CSE 5524 - Homework #6 **10/4/2016**

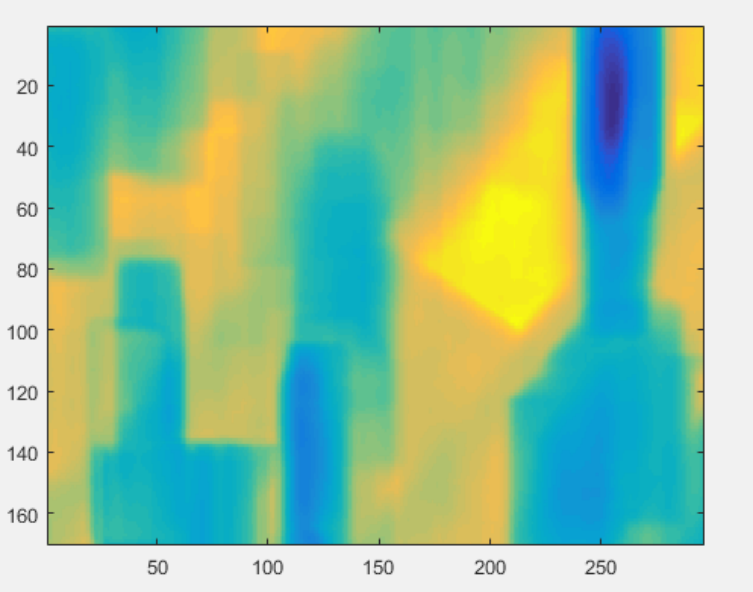
Chandrasekar Swaminathan (swaminathan.42)

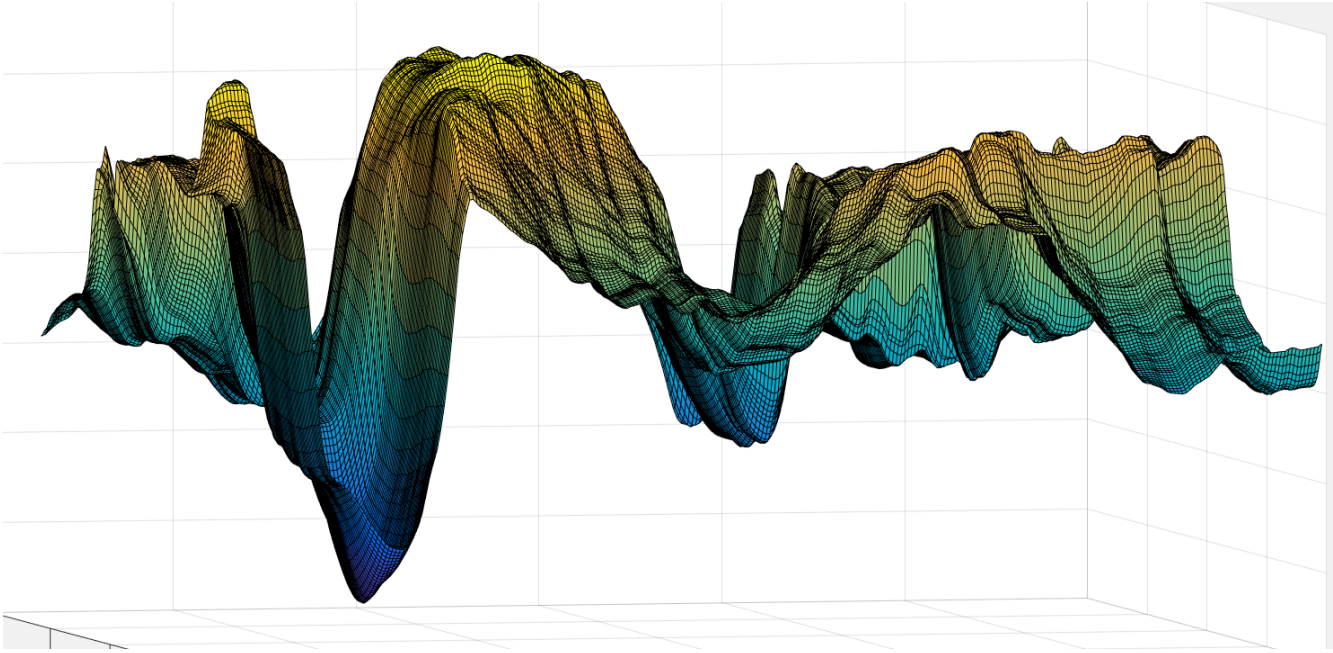
Covariance Tracking:

