**(Bu notlardan sınavlarda sorumlu değilsiniz. Ek bilgi olarak verilmiştir.)**

**Registering an Image Using Normalized Cross-Correlation**

Sometimes one image is a subset of another. Normalized cross-correlation can be used to determine how to register or align the images by translating one of them.

|  |
| --- |
|  |

**Overview of Example**

**Step 1: Read Images**

onion = imread('onion.png');

peppers = imread('peppers.png');

imshow(onion)

figure, imshow(peppers)



**Step 2: Choose Subregions of Each Image**

It is important to choose regions that are similar. The image sub\_onion will be the template, and must be smaller than the image sub\_peppers. You can get these sub regions using either the non-interactive script below *or* the interactive script.

% non-interactively

rect\_onion = [111 33 65 58];

rect\_peppers = [163 47 143 151];

sub\_onion = imcrop(onion,rect\_onion);

sub\_peppers = imcrop(peppers,rect\_peppers);

% OR

% interactively

[sub\_onion,rect\_onion] = imcrop(onion); % choose the pepper below the onion

[sub\_peppers,rect\_peppers] = imcrop(peppers); % choose the whole onion

% display sub images

figure, imshow(sub\_onion)

figure, imshow(sub\_peppers)

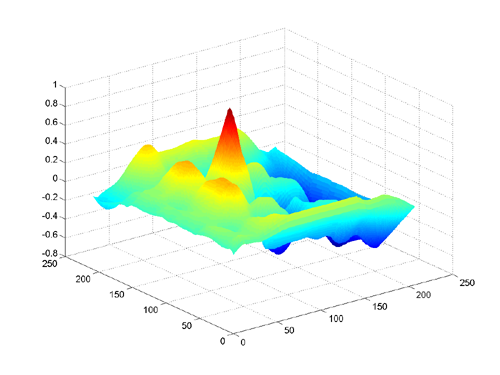


**Step 3: Do Normalized Cross-Correlation and Find Coordinates of Peak**

Calculate the normalized cross-correlation and display it as a surface plot. The peak of the cross-correlation matrix occurs where the sub\_images are best correlated. normxcorr2 only works on grayscale images, so we pass it the red plane of each sub image.

c = normxcorr2(sub\_onion(:,:,1),sub\_peppers(:,:,1));

figure, surf(c), shading flat



**Step 4: Find the Total Offset Between the Images**

The total offset or translation between images depends on the location of the peak in the cross-correlation matrix, and on the size and position of the sub images.

% offset found by correlation

[max\_c, imax] = max(abs(c(:)));

[ypeak, xpeak] = ind2sub(size(c),imax(1));

corr\_offset = [(xpeak-size(sub\_onion,2))

               (ypeak-size(sub\_onion,1))];

% relative offset of position of subimages

rect\_offset = [(rect\_peppers(1)-rect\_onion(1))

               (rect\_peppers(2)-rect\_onion(2))];

% total offset

offset = corr\_offset + rect\_offset;

xoffset = offset(1);

yoffset = offset(2);

**Step 5: See if the Image Onion Was Extracted from the Image Peppers**

Figure out where onion falls inside of peppers.

xbegin = xoffset+1;

xend = xoffset+ size(onion,2);

ybegin = yoffset+1;

yend = yoffset+size(onion,1);

% extract region from peppers and compare to onion

extracted\_onion = peppers(ybegin:yend,xbegin:xend,:);

if isequal(onion,extracted\_onion)

   disp('onion.png was extracted from peppers.png')

end

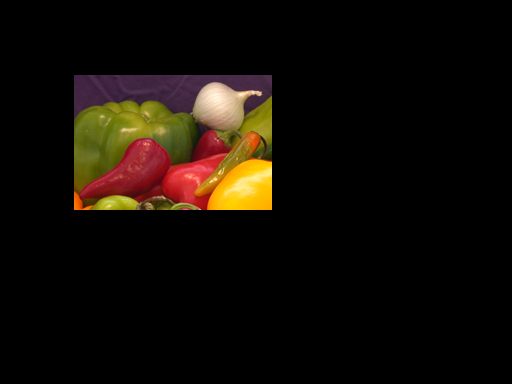
**Step 6: Pad the Onion Image to the Size of the Peppers Image**

Pad the onion image to overlay on peppers, using the offset determined above.

recovered\_onion = uint8(zeros(size(peppers)));

recovered\_onion(ybegin:yend,xbegin:xend,:) = onion;

figure, imshow(recovered\_onion)



**Step 7: Transparently Overlay Onion Image on Peppers Image**

Create transparency mask to be opaque for onion and semi-transparent elsewhere.

[m,n,p] = size(peppers);

mask = ones(m,n);

i = find(recovered\_onion(:,:,1)==0);

mask(i) = .2; % try experimenting with different levels of

              % transparency

% overlay images with transparency

figure, imshow(peppers(:,:,1)) % show only red plane of peppers

hold on

h = imshow(recovered\_onion); % overlay recovered\_onion

set(h,'AlphaData',mask)

