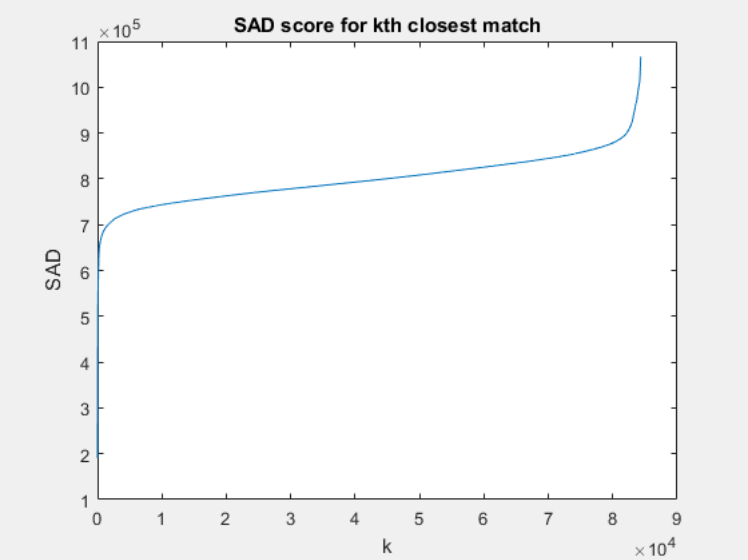
CSE 5524 - Homework #9 **11/1/2016**

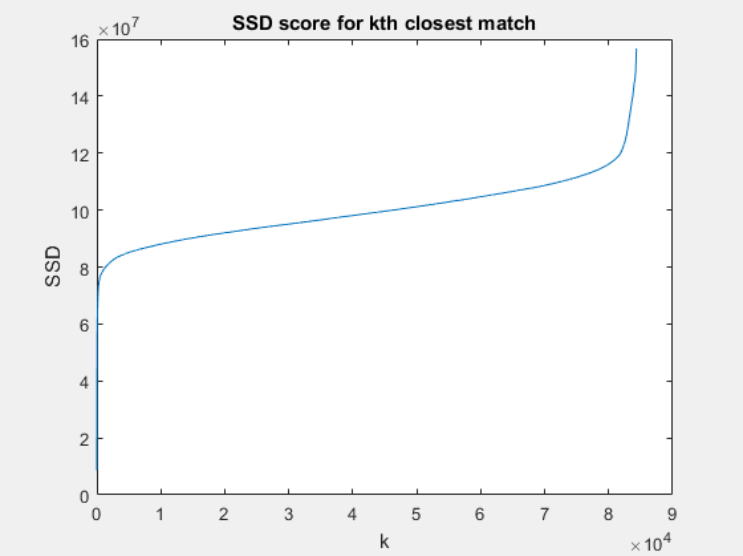
Chandrasekar Swaminathan (swaminathan.42)

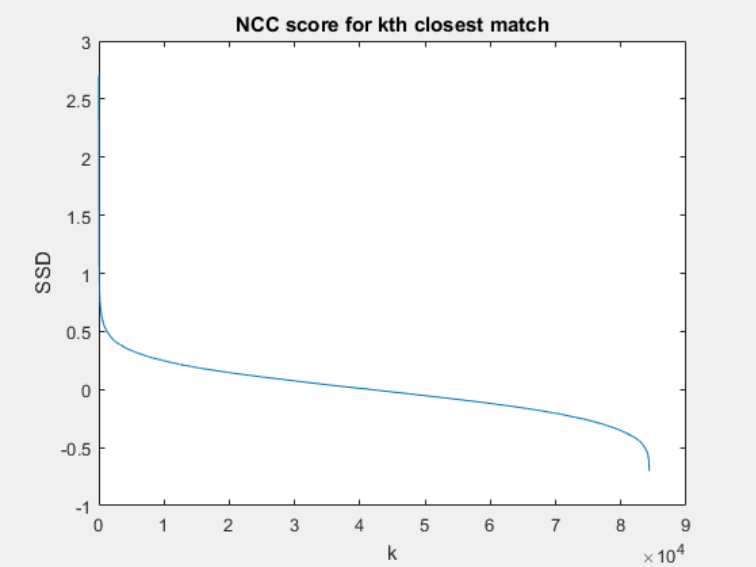
Template Matching:

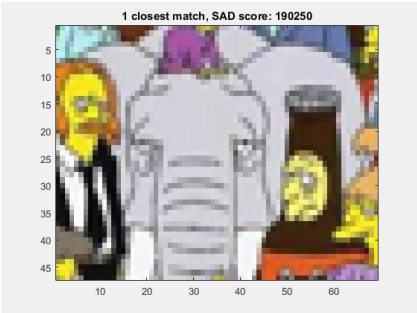
1) There's an elephant in the room. Can you find it? Search for the template template.png in the search image search.png using color-based SSD, SAD, and NCC (make sure the standard deviation is "unbiased" with N-1). Assume the origin is in the center of the template image for each approach (Note: there should be a border around the search image where the metrics cannot be computed).