1) Use the covariance matching technique to find the correct match in the color image given on the WWW site (target.jpg). The model covariance matrix (of <x,y,R,G,B> features) is given below.

modelCovMatrix = [47.917 0 -146.636 -141.572 -123.269;
0 408.250 68.487 69.828 53.479;
-146.636 68.487 2654.285 2621.672 2440.381;
-141.572 69.828 2621.672 2597.818 2435.368;
-123.269 53.479 2440.381 2435.368 2404.923];

Test all possible overlapping windows (each of size 70 rows by 24 columns, with the upperleft-corner as the window origin) in the image with the given model. Save the match distance for each box location in the image at each pixel location (for the origin of the window). Plot/display the match-distance-image. Provide the location of the best match distance for the best candidate. Note the above covariance matrix is biased (normalized with 1/(M*N)), and Matlab's cov function is unbiased by default using 1/(M*N-1), so call cov(X, 1) to make it consistent. Leave the image with colors ranging 0-255 (do not scale/normalize the colors).





The first image indicates the area of the image that matches closest to the model covariance matrix given in the question. The above two images show how the distance between the modelCovarianceMatrix and the covariance matrix defined at each pixel of the image varies. We can see that the distance becomes the smallest at a single pixel in the image and this pixel corresponds to the top left location of the bounding box drawn in the first image.

Mean-Shift:
2) Create a function to extract a feature vector for each pixel in a circular neighborhood (< radius) around (x,y):
[X]=circularNeighbors(img,x,y,radius);
For each pixel, use the same format to return as above (<xi,yi,R,G,B>). That is, X should be a Kx5 matrix, where each row is for one of the pixels in the neighborhood. Assume that the (x,y) passed into the function are real (non-integer) values, and do NOT round them in the function for computation of the neighborhood.

3) Create a function to build a color histogram from the neighborhood of points:
[hist]=colorHistogram(X,bins,x,y,h);
The histogram (hist) should be a bins x bins x bins color cube (RxGxB). The bins should be evenly spaced. For this assignment, use bins=16 then the pixel-value limits for each bin will be {0-15, 16-31, ..., 240-255}.
Weight the construction of the histogram using an Epanechnikov kernel centered at real-valued (x,y) and with bandwidth h. Normalize the histogram/cube so it sums to 1. (This function will be used to make your model histogram “q_model” and to make the candidate test histogram “p_test”)
4) Create a function to calculate a vector of the mean-shift weights (w), a wi for each pixel in the neighborhood:
[w]=meanshiftWeights(X,q_model,p_test);
5) Load the images img1.jpg and img2.jpg, and use the functions above to perform mean-shift tracking.
Build a model from img1 using a circular neighborhood with a radius of 25 pixels centered at (x0,y0) = (150.0, 175.0) and a color histogram of size 16x16x16 (cube).
Build the weighted cube histogram using an Epanechnikov kernel with bandwidth h = 25 (same as the earlier radius).
Run 25 iterations of mean-shift tracking on img2. DO NOT ROUND coordinates or values at any time!
Report the final (x,y) location (DO NOT ROUND) and Euclidean distance between the last two iterations (see Step 4 on the Algorithm slide).

The above image indicates the center pixel and the circular neighborhood around the center that we consider for generating the feature matrix, **[x y R G B]**. The center pixel is near the center of the car. The next three images indicate the center pixel and the neighborhood circular region that is being considered. We can see that the center pixel moves to the left after every iteration towards the center of the car.

