

end
```

```
matches = matches(matches(:,1)>0, :);




function [cx, cy, w, h, orient, count] = getObjectRegion(keypt1, keypt2, ...
 matches, objbox, thresh)

% Find parameters for object bounding box
objx = mean(objbox([1 3])); % x-center
objy = mean(objbox([2 4])); % y-center
objw = objbox(3)-objbox(1);
objh = objbox(4)-objbox(2);

% Find parameters for keypoints in image 1
s1 = keypt1(3, matches(1, :));
o1 = keypt1(4, matches(1, :));
x1 = keypt1(1, matches(1, :));
y1 = keypt1(2, matches(1, :));

% Find parameters for keypoints in image 2
s2 = keypt2(3, matches(2, :));
o2 = keypt2(4, matches(2, :));
x2 = keypt2(1, matches(2, :));
y2 = keypt2(2, matches(2, :));

% vote from each keypoint
vote_w = s2./s1*objw;
vote_x = x2 + sum([(objx-x1) ; (objy-y1)] .* [cos(o2-o1) ; -sin(o2-o1)],1).* ...
(s2./s1);%(objx-x1)./s1.*s2;
vote_y = y2 + sum([(objx-x1) ; (objy-y1)] .* [sin(o2-o1) ; cos(o2-o1)],1).*...
(s2./s1); %(objy-y1)./s1.*s2;
vote_o = mod(o2-o1+pi/4, 2*pi)-pi/4; % pi/4 shift is so that 0 rotation...
 doesn't get split bins

% Use four uniform bins for each dimension within range of x.  This
% certainly isn't optimal, but it gets the job done for this problem.
nbins = 4;
bs = assign2bins(vote_w, nbins);
bo = assign2bins(vote_o, nbins);
bx = assign2bins(vote_x, nbins);
by = assign2bins(vote_y, nbins);

% note: having many nested for loops and dynamic array extensions (end+1)
% is very slow, but it makes the code easier to read
cx = []; cy = []; w = []; h = []; orient = []; count=[];
for ks = 1:nbins
  for ko = 1:nbins
```

```
      for kx = 1:nbins
        for ky = 1:nbins
          ind = bs==ks & bo==ko & bx==kx & by==ky;
          if sum(ind)>thresh
            cx(end+1) = median(vote_x(ind));
            cy(end+1) = median(vote_y(ind));
            w(end+1) =  median(vote_w(ind));
            h(end+1) = median(vote_w(ind))/objw*objh;
            % "orient" for SIFT seems to be defined differently than...
            % "orient" in display code, probably because of flipped vertical axis
            orient(end+1) = -median(vote_o(ind));
            count(end+1) = sum(ind);
          end
        end
      end
    end
  end

