CSE578 Computer Vision	Homography
April 15, 2016	Abhinav Moudgil, 201331039

Direct Linear Transformation

Derivation

The Direct Linear Transform (DLT) algorithm is a simple algorithm used to solve for the homography matrix H given a sufficient set of point correspondences.
Since we are working in homogeneous coordinates, the relationship between two corresponding points x and x' can be re-written as

$$c \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = H \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \tag{1}$$

where c is a non zero constant and,

$$H = \begin{bmatrix} h_1 & h_2 & h_3 \\ h_4 & h_5 & h_6 \\ h_7 & h_8 & h_9 \end{bmatrix} \tag{2}$$

Dividing the first row of equation by the third row and the second row by the third row we get the following two equations:

$$h_1 x - h_2 y - h_3 + (h_7 x + h_8 y + h_9)u = 0$$

$$h_4 x - h_5 y - h_6 + (h_7 x + h_8 y + h_9)u = 0$$

They can be written in matrix form as:

$$A_i h = 0$$

where,

$$A_i = \begin{bmatrix} -x & -y & 1 & 0 & 0 & 0 & ux & uy & u \\ 0 & 0 & 0 & -x & -y & -1 & vx & vy & v \end{bmatrix} \tag{3}$$

and,

$$h = \begin{bmatrix} h_1 & h_2 & h_3 & h_4 & h_5 & h_6 & h_7 & h_8 & h_9 \end{bmatrix}^T \tag{4}$$

Here H has 8 degrees of freedom and hence we require atleast 4 correspondences to solve the equation. The solution to H is obtained by decomposing A using SVD. The solution is the last coloumn of V.

Using SURF features

This requires finding matching features in the two images. This was achieved by detecting SURF features in both the images. These features were matched in both the images by using sun of squares distance (SSD). The n best matching features were extracted.

```
function [matched_points1, matched_points2] = surfFeatures(im1, im2, n)
    points1 = detectSURFFeatures(im1);
    points2 = detectSURFFeatures(im2);
    [f1, vpnts1] = extractFeatures(im1, points1);
    [f2, vpnts2] = extractFeatures(im2, points2);
    [indexPairs, distance] = matchFeatures(f1, f2);
    [smallest4, bestIndices] = getNElements(distance, n);
    indexPairs = indexPairs(bestIndices, :);
    matched_points1 = vpnts1(indexPairs(:, 1));
    matched_points2 = vpnts2(indexPairs(:, 2));
end
```

```
close all;
clear;
im1 = rgb2gray(imread('../images/im1.jpg'));
im1 = imresize(im1, [480 640]);
figure, imshow(im1);
im2 = imresize(imrotate(im1,-20), 1.2);
figure, imshow(im2);
n = 15;
[matched_points1, matched_points2] = surfFeatures(im1, im2, n);
x = [matched_points1.Location ones(n,1)];
x_i = [matched_points2.Location ones(n, 1)];
A = zeros(2 * n, 9);
for i = 1:n
    A(2 * i - 1, :) = [zeros(1,3) -x_i(i, :) x(i,2) * x_i(i, :)];
    A(2 * i, :) = [x_i(i, :) zeros(1, 3) -x(i,1) * x_i(i, :)];
end
[U, D, V] = svd(A);
H = V(:, 9);
H = reshape(H, [3, 3]);

% Transform the image according to the obtained H matrix
T = maketform('projective', H);
t_img = imtransform(im2, T, 'Size', size(im2));
figure, imshow(t_img);
```

Using SIFT Features

```
function [matched_points1, matched_points2] = siftMatch(im1, im2, n)
    I = single(im1);
    J = single(im2);

    [F1, D1] = vl_sift(I);
```

(a) Original (b) Transformed

Figure 1: Input Images

```
    [F2, D2] = vl_sift(J);

    % Where 1.5 = ratio between euclidean distance of NN2/NN1
    [matches, score] = vl_ubcmatch(D1,D2,1.5);
    [smallestElements, smallestIndices] = getNElements(score, n);
    matches = matches(:, smallestIndices);
    matched_points1 = [F1(1,matches(1,:));F1(2,matches(1,:))]';
    matched_points2 = [F2(1,matches(2,:));F2(2,matches(2,:))]';
end
```

```
function H = findHomographyDLT(im1, im2)

    % Number of matching points
    n = 15;
    [matched_points1, matched_points2] = siftMatch(im1, im2, n);
    x = [matched_points1 ones(n, 1)];
    xi = [matched_points2 ones(n, 1)];
    A = zeros(2*n, 9);
    for i = 1:n
        A(2*i-1:2*i, :) = [zeros(1, 3) -xi(i, :) x(i, 2)*xi(i,:);
                           xi(i, :) zeros(1, 3) -x(i, 1)*xi(i, :)];
    end
    [u,s,v] = svd(A);
    H = reshape(v(:, 9), [3, 3]);
end
```

(a) n = 4 (b) n = 15

Figure 2: DLT Results using SURF Features

RANSAC

RANSAC is used to find homography H by randomly sampling from a give set of features and the best is selected. The best H is characterized by the minimum number of outliers created using the given matrix.