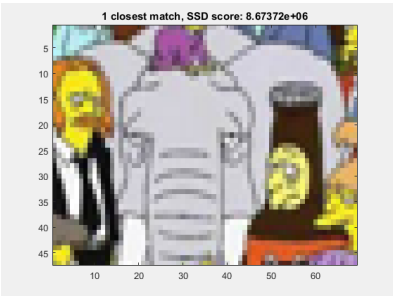
Sort the resulting scores from best to worst for each of the approaches. Plot the sorted scores and show the patches corresponding to the 1st, 2nd, 5th, 10th, 100th, and 500th closest matches. Compare your results.

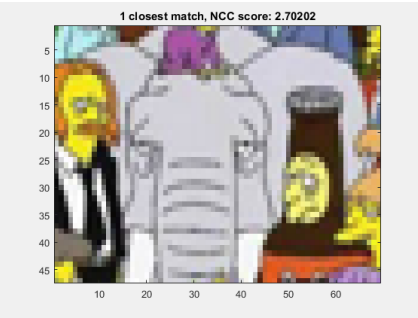


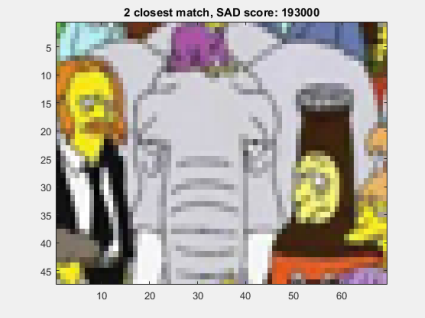


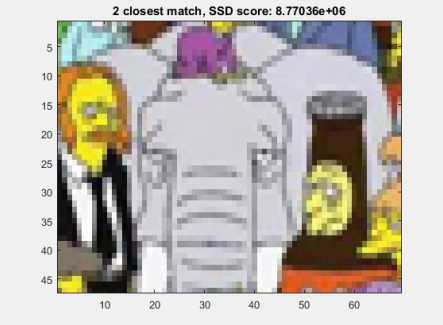

NCC score for kth closest match
SSD
k
×10^4

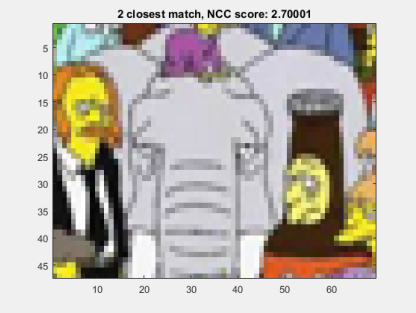

1 closest match, SAD score: 190250

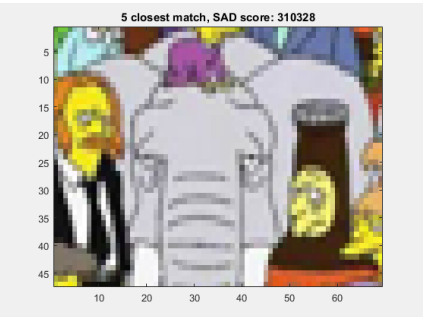

1 closest match, SSD score: 8.67372e+06

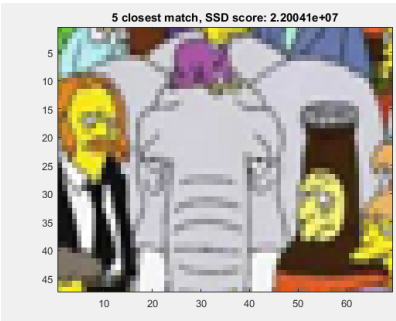

1 closest match, NCC score: 2.70202

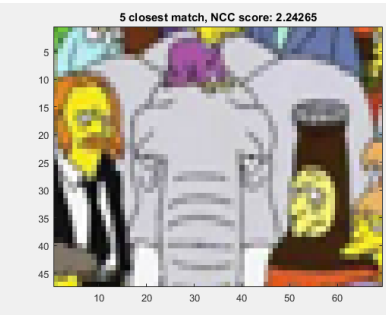

2 closest match, SAD score: 193000

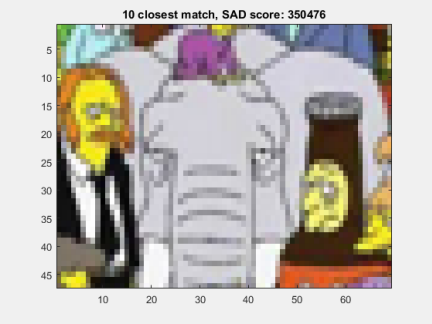

