Multiscale Optical Flow Estimation: Estimate Flow

In this project you will extend your previous optical flow solution to a multiscale approach. A multiscale approach allows motions of various magnitudes to be estimated with the same kernel size.

To complete this lab you will (1) write a function to warp an image according to the optical flow estimate and (2) incorporate the image warping function in a pyramidal (multiscale) optical flow estimation function.

Your Script

Save Reset MATLAB Documentation (https://www.mathworks.com/help/)

```
v = VideoReader('shuttle.avi');
fr1 = readFrame(v);
im1t = im2double(rgb2gray(fr1));

hasFrame(v);
fr2 = readFrame(v); fr2 = readFrame(v);
im2t = im2double(rgb2gray(fr2));

[u,v] = multiscale_flow(im1t,im2t);

figure()
subplot(221)
imshow(im1t)
subplot(222)
imshow(im2t)
subplot(223)
imagesc(u)
subplot(224)
imagesc(v)

function [u,v] = multiscale_flow(I1,I2)
    % The number of pyramid levels will be determined by the image size
    % At the highest pyramid level the smallest image dimension will be around
    % 30 pixels.
    lmax = round(log2(min(size(I1))/30));
    % The pyramidal approach can be implemented with a recursive strategy
    [u,v] = multiscale_aux(I1,I2,1,lmax);
end

function [u,v] = multiscale_aux(I1,I2,l,lmax)
    % Downsample the images by half using imresize and bicubic interpolation.
    % Use your gauss_blur function to smooth the result.
    I1_ = gauss_blur(imresize(I1,.5,'bicubic'));
    I2_ = gauss_blur(imresize(I2,.5,'bicubic'));
    % If the highest pyramid level has been reached, estimate the optical flow
    % on the downsampled images with your estimate_flow function using 2 as the wsize parameter.
    if l == lmax
        [u,v] = estimate_flow(I1_,I2_,2);
    % If we are beyond the highest pyramid level, estimate the optical flow
    % on the input images (not the downsampled images) with your estimate_flow function
    % using 2 as the wsize parameter.
    elseif l > lmax
        l = lmax;
        [u,v] = estimate_flow(I1,I2,2);
    % Otherwise, increment the current level and continue up the pyramid (i.e. recurse)
    % using the downsampled images.
    else
        [u,v] = multiscale_aux(I1_,I2_,l+1,lmax);
    end
    % After flow has been estimated at the current level, pass this estimate along with
    % the input images (not the downsampled images) to multiscale_down for iterative
    % improvement of the flow estimate.
    if l==0
```

```
        display('hi');
    end
    [u,v] = multiscale_down(I1,I2,u,v,l);

end

function [u,v] = multiscale_down(I1,I2,u,v,l)
    % If the base pyramid level has been reached, return.
    if l == 0
        return
    end
    % Otherwise, upsample the previous flow estimate by a factor of 2 using imresize with
    % bicubic interpolation. The flow values should be doubled.

    u = 2*imresize(u,2,'bicubic');
    v = 2*imresize(v,2,'bicubic');

    % Warp the input image, I2, according to the upsampled flow estimate.
    I2_ = warp_image(I2,u,v);
    % Estimate the incremental flow update using your estimate_flow function and the warped
    % input image with 2 as the wsize parameter.
    [u_,v_] = estimate_flow(I1,I2_,2);
    % Update the flow estimate by adding the incremental estimate above to the previous estimate.
    u = u_ + u;
    v = v_ + v;

end

function warp = warp_image(I,u,v)
    %% INPUT:
    %% I: image to be warped
    %% u,v: x and y displacement
    %% OUTPUT:
    %% warp: image I deformed by u,v

    % initialize warp as zeros
    warp = zeros(size(I));
    % construct warp so that warp(x,y) = I(x + u, y + v)
    [r c] = size(I);

    % see https://www.mathworks.com/company/newsletters/articles/matrix-indexing-in-matlab.html
    x = 1:c;
    y = 1:r;
    [X, Y] = meshgrid(x,y);
    % add u to x, v to y
    Xq = round(X + u);
    Yq = round(Y + v);