  % creates nb uniform bins within range of x and assigns each x to a bin
  function b = assign2bins(x, nb)
  b = min(max(ceil((x-min(x))/(max(x)-min(x))*nb), 1), nb);
```

**Grading Guideline:**

Total (30 points)
(a) (7 points)
  (3 points) Correctly find and sort distance
  (3 points) Correctly use threshold to determine matches
  (2 points) Overall correctness
(b) (7 points)
  (i) (5 points) 1 point for each correct equation
  (ii) (2 points) Reasonable explanation
(c) (16 points)
  (3 points) Correctly find all parameters
  (6 points) Correct Hough transform and voting
  (2 points) Use threshold to limit the number of detections
  (2 points) Correct matching result images
  (3 points) Reasonable explanation

# 4 Single Object Matching Via Shape Context (20 points)

Depending on a given problem, some feature descriptors may be better suited than others for object recognition. In contrast to other detectors studied in class, Shape Context [Belongie et al 2002] measures similarity between shapes and exploits it for classification. The methodology also calculates a transformation that would maximally align two potential matches. It will be useful to familiarize yourself with section 3.1 as well as the introduction of section 3 in the paper, which can be found on the class website.

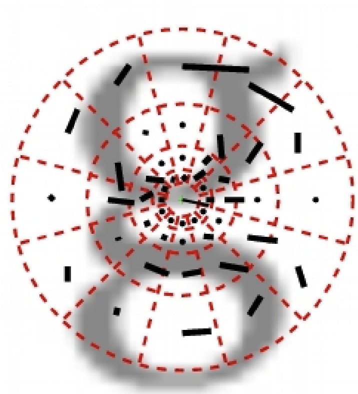
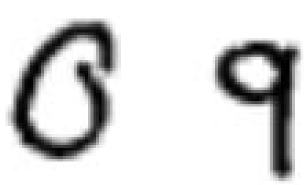


Figure 11: Visualization of polar-coordinate binning

(a) One natural application of Shape Context is to match handwritten digits. In this problem we have provided the data, interest point detection, and matching code for you.

(i) Write a function `compute_shape_context.m` to generate the shape context feature to be used when matching. Specifically your function should take in the minimum radius, maximum radius, number of bins for radius, number of bins for angle, the input data from the edge detection, the mean radius and outliers. Your code should output the mean radius and the shape context feature for each data point. Write your code in `compute_shape_context.m`. In this file there is a description of the input and output data structures, as well as detailed skeleton code on what to do. This code will take in data from the interest point detector and output data to be used in the matching process for shape context. Run the `test_shape_context.m` script to verify your implementation. Turn in your code for `compute_shape_context.m`.

(ii) Run the `compare_digits.m` script and turn in the error for each of the 3 matches. The error for each match is the sum of the squared distances. We calculate the error for you and print it out to a figure after running the `compare_digits.m` script.



(a) (b)

Figure 12: Example data from handwritten digit dataset

(b) Now we consider different images and matching scenarios other than handwriting to test how shape context performs.

(i) We will use preprocessed versions of the images in Fig 13 to test your shape context descriptor. Run the `extend_shape_context.m` script. Turn in the resulting error values and warped figure (figure 3 generated from the code) and indicate what properties of shape context yield this performance.


(a) (b)

Figure 13: Other test images for shape context

(ii) Considering the invariances of Shape Context when would you expect Shape Context descriptor to have poor performance? Give specific circumstances citing the invariances of Shape Context, and use the previous football result as evidence. Given that Shape Context performs well for the digit data, when else would we expect shape context to perform well?

**Solution:**

(a) Some of the letters that look a like get confused such as a 4 and a 9. For the most part there is a certain threshold of that allows for correct detection.

(i)

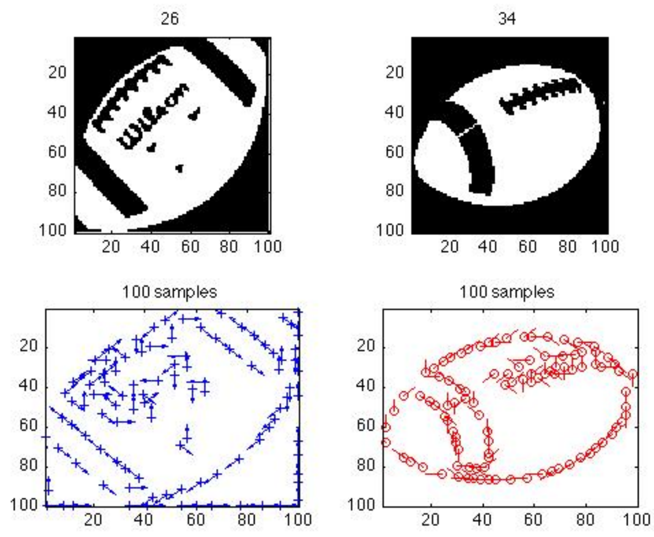
```
function [BH,mean_dist]=compute_shape_context(Bsamp,mean_dist, ...
nbins_theta,nbins_r,r_inner,r_outer,out_vec)
% [BH,mean_dist]=compute_shape_context(Bsamp,Tsamp,mean_dist, ...
%                nbins_theta,nbins_r,r_inner,r_outer,out_vec);
%
% compute (r,theta) histograms for points along boundary
%
% Bsamp is 2 x nsamp (x and y coords.)
% Tsamp is 1 x nsamp (tangent theta)
% out_vec is 1 x nsamp (0 for inlier, 1 for outlier)
%
% mean_dist is the mean distance, used for length normalization
% if it is not supplied, then it is computed from the data
%
% outliers are not counted in the histograms, but they do get
% assigned a histogram
%

nsamp=size(Bsamp,2);
in_vec=out_vec==0;
addpath('SupportCode')
```

```
% compute r,theta arrays
r_array=real(sqrt(dist2(Bsamp',Bsamp'))); % real is needed to
                                           % prevent bug in Unix version
theta_array_abs=atan2(Bsamp(2,:)'*ones(1,nsamp)-ones(nsamp,1)*Bsamp(2,:),...
Bsamp(1,:)'*ones(1,nsamp)-ones(nsamp,1)*Bsamp(1,:))';
theta_array=theta_array_abs;