2 closest match, SSD score: 8.77036e+06

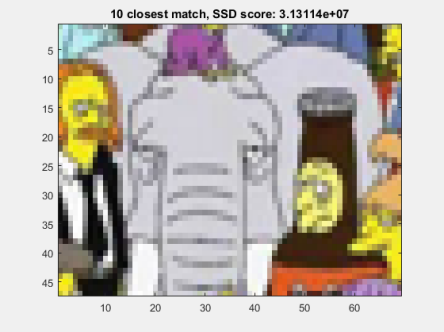

2 closest match, NCC score: 2.70001

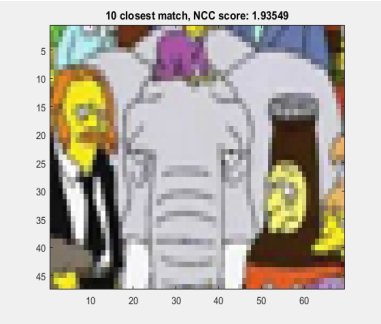

5 closest match, SAD score: 310328

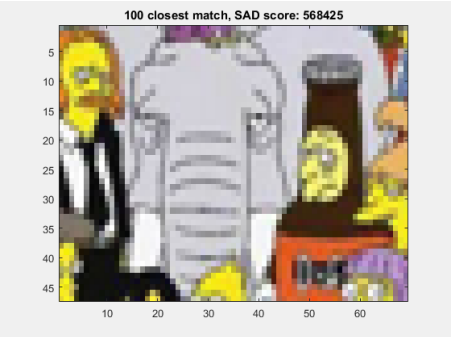

5 closest match, SSD score: 2.20041e+07

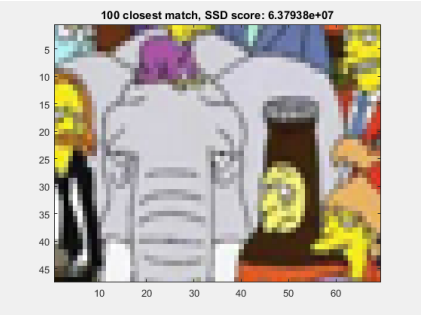

5 closest match, NCC score: 2.24265

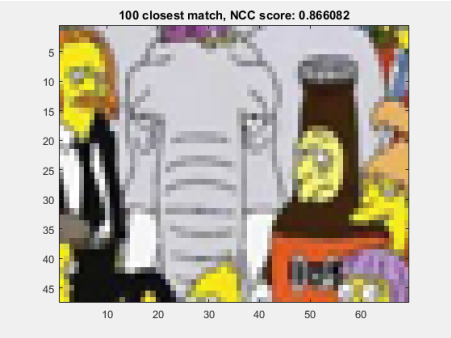


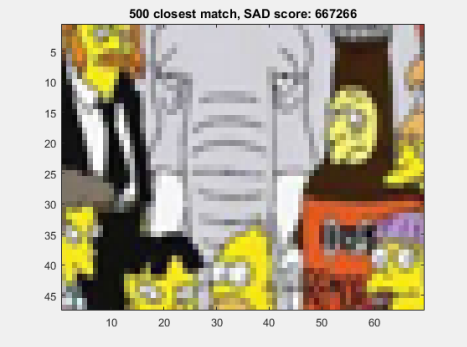


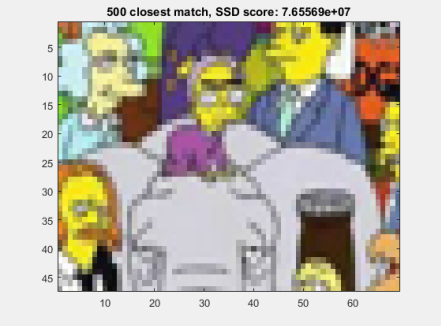


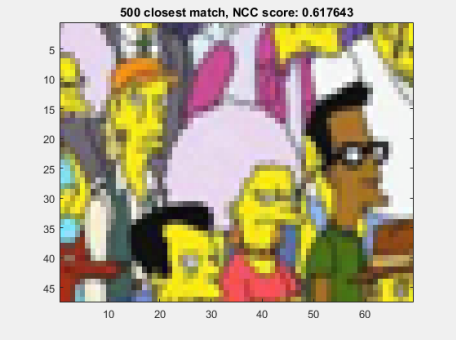








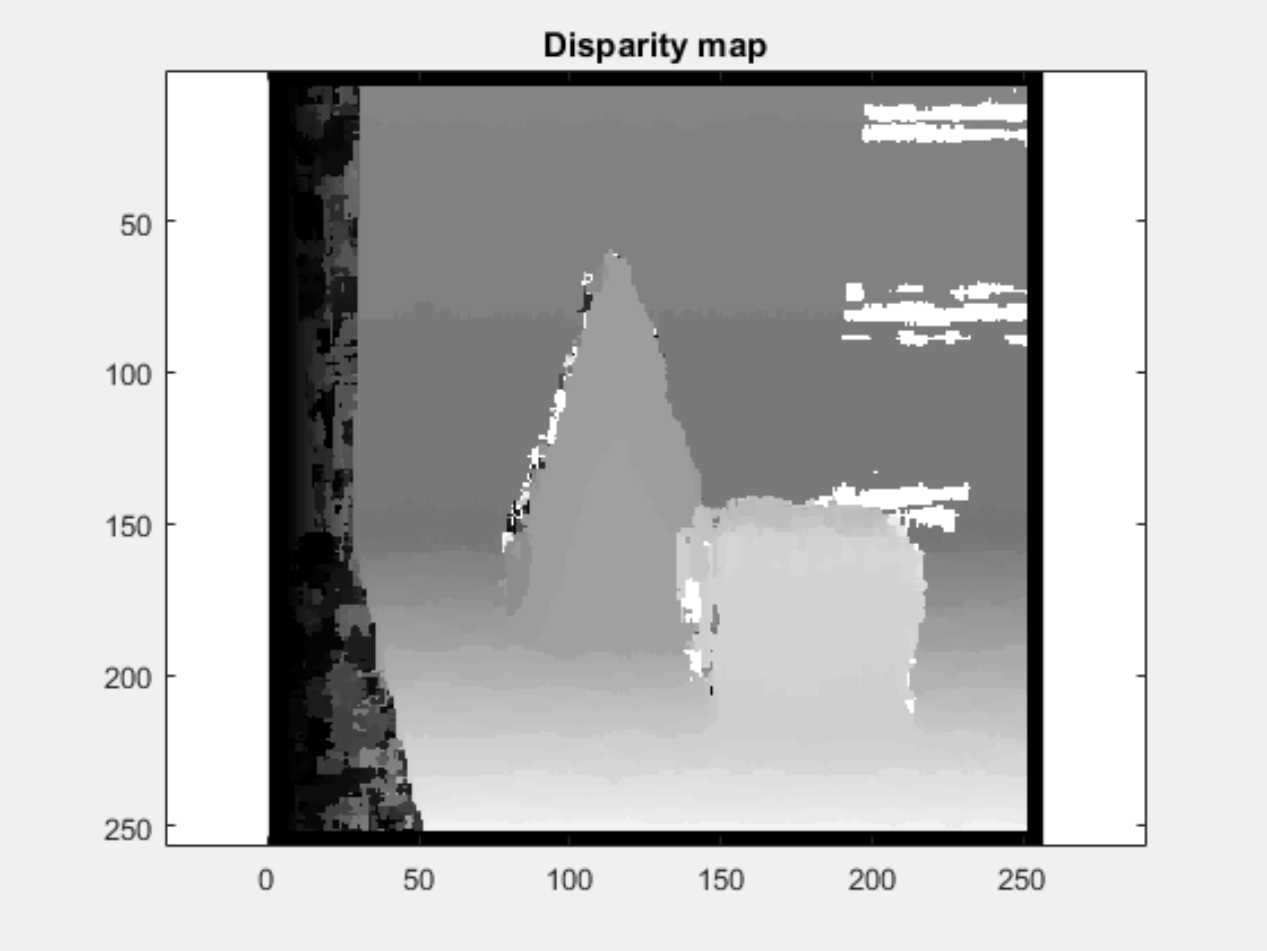




The top ten closest matches in all three approaches are very similar and match very closely with the template image we are looking for. The 500th closest match using SAD technique is not an exact match of the template, but it is a very