Image warping

Render an image transformed via an homography References: Andrea Fusiello, Visione Computazionale. Chapter 10 on image mosaicing

.\\°//.\\°//.\\°//.\\°//.\\°//.\\°//.\\°//.\\°//.\\°//.\\°//.\\°//.\\°//.\\°//.

Image Analysis and Computer Vision Politecnico di Milano

Luca Magri for comments and suggestions please send an email to luca.magri@polimi.it

.\\°//.\\°//.\\°//.\\°//.\\°//.\\°//.\\°//.\\°//.\\°//.\\°//.\\°//.\\°//.

Contents


```
clear;
close all;
clc;
```

We are given: - two images im1 and im2 - the homography between im1 and im2 our aim is to synthetize a novel image obtained by superimposing im2 and H(im1). Let's use the backward mapping.

```
im1 = imread("E3_data/ella1.jpg");
im2 = imread("E3_data/ella2.jpg");
im1 = imresize(im1,0.6);
im2 = imresize(im2,0.6);
data = load('E3_data/ella_homo.mat');
m1 = data.m1;
m2 = data.m2;
H = data.H;
```

overlay the transformed im1 over im2

let's use bilinear interpolation

```
canvas = im2;
lambda = 0.7;
for i=1:size(canvas,1)
    for j = 1:size(canvas,2)
        q =[j;i;1];
        % apply the inverse transformation
        p = inv(H)*q; % column is x and row is y
        p = p./p(3); % cartesian coordinate in the plane of the first image
        p = p;
        % if the preimage p lies inside the first image
        if(round(p(1))>0 && round(p(1))<=size(im1,2) && round(p(2))>0 && round(p(2))<=size(im1,1))
            for ch = 1:3
                source_value = im1(round(p(2)),round(p(1)),ch);
                % apply bilinear interpolation
                if(p(1)>1 && p(1)<size(im1,2) && p(2)>1 && p(2)<size(im1,1))
                    y1 = floor(p(2));
                    y2 = ceil(p(2));
                    % interpolate along the x axis
                    x1 = floor(p(1));
                    x2 = ceil(p(1));
                    % at y1
                    v1 = (p(1)-x1)*im1(y1,x1,ch) + (x2-p(1))*im1(y1,x2,ch);
                    % at y2
                    v2 = (p(1)-x1)*im1(y2,x1,ch) + (x2-p(1))*im1(y2,x2,ch);
                    % interpolate along the y axis
                    source_value = (p(2)-y1).*v1 + (y2-p(2)).*v2;
                end
                canvas(i,j,ch) = (1-lambda).*canvas(i,j,ch) + lambda.*source_value;
            end
        end

    end
end
```

```
figure;
imshow(canvas);
% still some misalagnment are visible
```

a bigger canvas

first we must create a canvas to display our result. The canvas must be large enough to contain both im2 and im1. let's see where the corners of im1 are mapped

```
tl = H*[0;0;1]; % where top-left is mapped
tl = tl./tl(3);
tr = H*[size(im1,2);0;1]; % where top-right is mapped
tr = tr./tr(3);
bl = H*[0;size(im1,1);1]; % where bottom-left is mapped
bl = bl./bl(3);
br = H*[size(im1,2);size(im1,1);1];
br = br./br(3); % where bottom-right is mapped

% here we display the perimeter of im1 via H
figure;
imshow(canvas);
hold all;
plot(tl(1),tl(2),'ro','MarkerSize',10,'MarkerFaceColor','r');
plot(tr(1),tr(2),'ro','MarkerSize',10,'MarkerFaceColor','r');
```

```
plot(bl(1),bl(2),'ro','MarkerSize',10,'MarkerFaceColor','r');
plot(br(1),br(2),'ro','MarkerSize',10,'MarkerFaceColor','r');
line([tl(1),tr(1)],[tl(2),tr(2)],'Color','r','LineWidth',3);
line([tr(1),br(1)],[tr(2),br(2)],'Color','r','LineWidth',3);
line([br(1),bl(1)],[br(2),bl(2)],'Color','r','LineWidth',3);
line([bl(1),tl(1)],[bl(2),tl(2)],'Color','r','LineWidth',3);
```

determine the dimension of the canvas

```
w_max = max([size(im2,2),tl(1),tr(1),bl(1),br(1)]);
w_min = min([[size(im2,2),tl(1),tr(1),bl(1),br(1)]]);
w = ceil(w_max-w_min);
h_max = max([size(im2,1),tl(2),tr(2),bl(2),br(2)]);
h_min = min([[size(im2,1),tl(2),tr(2),bl(2),br(2)]]);
h = ceil(h_max-h_min);
big_canvas = uint8(255.*ones(h,w,3));
big_canvas(-h_min+1:-h_min+size(im2,1), -w_min+1:-w_min+size(im2,2),:) = im2;

figure;
imshow(big_canvas);
```

```
Warning: Integer operands are required for colon operator when used as index.
Warning: Integer operands are required for colon operator when used as index.
```

render the novel image using bilinear interpolation

```
for i=1:size(big_canvas,1)
    for j = 1:size(big_canvas,2)
        % bring [j;i;1] from the reference system of the big_canvas to the
        % reference system of im2
        q =[j+w_min;i+h_min;1];
        % apply the inverse transformation
        p = inv(H)*q; % column is x and row is y
        p = p./p(3); % cartesian coordinate in the plane of the first image
        % if the preimage p lies inside the first image
        if(round(p(1))>0 && round(p(1))<=size(im1,2) && round(p(2))>0 && round(p(2))<=size(im1,1))
            for ch = 1:3
                source_value = im1(round(p(2)),round(p(1)),ch);
```

```
                % apply bilinear interpolation
                if(p(1)>1 && p(1)<size(im1,2) && p(2)>1 && p(2)<size(im1,1))
                    y1 = floor(p(2));
                    y2 = ceil(p(2));
                    % interpolate along the x axis
                    x1 = floor(p(1));
                    x2 = ceil(p(1));
                    % at y1
                    v1 = (p(1)-x1)*im1(y1,x1,ch) + (x2-p(1))*im1(y1,x2,ch);
                    % at y2
                    v2 = (p(1)-x1)*im1(y2,x1,ch) + (x2-p(1))*im1(y2,x2,ch);
                    % interpolate along the y axis
                    source_value = (p(2)-y1).*v1 + (y2-p(2)).*v2;
                end
                big_canvas(i,j,ch) = (1-lambda).*big_canvas(i,j,ch) + lambda.*source_value;
            end % end for channels
        end % end if inside the image
    end % end for j
end % end for i
```

```
figure;
imshow(big_canvas);
```

Published with MATLAB® R2021b