The distance between subsequent coordinates decreases as the number of iterations increases. By the end of the last iteration, the center pixel stabilizes around **(138.75, 175.05)** and the euclidean distance between the last point and the point in the previous iteration is **0.203239.**

Code

circularNeighbors.m

```
function [feature_matrix] = circularNeighbors(img, center_x, center_y, radius)
    num_cols = 2*radius;
    num_rows = 2*radius;
    [height, width, ~] = size(img);
    top_left_x = max(1, center_x - radius);
    top_left_y = max(1, center_y - radius);
    % handle scenario when the distance between center of the image and the
    % left/top edge is less than radius
    num_cols = num_cols - (top_left_x-(center_x - radius));
    num_rows = num_rows - (top_left_y-(center_y - radius));
    % handle scenario when the distance between center of the image and the
    % right/bottom edge is less than radius
    bottom_right_x = min(width, top_left_x + num_cols);
    bottom_right_y = min(height, top_left_y + num_rows);
    num_cols = num_cols - ((center_x + radius) - bottom_right_x);
    num_rows = num_rows - ((center_y + radius) - bottom_right_y);
    % center inside the meshgrid
    % relative to 1-numcols, 1-numrows
```

```
    relative_center_x = center_x - top_left_x + 1;
    relative_center_y = center_y - top_left_y + 1;
    [X, Y] = meshgrid(1:num_cols, 1:num_rows);
    % get all points inside the circle
    circle_points = ((X-relative_center_x).^2 + (Y-relative_center_y).^2 ) <= radius^2;
    % create a bounding box around the circle and compute the feature matrix
    % for all pixels in the bounding box
    bounding_box_feature_matrix = generate_feature_matrix(img, top_left_x, top_left_y, num_cols,
num_rows);
    transposed_circle_points = circle_points';
    % change size of circle_points to Kx5 to match the dimensions of the
    % bounding box feature matrix.
    vectorized_circle_points = repmat(transposed_circle_points(:), 1, 5);
    % zero out all entries in the bounding_box_feature_matrix where the point is not
    % in circle
    bounding_box_feature_matrix = bounding_box_feature_matrix .* vectorized_circle_points;
    % filter out rows with all zeros values
    feature_matrix = bounding_box_feature_matrix(all(bounding_box_feature_matrix, 2), :);
    % change all x and y to be relative to the original center
    feature_matrix(:, 1) = feature_matrix(:, 1) + center_x - relative_center_x;
    feature_matrix(:, 2) = feature_matrix(:, 2) + center_y - relative_center_y;
end
```

colorHistogram.m

```
function [hist] = colorHistogram(feature_matrix, bins, center_x, center_y, h)
    min_x = min(feature_matrix(:, 1));
    min_y = min(feature_matrix(:, 2));
    max_x = max(feature_matrix(:, 1));
    max_y = max(feature_matrix(:, 2));
    % Epanechnikov kernel
    [X, Y] = meshgrid(min_x:max_x, min_y:max_y);
    kernel = 1 - (sqrt((X-center_x).^2 + (Y-center_y).^2) ./ h).^2;
    kernel(kernel < 0) = 0;
    kernel_vectors = [];
    % convert the kernel to Nx3 matrix where each row is of the form
    % [X Y weightage]
    for col=1:size(X, 2)
        kernel_vectors = [ kernel_vectors; X(:, col) Y(:, col) kernel(:, col)];
    end
    % sort the feature matrix by X followed by Y coordinate
    sorted_feature_matrix = sortrows(feature_matrix, [1 2]);
    % remove x,y points in the kernel but not in the feature matrix
    filtered_kernel = kernel_vectors(ismember(kernel_vectors(:, 1:2), sorted_feature_matrix(:, 1:2),
'rows'), :);
    % remove x,y points in the feature_matrix but not in the kernel
    filtered_feature_matrix = sorted_feature_matrix(ismember(sorted_feature_matrix(:, 1:2),
filtered_kernel(:, 1:2), 'rows'), :);
```

```
    % filtered_kernel and filtered_feature_matrix have the same X Y points
    % in the same order in the array.
    weighted_hist_vector(:, 1) = floor(filtered_feature_matrix(:, 3)/bins) + 1;
    weighted_hist_vector(:, 2) = floor(filtered_feature_matrix(:, 4)/bins) + 1;
    weighted_hist_vector(:, 3) = floor(filtered_feature_matrix(:, 5)/bins) + 1;
    weighted_hist_vector(:, 4) = filtered_kernel(:, 3);

    % iterative way to create the hist cube
    % hist_cube = zeros(bins, bins, bins);
    % for n=1:size(weighted_hist_vector, 1)
    %     rbin = weighted_hist_vector(n, 1);
    %     gbin = weighted_hist_vector(n, 2);
    %     bbin = weighted_hist_vector(n, 3);
    %     value = weighted_hist_vector(n, 4);
    %     hist_cube(rbin, gbin, bbin) = hist_cube(rbin, gbin, bbin) + value;
    % end

    % computing the same using matrix operations
    all_cube_indices(:, 1) = repmat((1:bins)', bins*bins, 1);
    all_cube_indices(:, 2) = repmat(repeat_each_element(1:bins, bins)', bins, 1);
    all_cube_indices(:, 3) = repeat_each_element(1:bins, bins*bins)';
    all_cube_indices(:, 4) = zeros(bins*bins*bins, 1);
    weighted_hist_vector = sortrows(weighted_hist_vector, [3 2 1]);
    [unique_hist_rows, ~, groupings]  = unique(weighted_hist_vector(:, 1:3), 'rows', 'stable');
    summed_hist_vector = [unique_hist_rows, accumarray(groupings, weighted_hist_vector(:, 4))];
    cubes_indices_present = ismember(all_cube_indices(:, 1:3), summed_hist_vector(:, 1:3), 'rows');
    all_cube_indices(cubes_indices_present, 4) = summed_hist_vector(:, 4);
    hist_cube = reshape(all_cube_indices(:, 4), bins, bins, bins);

    hist = hist_cube ./ sum(sum(sum(hist_cube,3),2),1);
end
```