close match to the template. The 500th closest match using SSD technique is a few pixels above the part where the template is present. The 500th closest match when using NCC template matching is a completely different part in the image. The reason this was picked up could be due to the fact that this region has a very similar color composition as the template image we are looking for, therefore the average of the pixel color values could be close to the average in our template image.

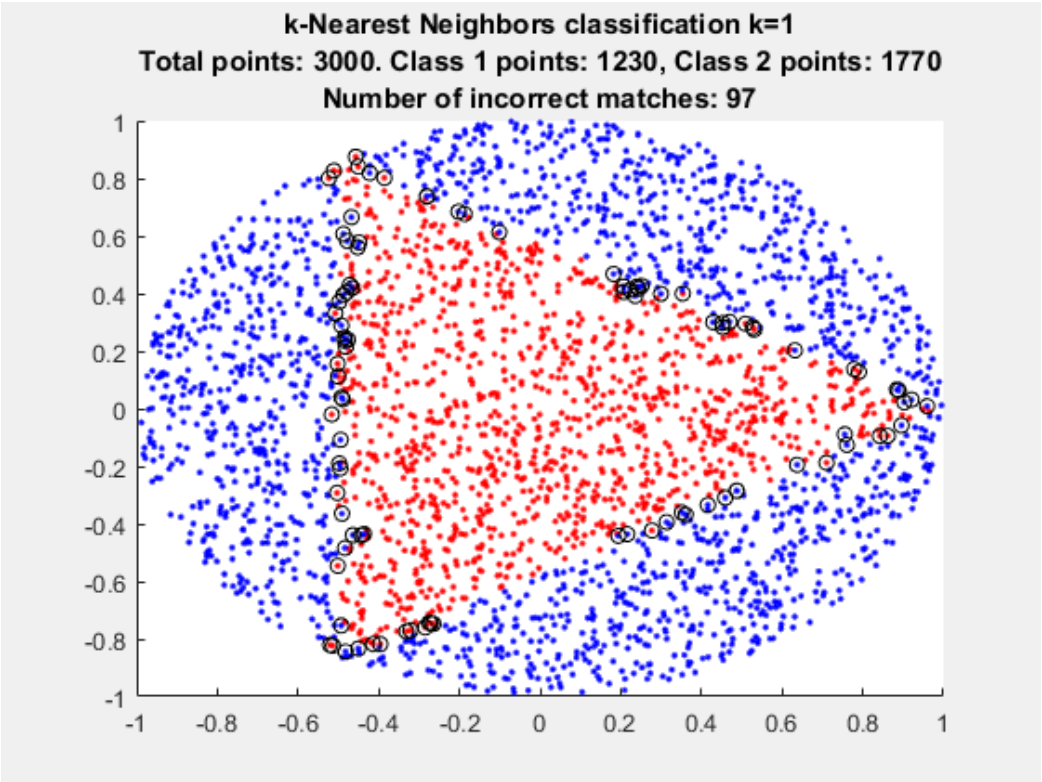
The three plots in beginning show the SAD, SSD and NCC scores for the k-th closest match. The y-axis represents the score value and the x axis represents k. We can see that for SAD and SSD the best match has the least score value, while in NCC it is the other way around.