% create joint (r,theta) histogram by binning r_array and
% theta_array

% normalize distance by mean, ignoring outliers
if isempty(mean_dist)
  tmp=r_array(in_vec,:);
  tmp=tmp(:,in_vec);
  mean_dist=mean(tmp(:));
end
r_array_n=r_array/mean_dist;

% use a log. scale for binning the distances
r_bin_edges=logspace(log10(r_inner),log10(r_outer),5);
r_array_q=zeros(nsamp,nsamp);
for m=1:nbins_r
  r_array_q=r_array_q+(r_array_n<r_bin_edges(m));
end
fz=r_array_q>0; % flag all points inside outer boundary

% put all angles in [0,2pi) range
theta_array_2 = rem(rem(theta_array,2*pi)+2*pi,2*pi);
% quantize to a fixed set of angles (bin edges lie on 0,(2*pi)/k,...2*pi
theta_array_q = 1+floor(theta_array_2/(2*pi/nbins_theta));

nbins=nbins_theta*nbins_r;
BH=zeros(nsamp,nbins);
for n=1:nsamp
  fzn=fz(n,:)&in_vec;
  Sn=sparse(theta_array_q(n,fzn),r_array_q(n,fzn),1,nbins_theta,nbins_r);
  BH(n,:)=Sn(:)';
end
```

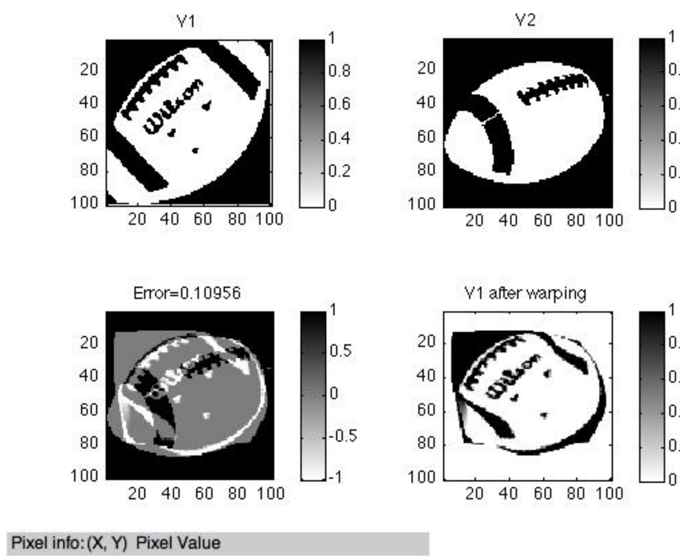
(ii) The errors for all three cases are in the ranges around: 9 & 5 $\rightarrow$ 0.04513, 9 & 0 $\rightarrow$ 0.028678, 9 & 9 $\rightarrow$ 0.024491.

(b) (i) The error given is in the range around 0.06824 this is due to the affine transformation that the football has undergone.

(ii) In this problem our shape context descriptor is scale invariant, but not affine or rotation invariant. You can make the shape context descriptor rotationally invariant but this isn't desirable for our application in this problem. To see this consider a 6 and a 9, rotated these two numbers are very similar to each other and thus we




(a)




(b)

Figure 14: Visualization from script

don't want the descriptor to be rotationally invariant. Thus anytime there is an affine transformation, or rotation between the two candidate correspondences shape context will not perform well as in the football example.

## Grading Guideline:

Total (20 points)
(a) (12 points)
  (10 points) correctly implement code for `compute_shape_context`
  (2 points) correct errors for comparing digits

(b) (8 points)

(2 points) correct errors and warped figure

(3 points) correct interpretation of affine transformation

(3 points) correct interpretation for scale invariance and lack of rotation, affine invariance