covariance_distance.m

```
function [distance] = covariance_distance(cov_a, cov_b)
    [~, eigen_values, ~] = eig(cov_a, cov_b);
    eigen_values = eigen_values(eigen_values~=0);
    distance = sqrt(sum(log(eigen_values).^2));
end
```

find_element.m

```
function [row, col] = find_element(A, elem)
    matching_index = find(A==elem);
    col = ceil(matching_index / size(A, 1));
    row = mod(int32(matching_index), size(A, 1));
    if row == 0
        row = size(A, 1);
    end
```

```
end
```

generate_feature_matrix.m

```
function [feature_matrix] = generate_feature_matrix (img, top_left_x, top_left_y, num_cols,
num_rows)
   feature_matrix(:, 1) = repmat(1:num_cols, 1, num_rows)';
   feature_matrix(:, 2) = repeat_each_element(1:num_rows, num_cols)';
   patch = img(floor(top_left_y):(floor(top_left_y)+num_rows-1), floor(top_left_x):(floor(top_left_x)
+num_cols-1), :);
   feature_matrix(:, 3) = reshape(patch(:, :, 1)', num_rows*num_cols, 1);
   feature_matrix(:, 4) = reshape(patch(:, :, 2)', num_rows*num_cols, 1);
   feature_matrix(:, 5) = reshape(patch(:, :, 3)', num_rows*num_cols, 1);
end
```