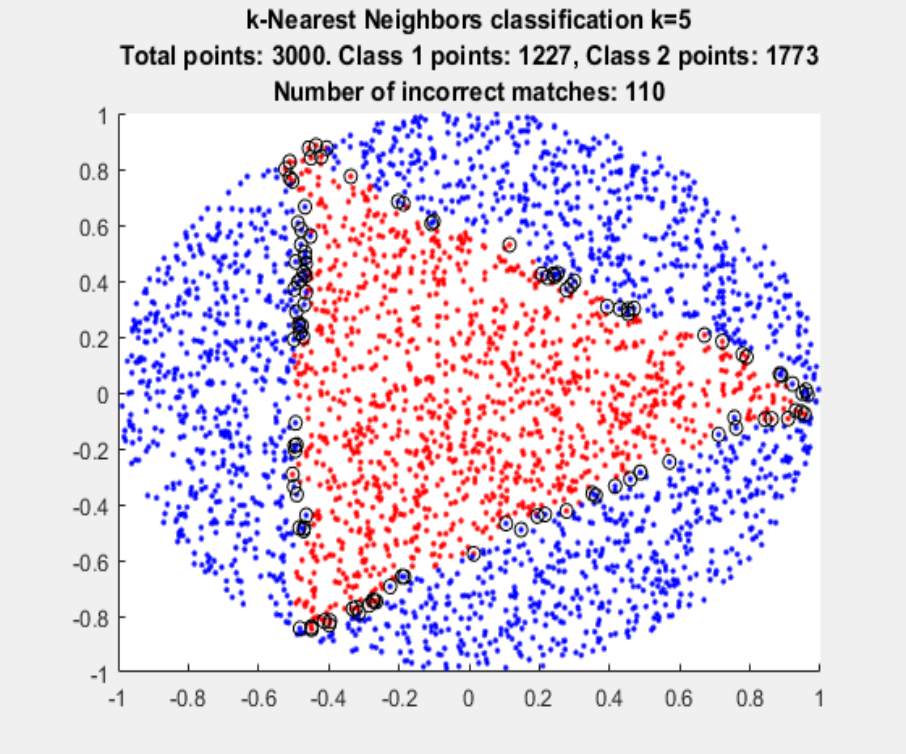
Stereo/Disparity:
2) Compute a disparity map for the images left.png and right.png (having parallel optical axes) using the basic stereo matching algorithm. Use your NCC function to perform the template matching for each patch in the left image searching in the right image (search only leftward from the starting point along each row!), and use a window size of 11x11 pixels. Use the following code to display the disparity map D with a gray colormap and clip the disparity values at 50 pixels.