    warp = interp2(X,Y,I,Xq,Yq);

    warp(isnan(warp)) = 0;

end

function [u,v] = estimate_flow(I1,I2,wsize)
    %% INPUT:
    %% I1, I2: nxm sequential frames of a video
    %% wsize: (wsize*2)^2 is the size of the neighborhood used for displacement estimation
    %% OUTPUT:
    %% u,v: nxm flow estimates in the x and y directions respectively

    [I2_x,I2_y] = grad2d(I2);
    % The temporal gradient is the smoothed difference image
    I2_t = gauss_blur(I1-I2);
    % initialize x and y displacement to zero
    u = zeros(size(I2));
    v = zeros(size(I2));

    % loop over all pixels in the allowable range
```

```
    % loop over all pixels in the allowable range
    % size(A,1/2) gives number of rows/columns
    for i = wsize+1:size(I2_x,1)-wsize
       for j = wsize+1:size(I2_x,2)-wsize

          % Select the appropriate window
          Ix = I2_x(i-wsize:i+wsize, j-wsize:j+wsize);
          Iy = I2_y(i-wsize:i+wsize, j-wsize:j+wsize);
          It = I2_t(i-wsize:i+wsize, j-wsize:j+wsize);

          d = estimate_displacement(Ix,Iy,It);

          u(i,j) = d(1);
          v(i,j) = d(2);
       end
    end
    % use medifilt2 with a 5x5 filter to reduce outliers in the flow estimate
    u = medfilt2(u,[5 5]);
    v = medfilt2(v,[5 5]);
end

function d = estimate_displacement(Ix,Iy,It)
    %% INPUT:
    %% Ix, Iy, It: m x m matrices, gradient in the x, y and t directions
    %% Note: gradient in the t direction is the image difference
    %% OUTPUT:
    %% d: least squares solution
    b = [ Ix(:) Iy(:) ]' * It(:);
    A = [ Ix(:) Iy(:) ]' * [ Ix(:) Iy(:) ];

    % to help mitigate effects of degenerate solutions add eye(2)*eps to the 2x2 matrix A
    % add eps value
    A = A + eye(2)*eps;

    % use pinv(A)*b to compute the least squares solution
    d = pinv(A)*b;
end

function [I_x,I_y] = grad2d(img)
        %% compute image gradients in the x direction
        %% convolve the image with the derivative filter from the lecture
        %% using the conv2 function and the 'same' option
        dx_filter = [1/2 0 -1/2];
        I_x = conv2(img, dx_filter, 'same');

        %% compute image gradients in the y direction
        %% convolve the image with the derivative filter from the lecture
        %% using the conv2 function and the 'same' option
        dy_filter = dx_filter';
        I_y = conv2(img, dy_filter, 'same');
end

function smooth = gauss_blur(img)
    %% Since the Gaussian filter is separable in x and y we can perform Gaussian smoothing by
    %% convolving the input image with a 1D Gaussian filter in the x direction then
    %% convolving the output of this operation with the same 1D Gaussian filter in the y direction.

    %% Gaussian filter of size 5
    %% the Gaussian function is defined f(x) = 1/(sqrt(2*pi)*sigma)*exp(-x.^2/(2*sigma))
    x = -2:2;
    sigma = 1;
    gauss_size = [1 5];

    % my soln: I use fspecial('gaussian', hsize = [1 5], sigma)
    %gauss_filter_x = fspecial('gaussian', gauss_size, sigma);
    %gauss_filter_y = fspecial('gaussian', gauss_size', sigma);
    %smooth_x = imfilter(img, gauss_filter_x);
    %% convolve smooth_x with the transpose of the Gaussian filter
    %smooth = imfilter(smooth_x, gauss_filter_y);
```

```
    %smooth = imfilter(smooth_x, gauss_filter_y);

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % for edX class:
    gauss_filter = 1/(sqrt(2*pi)*sigma)*exp(-x.^2/(2*sigma^2));

    %% using the conv2 function and the 'same' option
    %% convolve the input image with the Gaussian filter in the x
    smooth_x = conv2(img, gauss_filter, 'same');
    %% convolve smooth_x with the transpose of the Gaussian filter
    smooth = conv2(smooth_x, gauss_filter', 'same');
end

```

Run Script

Previous Assessment: All Tests Passed

Submit

Is the optical flow estimate correct: u?

Is the optical flow estimate correct: v?

Output