After each iteration, errors are caluculated between the projected points using H and the original points. A point is considered an outlier if the corresponding error value is greater than a given threshold.
The threshold is calculated to be the mean value of the errors for the points used to calculate H for the current iteration.

```
function [best_H] = findHomographyRANSAC(im1, im2)
    N = 50;
    [matched_points1, matched_points2] = siftMatch(im1, im2, N);
    all_x = [matched_points1 ones(N, 1)];
    all_x_i = [matched_points2 ones(N, 1)];

    min_outliers = N;
    best_H = [];
    for iterations = 1:50
        % Randomly picking 5 indices
        n = 15;
        r = randi([1 N],1,n);
        while length(unique(r)) ~= length(r)
            r = randi([1 N],1,n);
        end

```

(a) n = 5 (b) n = 15

Figure 3: DLT Results using SIFT Features

```
        % Implement DLT on the chosen points
        x = all_x(r, :);
        x_i = all_x_i(r, :);

        A = zeros(2 * n, 9);
        for i = 1:n
            A(2 * i - 1, :) = [zeros(1,3) -x_i(i, :) x(i,2) * x_i(i, :)];
            A(2 * i, :) = [x_i(i, :) zeros(1, 3) -x(i,1) * x_i(i, :)];
        end
        [U, D, V] = svd(A);
        H = V(:, 9);
        H = reshape(H, [3, 3]);

        x_predicted = (H * all_x_i')';
        for i = 1:N
            x_predicted(i, 1) = x_predicted(i, 1) ./ x_predicted(i, 3);
            x_predicted(i, 2) = x_predicted(i, 2) ./ x_predicted(i, 3);
        end

        error = abs(x_predicted(:, [1, 2]) - all_x(:, [1, 2]));
        norms = sqrt(sum(error.^2,2));

        % Taking the threshold as the mean of the errors for the seleted
    points
        thresh = mean(norms(r));
        difference = norms - thresh;
        outliers = sum(difference > 0);

        if outliers < min_outliers
            min_outliers = outliers;
```

```
                best_H = H;
            end
        end
    end
```

(a) n = 5 (b) n = 15

Figure 4: RANSAC results obtained using SIFT Features

As shown in the results, in case of the RANSAC algorithm, even by choosing a lesser numberof correspondences to calculate the homography matrix, the results are better compared to DLT.

Using manually marked points

```
close all;
clear;
im1 = rgb2gray(imread('../images/im1.jpg'));
im1 = imresize(im1, [480 640]);
im2 = imresize(imrotate(im1,-20), 1.2);
n = 10;
% Read saved points
x = dlmread('marked_original.txt');
x_i = dlmread('marked_skewed.txt');

% Select the first 6 points and compute DLT
n = 6;
A = zeros(2 * n, 9);
for i = 1:n
    A(2 * i - 1, :) = [zeros(1,3) -x_i(i, :) x(i,2) * x_i(i, :)];
    A(2 * i, :) = [x_i(i, :) zeros(1, 3) -x(i,1) * x_i(i, :)];
```

```
end
[U, D, V] = svd(A);
H = V(:, 9);
H = reshape(H, [3, 3]);

% Transform the image according to the obtained H matrix
T = maketform('projective', H);
t_img = imtransform(im2, T, 'Size', size(im2));
imshow(t_img);
```

(a) n = 5 (b) n = 10

Figure 5: DLT results obtained using manually marked points

Mosaicing

Two images having sufficient overlap were captured. The homography between the images was calculated using the standard DLT algorithm shown above.

```
im1_color = imread('../images/mosaic_im1.jpg');
im2_color = imread('../images/mosaic_im2.jpg');
im1_color = imresize(im1_color, 0.25);
im2_color = imresize(im2_color, 0.25);
im1 = rgb2gray(im1_color);
im2 = rgb2gray(im2_color);

H = findHomographyDLT(im1, im2);

% Transform the image according to the obtained H matrix
T = maketform('projective', H);

% Add RGB channels to the gray image
```

```
im2(:,:,1) = im2_color(:,:,1);
im2(:,:,2) = im2_color(:,:,2);
im2(:,:,3) = im2_color(:,:,3);

im1(:,:,1) = im1_color(:,:,1);
im1(:,:,2) = im1_color(:,:,2);
im1(:,:,3) = im1_color(:,:,3);

[im2t,xdataim2t,ydataim2t]=imtransform(im2,T);

% xdataim2t and ydataim2t store the bounds of the transformed im2
xdataout=[min(1,xdataim2t(1)) max(size(im1,2),xdataim2t(2))];
ydataout=[min(1,ydataim2t(1)) max(size(im1,1),ydataim2t(2))];
im2t=imtransform(im2,T,'XData',xdataout,'YData',ydataout);
im1t=imtransform(im1,maketform('affine',eye(3)),'XData',xdataout,'YData',
    ydataout);

% Averaging the pixel values at the common points
ims=im1t/2+im2t/2;
figure, imshow(ims);
```