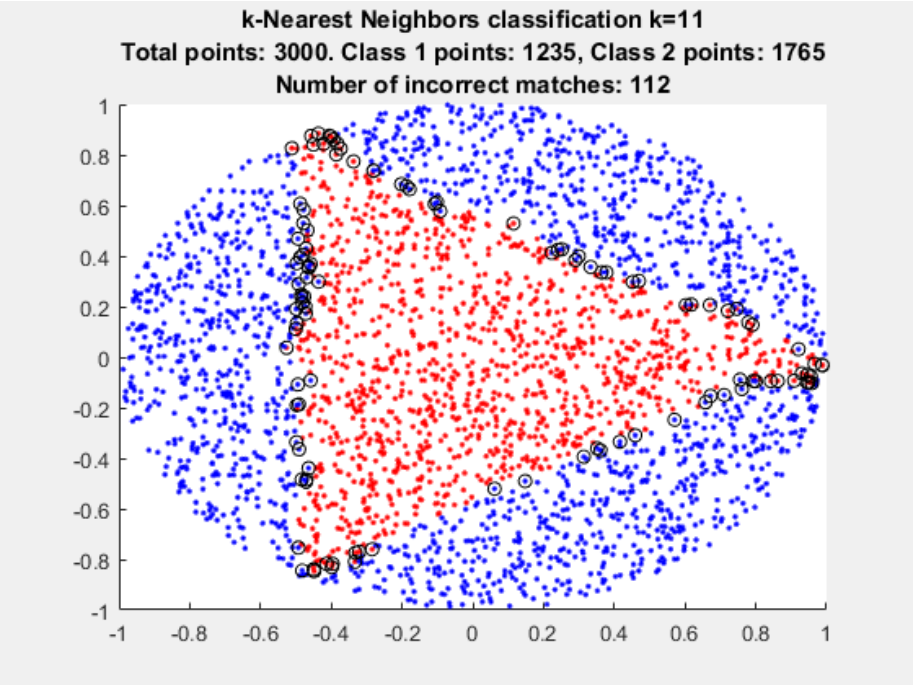


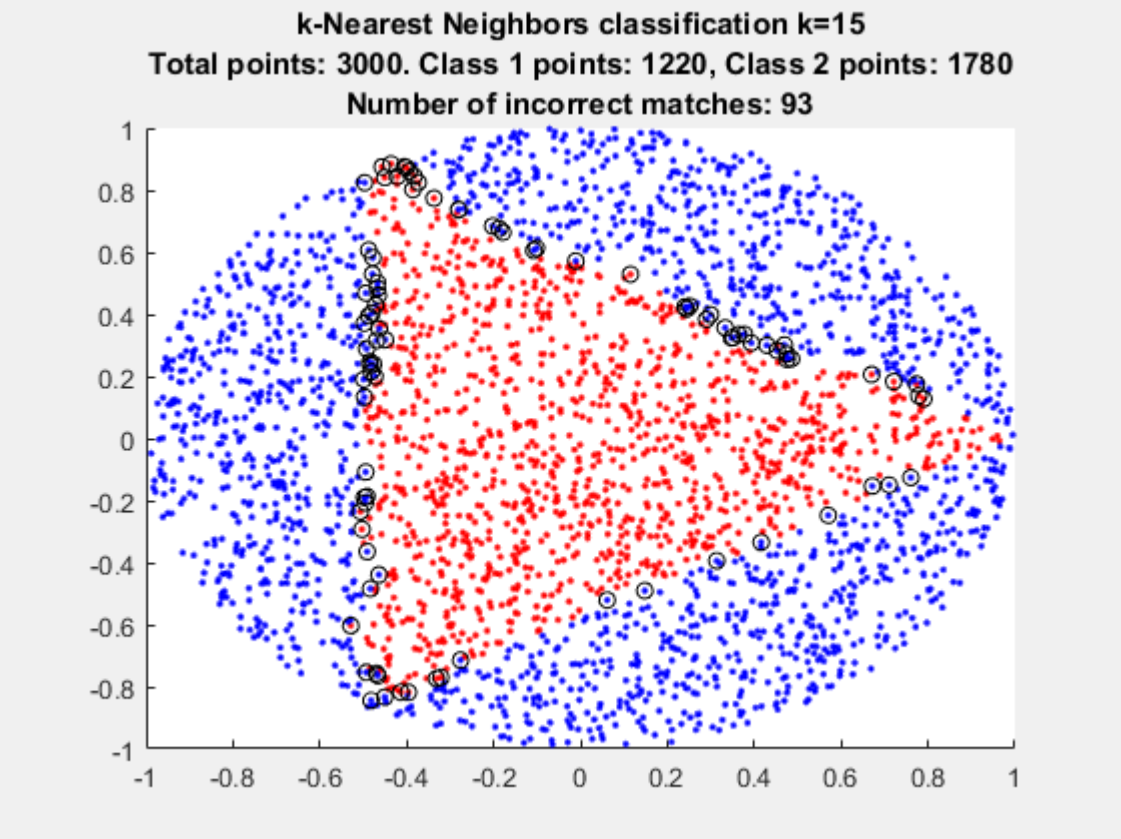
The darker regions in the above image are relatively deeper than the lighter regions of the image. We can see that the cube is present in front of the cone, which is true in the given two input images.

Classification:
3) Write an implementation of the simple k-Nearest Neighbors (kNN) algorithm to classify data points.









We can see from the above images that only the points along the border the triangular region has been classified incorrectly by the k-nearest neighbors classification algorithm. This is because these points points have both class 1 points and class 2 points as neighbors. For the k values of 1, 5, 11, 15, we can see that the number of incorrect classifications is the least for k=15, however it should be noted that k=1 comes a close second with 97 incorrect classifications. We can clearly see that increasing the value of k doesn't necessarily increase the accuracy.