HW6.m

```
% Problem 1
close all;clc;clear;
img = double(imread(fullfile('input','target.jpg')));
[height, width, ~] = size(img);
model_cov_matrix = [47.917 0 -146.636 -141.572 -123.269; 0 408.250 68.487 69.828 53.479;
-146.636 68.487 2654.285 2621.672 2440.381; -141.572 69.828 2621.672 2597.818 2435.368;
-123.269 53.479 2440.381 2435.368 2404.923];
patch_rows = 70;
patch_cols = 24;
last_col = width - patch_cols;
last_row = height - patch_rows;
k = 1;
cov_diff_mat = zeros(last_row, last_col);
for x=1:last_col
   for y=1:last_row
      feature_matrix=generate_feature_matrix(img, x, y, patch_cols, patch_rows);
      patch_cov = cov(feature_matrix, 1);
      cov_diff_mat(y,x) = covariance_distance(model_cov_matrix, patch_cov);
   end
end
[row, col] = find_element(cov_diff_mat, min(cov_diff_mat(:)));
figure;imagesc(cov_diff_mat);
figure;imagesc(img/255);
hold on;
rectangle('Position', [col row patch_cols patch_rows], 'EdgeColor', 'yellow', 'LineWidth', 3);
hold off;
figure;surf(cov_diff_mat);axis('ij');
pause;

% Problem 2
clc; close all;
img = double(imread(fullfile('input','target.jpg')));
```

```
center_x = 275;
center_y = 25;
radius = 20;
non_zero_rows = circularNeighbors(img, center_x, center_y, radius);
x = non_zero_rows(:, 1);
y = non_zero_rows(:, 2);
imagesc(img/255);
hold on;
% plotting all points considered by the circularNeighbors function, to
% verify if the right points are considered by the circular neighbors
for n=1:size(x, 1)
   plot(x(n)+center_x, y(n)+center_y, 'y.', 'MarkerSize', 20);
end
plot(center_x, center_y, 'r+', 'MarkerSize', 20);
hold off;

% Problem 3
clc;close all;
h = 20;
bins = 16;
hist_cube = colorHistogram(feature_matrix, bins, center_x, center_y, h);

q_model = hist_cube;
p_test = hist_cube;

% Problem 4
weights = meanShiftWeights(feature_matrix, q_model, p_test);

% Problem 5
clc; clear; close all;
img1 = double(imread(fullfile('input','img1.jpg')));
img2 = double(imread(fullfile('input','img2.jpg')));
radius = 25;
h = 25;
center_x = 150.0;
center_y = 175.0;
bins = 16;
number_of_iterations = 25;
original_feature_matrix = circularNeighbors(img1, center_x, center_y, radius);
q_model = colorHistogram(original_feature_matrix, bins, center_x, center_y, h);
figure;imagesc(img1/255);
hold on;
viscircles([center_x center_y], [radius]);
plot(center_x, center_y, 'y+', 'MarkerSize', 5);
hold off;
x = center_x;
y = center_y;
```

```
coordinates = [x y];
for n=1:number_of_iterations
   feature_matrix = circularNeighbors(img2, x, y, radius);
   p_test = colorHistogram(feature_matrix, bins, x, y, h);
   weights = meanShiftWeights(feature_matrix, q_model, p_test);
   sum_of_weights = sum(weights);
   new_coordinate = sum([feature_matrix(:, 1) .* weights', feature_matrix(:, 2) .* weights'], 1) ./
sum_of_weights;
   x = new_coordinate(1);
   y = new_coordinate(2);
   coordinates = [coordinates; x y];
   fprintf('New coordinate: (%g, %g)\n', x, y);
end
for n=1:size(coordinates, 1)
   imagesc(img2/255);
   hold on;
   viscircles([coordinates(n, 1) coordinates(n, 2)], radius);
   plot(coordinates(n, 1), coordinates(n, 2), 'yellow+', 'MarkerSize', 5);
   if n > 1
      title(sprintf('Iteration: %d, Distance between last 2 points: %0.5g \n Point: (%g, %g)', n,
sqrt( (coordinates(n, 1) - coordinates(n-1, 1)).^2 +(coordinates(n, 2) - coordinates(n-1, 2)).^2 ),
coordinates(n ,1), coordinates(n, 2)));
   end
   hold off;
   pause(1);
end
```

meanShiftWeights.m

```
function [weights] = meanShiftWeights(feature_matrix, q_model, p_test)
   bins = size(q_model, 1);
   weights = zeros(1, size(feature_matrix, 1));
   for n=1:size(feature_matrix, 1)
      bin_index = floor(feature_matrix(n, 3:5)./ bins) + 1;
      if p_test(bin_index(1), bin_index(2), bin_index(3)) ~= 0
         weights(1, n) = sqrt(q_model(bin_index(1), bin_index(2), bin_index(3))/p_test(bin_index(1),
bin_index(2), bin_index(3)));
      end
   end
end
```

repeat_each_element.m

```
function [repeated_vector] = repeat_each_element(row_vector, times_to_repeat)
%   eg: row_vector = [1 2 3] & times_to_repeat = 2, output = [1 1 2 2 3 3]
   repeated_rows = repmat(row_vector, times_to_repeat, 1);
%   repeated_rows = [1 2 3; 1 2 3]
   repeated_vector = repeated_rows(:)';
%   repeated_vector = [1 1 2 2 3 3]
end
```