Figure 6: Captured images

Using planar scene

Fixing perspective distortion

The user is asked to select the four corners of the rectangle which are used as the corresponding points for the rectangle corners in the fronto-parallel view.

```
n = 4;
x = dlmread('orig.txt');
im = imread('../images/planer_im1.jpg');
im = imresize(im, 0.25);

figure; imshow(im);
```

Figure 7: Result

```
x_i = ginput(n);
x_i = [x_i ones(n, 1)];

A = zeros(2 * n, 9);
for i = 1:n
    A(2 * i - 1, :) = [zeros(1,3) -x_i(i, :) x(i,2) * x_i(i, :)];
    A(2 * i, :) = [x_i(i, :) zeros(1, 3) -x(i,1) * x_i(i, :)];
end
[U, D, V] = svd(A);
H = V(:, 9);
H = reshape(H, [3, 3]);

% Transform the image according to the obtained H matrix
T = maketform('projective', H);
t_img = imtransform(im, T, 'Size', size(im));
imshow(t_img);
```

Digital product placement in sports

```
im1_color = imread('../images/sports1.jpg');
im2_color = imread('../images/banner.jpg');
im1 = rgb2gray(im1_color);
im2 = rgb2gray(im2_color);
imshow(im1_color);
x = ginput(4);
x = [x ones(4, 1)];
x_i = dlmread('banner.txt');
n = 4;
A = zeros(2 * n, 9);
```

Figure 8: Captured image

Figure 9: Result

```
for i = 1:n
    A(2 * i - 1, :) = [zeros(1,3) -x_i(i, :) x(i,2) * x_i(i, :)];
    A(2 * i, :) = [x_i(i, :) zeros(1, 3) -x(i,1) * x_i(i, :)];
end
[U, D, V] = svd(A);
H = V(:, 9);
H = reshape(H, [3, 3]);
T = maketform('projective', H);

im2(:,:,1) = im2_color(:,:,1);
im2(:,:,2) = im2_color(:,:,2);
im2(:,:,3) = im2_color(:,:,3);

im1(:,:,1) = im1_color(:,:,1);
im1(:,:,2) = im1_color(:,:,2);
im1(:,:,3) = im1_color(:,:,3);
[im2t , xdataim2t , ydataim2t] = imtransform(im2 ,T);

% xdataim2t and ydataim2t store the bounds of the transformed im2
xdataout=[min(1,xdataim2t(1)) max(size(im1,2),xdataim2t(2))];
```

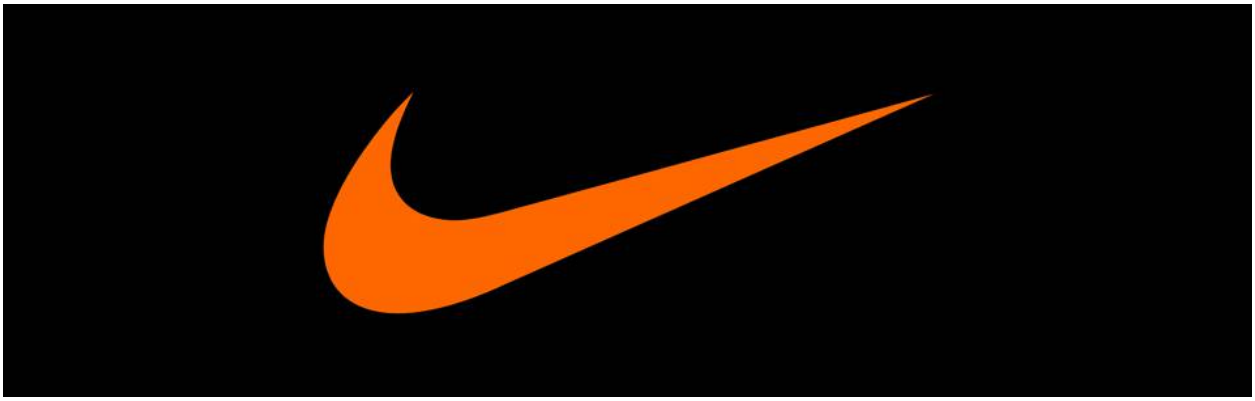

Figure 10: Banner

```
ydataout=[min(1,ydataim2t(1)) max(size(im1,1),ydataim2t(2))];
im2t=imtransform(im2,T,'XData',xdataout,'YData',ydataout);
im1t=imtransform(im1,maketform('affine',eye(3)),'XData',xdataout,'YData',
    ydataout);

% Create a mask of the polygon selected to be replaced
BW = imcomplement(poly2mask(x(:, 1)', x(:, 2)', size(im2t, 1), size(im2t, 2)))
    ;
mask = cat(3, BW, BW, BW);
ims = uint8(double(im1t) .* mask + double(im2t));
imshow(ims) ;
```

Figure 11: Results