Code

```
==> disparity_map.m <==
function [disp_map] = disparity_map(left_image, right_image, window_width, window_height)
    [height, width] = size(left_image);
    half_width = floor(window_width/2);
    half_height = floor(window_height/2);
    start_x = ceil(window_width/2);
    end_x = width - half_width;
    start_y = ceil(window_height/2);
    end_y = height - half_height;
    disp_map = zeros(height, width);
    for x=start_x:end_x
        for y=start_y:end_y
            template = left_image(y-half_height:y+half_height, x-half_width:x+half_width);
            scan_line = right_image(y-half_height:y+half_height, 1:x+half_width);
            [~, points_and_scores] = ncc_template_matching(scan_line, template);
            match_x = points_and_scores(1, 1) + half_width;
            disp_map(y,x) = (x-match_x);
        end
    end
end
==> HW9.m <==
mkdir out;
close all; clear; clc;

closest_matches = [1 2 5 10 100 500];
search_img = double(imread('input\search.png'));
template = double(imread('input\template.png'));
[t_height, t_width, ~] = size(template);
patches = zeros (t_height, t_width, 3, size(closest_matches, 2));
scores = zeros(size(closest_matches, 2), 1);

% Problem 1 - SAD
[sad, points_and_scores] = sad_template_matching(search_img, template);
i = 1;
for n=closest_matches
    nth_match = points_and_scores(n, :);
    x =  nth_match(1, 1);
    y =  nth_match(1, 2);
    patch = search_img(y:y+t_height-1, x:x+t_width-1, :);
    patches(:,:,:, i) = patch;
    scores(i) = nth_match(1, 3);
    i = i+1;
end
for n=1:size(closest_matches, 2)
```

```
    figure;imagesc(patches(:, :, :, n)/255); title(sprintf('%d closest match, SAD score: %g',
closest_matches(n), scores(n)));save_current_frame(fullfile('out', sprintf('sad-%d-closest-
match.png', closest_matches(n))));
end
figure;plot(points_and_scores(:, 3));xlabel('k');ylabel('SAD');title('SAD score for kth closest
match');save_current_frame(fullfile('out', 'sad-plot.png'));
pause;

% Problem 1 - SSD
close all;
i = 1;
[ssd, points_and_scores] = ssd_template_matching(search_img, template);
for n=closest_matches
    nth_match = points_and_scores(n, :);
    x =  nth_match(1, 1);
    y =  nth_match(1, 2);
    patch = search_img(y:y+t_height-1, x:x+t_width-1, :);
    patches(:,:,:, i) = patch;
    scores(i) = nth_match(1, 3);
    i = i+1;
end
for n=1:size(closest_matches, 2)
    figure;imagesc(patches(:, :, :, n)/255); title(sprintf('%d closest match, SSD score: %g',
closest_matches(n), scores(n)));save_current_frame(fullfile('out', sprintf('ssd-%d-closest-
match.png', closest_matches(n))));
end
figure;plot(points_and_scores(:, 3));xlabel('k');ylabel('SSD');title('SSD score for kth closest
match');save_current_frame(fullfile('out', 'ssd-plot.png'));
pause;
close all;

% Problem 1 - NCC
i = 1;
[ncc, points_and_scores] = ncc_template_matching(search_img, template);
for n=closest_matches
    nth_match = points_and_scores(n, :);
    x =  nth_match(1, 1);
    y =  nth_match(1, 2);
    patch = search_img(y:y+t_height-1, x:x+t_width-1, :);
    patches(:,:,:, i) = patch;
    scores(i) = nth_match(1, 3);
    i = i+1;
end
for n=1:size(closest_matches, 2)
```

```
    figure;imagesc(patches(:, :, :, n)/255); title(sprintf('%d closest match, NCC score: %g',
closest_matches(n), scores(n)));save_current_frame(fullfile('out', sprintf('ncc-%d-closest-
match.png', closest_matches(n))));
end
figure;plot(points_and_scores(:, 3));xlabel('k');ylabel('SSD');title('NCC score for kth closest
match');save_current_frame(fullfile('out', 'ncc-plot.png'));
pause;
close all;


% Problem 2 - disparity map
clear; close all; clc;
left_image = double(imread('input\left.png'));
right_image = double(imread('input\right.png'));
disp('Computing disparity map');
disp('Now we wait...');
disp_map = disparity_map(left_image, right_image, 11, 11);
figure;imagesc(disp_map, [0 50]); axis equal;colormap gray; title('Disparity map');
save_current_frame(fullfile('out', 'disparity_map.png'));
pause;

% Problem 3 - classification
clear; clc; close all;
training_data = load ('input\train.txt');
test_data = load ('input\test.txt');

X = training_data(:, 1:2);
Y = test_data(:, 1:2);
values = [1, 5, 11, 15];
ground_truth = test_data(:, 3);
for k=values
    [idx, dist] = knnsearch(X, Y, 'K', k);
    k_matches = zeros(size(Y, 1), k);
    for i=1:k
        k_matches(:, i) = training_data(idx(:,i), 3);
    end
    if k ~= 1
        classified_labels = mode(k_matches')';
    else
        classified_labels = k_matches;
    end
    class_1 = numel(find(classified_labels == 1));
    class_2 = numel(find(classified_labels == 2));
    mis_matched_points = test_data(ground_truth ~= classified_labels, 1:2);
    class_1_matches = test_data(classified_labels == 1, 1:2);
    class_2_matches = test_data(classified_labels == 2, 1:2);
```

```
    figure;
    hold on;
    title({sprintf('k-Nearest Neighbors classification k=%d', k), sprintf('Total points: %d. Class 1
points: %d, Class 2 points: %d', size(test_data,1), class_1, class_2), sprintf('Number of incorrect
matches: %d', size(mis_matched_points,1))});
    plot(class_1_matches(:, 1), class_1_matches(:, 2), 'r.');
    plot(class_2_matches(:, 1), class_2_matches(:, 2), 'b.');
    plot(mis_matched_points(:, 1), mis_matched_points(:, 2), 'ko');
    hold off;
    save_current_frame(fullfile('out', sprintf('%d-nearest-plot.png',k)));
    fprintf('k: %d, Number of mismatches: %d\n', k, numel(find(ground_truth ~=
classified_labels)));
end
pause;
close all;
==> ncc_template_matching.m <==
function [ncc, point_and_ncc_scores] = ncc_template_matching(search_img, template)
    [t_height, t_width, ~] = size(template);
    t_half_width = floor(t_width/2);
    t_half_height = floor(t_height/2);
    t_center_x = ceil(t_width/2);
    t_center_y = ceil(t_height/2);
    [s_height, s_width, ~] = size(search_img);
    ncc = zeros(s_height, s_width);
    for x=t_center_x:(s_width-t_half_width)
        for y=t_center_y:(s_height-t_half_height)
            patch = search_img(y-t_half_height:y+t_half_height, x-t_half_width:x+t_half_width, :);
            for j=1:size(template, 3)
                template_color = template(:, :, j);
                patch_color = patch(:, :, j);
                template_avg = mean(template_color(:));
                patch_avg = mean(patch_color(:));
                ncc(y,x) = ncc(y,x) + sum(sum((patch_color - patch_avg).*(template_color -
template_avg)/(std(patch_color(:))*std(template_color(:))))/(t_height*t_width - 1);
            end
        end
    end
    ncc_excluding_borders = ncc(t_center_y:(s_height-t_half_height), t_center_x:(s_width-
t_half_width));
    [sorted_values, sorted_index] = sort(ncc_excluding_borders(:), 'descend');
    [y, x] = ind2sub(size(ncc_excluding_borders), sorted_index);
    point_and_ncc_scores = [x y sorted_values];
end

==> sad_template_matching.m <==
function [sad, point_and_sad_scores] = sad_template_matching(search_img, template)
```

```
    [t_height, t_width, ~] = size(template);
    t_half_width = floor(t_width/2);
    t_half_height = floor(t_height/2);
    t_center_x = ceil(t_width/2);
    t_center_y = ceil(t_height/2);
    [s_height, s_width, ~] = size(search_img);
    sad = zeros(s_height, s_width);

    for x=t_center_x:(s_width-t_half_width)
        for y=t_center_y:(s_height-t_half_height)
            sad(y,x) = sum(sum(sum(abs(search_img(y-t_half_height:y+t_half_height, x-
t_half_width:x+t_half_width, :) - template))));
        end
    end
    sad_excluding_borders = sad(t_center_y:(s_height-t_half_height), t_center_x:(s_width-
t_half_width));
    [sorted_values, sorted_index] = sort(sad_excluding_borders(:));
    [y, x] = ind2sub(size(sad_excluding_borders), sorted_index);
    point_and_sad_scores = [x y sorted_values];
end
==> save_current_frame.m <==
function [] = save_current_frame (outputFileName)
% saves whatever is displayed in the window to the given file. useful when
% there is dynamic text rendered on top of the image and that should also
% be saved to the file
  fig = gcf;
  frame = getframe(fig);
  imwrite(frame.cdata, outputFileName);
end
==> scratchpad.m <==
clear; clc; close all;
training_data = load ('input\train.txt');
test_data = load ('input\test.txt');

X = training_data(:, 1:2);
Y = test_data(:, 1:2);
values = [1, 5, 11, 15];
ground_truth = test_data(:, 3);
for k=values
    [idx, dist] = knnsearch(X, Y, 'K', k);
    k_matches = zeros(size(Y, 1), k);
    for i=1:k
        k_matches(:, i) = training_data(idx(:,i), 3);
    end
    if k ~= 1
        classified_labels = mode(k_matches')';
```

```
    else
        classified_labels = k_matches;
    end
    class_1 = numel(find(classified_labels == 1));
    class_2 = numel(find(classified_labels == 2));
    mis_matched_points = test_data(ground_truth ~= classified_labels, 1:2);
    class_1_matches = test_data(classified_labels == 1, 1:2);
    class_2_matches = test_data(classified_labels == 2, 1:2);
    figure;
    hold on;
    title({sprintf('k-Nearest Neighbors classification k=%d', k), sprintf('Total points: %d. Class 1
points: %d, Class 2 points: %d', size(test_data,1), class_1, class_2), sprintf('Number of incorrect
matches: %d', size(mis_matched_points,1))});
    plot(class_1_matches(:, 1), class_1_matches(:, 2), 'r.');
    plot(class_2_matches(:, 1), class_2_matches(:, 2), 'b.');
    plot(mis_matched_points(:, 1), mis_matched_points(:, 2), 'ko');
    hold off;
    save_current_frame(fullfile('out', sprintf('%d-nearest-plot.png',k)));
    fprintf('k: %d, Number of mismatches: %d\n', k, numel(find(ground_truth ~=
classified_labels)));
end
pause;
close all;
==> ssd_template_matching.m <==
function [ssd, point_and_ssd_scores] = ssd_template_matching(search_img, template)
    [t_height, t_width, ~] = size(template);
    t_half_width = floor(t_width/2);
    t_half_height = floor(t_height/2);
    t_center_x = ceil(t_width/2);
    t_center_y = ceil(t_height/2);
    [s_height, s_width, ~] = size(search_img);
    ssd = zeros(s_height, s_width);

    for x=t_center_x:(s_width-t_half_width)
        for y=t_center_y:(s_height-t_half_height)
            ssd(y,x) = sum(sum(sum((search_img(y-t_half_height:y+t_half_height, x-
t_half_width:x+t_half_width, :) - template).^2)));
        end
    end
    ssd_excluding_borders = ssd(t_center_y:(s_height-t_half_height), t_center_x:(s_width-
t_half_width));
    [sorted_values, sorted_index] = sort(ssd_excluding_borders(:));
    [y, x] = ind2sub(size(ssd_excluding_borders), sorted_index);
    point_and_ssd_scores = [x y sorted_values];
end
```