Image Warping

Mikkel B. Stegmann[*]

Informatics and Mathematical Modelling, Technical University of Denmark
Richard Petersens Plads, Building 321, DK-2800 Kgs. Lyngby, Denmark

29th October 2001

Abstract

This note introduces the concept of image warping and treats the special case of Euclidean warping along with a discussion of a Matlab implementation. Secondly an application of image warping is given; namely *image mosaicing* where images are stitched together – e.g. to form a panoramic view.

Effort has been put into making the text as self-containing as possible. However, basic knowledge of linear algebra is a requirement. The note is intended for an introductory image analysis course.

Keywords: image warping, Euclidean warping, similarity transforms, image mosaicing.

1 Introduction

Image warping is in essence a transformation that changes the spatial configuration of an image. Using this definition a simple displacement of an image by – say – five pixels in the x-direction would be considered a warp. Formalized, this can be written as the vector function:

$$\mathbf{I}' = \mathbf{f}(\mathbf{I}) \quad , \quad \mathbf{f} : \mathbb{R}^2 \mapsto \mathbb{R}^2 \tag{1}$$

2 Euclidean Warping

In the following, we will study one particular type of warp namely; the Euclidean warp also called the Euclidean similarity transform. This type of warp involves four parameters: $\mathbf{p} = [\ s\ \alpha\ t_x\ t_y\ ]^{\mathrm{T}}$, which denotes scale, rotation and translation respectively.

Let $\mathbf{x} = [\ x\ y\ 1\ ]^{\mathrm{T}}$ denote a position in the original 2D image, $\mathbf{I}$, and let $\mathbf{x}' = [\ x'\ y'\ 1\ ]^{\mathrm{T}}$ denote the corresponding position in the warped image, $\mathbf{I}'$, (both in homogeneous coordinates). Looking at one pixel, equation (1) can now be written as a simple linear transformation, where $\mathbf{T}$ denotes the transformation matrix:

$$\mathbf{x}' = \mathbf{T}\mathbf{x} \quad \Rightarrow \tag{2}$$

[*]Contact: mbs@imm.dtu.dk, http://www.imm.dtu.dk/˜mbs/.

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} s\cos\alpha & s\sin\alpha & t_x \\ -s\sin\alpha & s\cos\alpha & t_y \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \tag{3}$$

Observe that instead of warping a single point we could warp a whole image of n points by replacing $\mathbf{x}$ with:

$$\mathbf{X} = \begin{bmatrix} x_1 & \dots & x_n \\ y_1 & \dots & y_n \\ 1 & \dots & 1 \end{bmatrix} \tag{4}$$

then equation (2) becomes:

$$\mathbf{X}' = \mathbf{TX} \tag{5}$$

The above transformation is implemented in the Matlab script `euclideanwarp` where the transformation matrix, $\mathbf{T}$, that rotates an image five degrees is written as:

```
s  = 1;
a  = 5*pi/180;
tx = 0;
ty = 0;
T  = [ s*cos(a) s*sin(a) tx ; -s*sin(a) s*cos(a) ty ; 0 0 1 ];
```

Due the discrete nature of raster images, one is in no way ensured that each input pixel exactly maps to an out pixel. Consequently warping is mostly performed backwards – i.e. from the output image to the input image. Since $\mathbf{T}$ is square and has full rank we can easily compute the inverse transformation as:

$$\mathbf{X} = \mathbf{T}^{-1}\mathbf{X}' \tag{6}$$

In Matlab this is accomplished by:

```
X = T \ [ Xp(:) Yp(:) ones(n,1) ]';
```

where `'` is the transpose operator, `Xp(:)` and `Yp(:)` are column vectors containing the x' and y' coordinates and `ones(n,1)` is a column vector of ones.

What's left now is to lookup (or interpolate) the intensity values in the incoming image, $\mathbf{I}$, at the positions in the `X` matrix. As seen below this is done for each color channel.

```
xI = reshape( X(1,:),wp,hp)';
yI = reshape( X(2,:),wp,hp)';
Ip(:,:,1) = interp2(I(:,:,1), xI, yI, '*bilinear'); % red
Ip(:,:,2) = interp2(I(:,:,2), xI, yI, '*bilinear'); % green
Ip(:,:,3) = interp2(I(:,:,3), xI, yI, '*bilinear'); % blue
```

Refer to to appendix A for a full listing of the script `euclideanwarp.m`.

3 Image Mosaicing

One application for image warping is merging of several images into a complete mosaic – e.g. to form a panoramic view. Here this is carried out by a Euclidean warp[1].

In mosaicing, the transformation between images is often not known beforehand. In this example, two images are merged and we will estimate the transformation by letting the user give points of correspondence (also called *landmarks* or *fiducial markers* etc.) in each of the images. In order to recover the transformation we rearrange the warping equation (2) so that the warping parameters is the vector **t** in:

$$\mathbf{x}' = \mathbf{Zt} \quad \Rightarrow \tag{7}$$

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} x & y & 1 & 0 & 0 \\ y & -x & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} s\cos\alpha \\ s\sin\alpha \\ t_x \\ t_y \\ 1 \end{bmatrix} \tag{8}$$

The warping parameters is now obtained by solving the linear system (7) in Matlab:

```
t  = Z \ xp;
```

From (8) we see that (at least) two points are required to determine the four warping parameters. In the Matlab script `mosaic` we prompt the user to deliver two such points on each image using the function `ginput2()`. The full code for recovering the parameters is then:

```
figure; image(I1);
[X1 Y1] = ginput2(2);
figure; image(I2);
[X2 Y2] = ginput2(2);
Z  = [ X2'  Y2' ; Y2' -X2' ; 1 1 0 0  ; 0 0 1 1 ]';
xp = [ X1' Y1' ]';
t  = Z \ xp;
a  = t(1); % = s cos(alpha)
b  = t(2); % = s sin(alpha)
tx = t(3);
ty = t(4);
```

A demonstration of mosaicing using two input images is given in figure 1 and 2.

[1]Actually, a Euclidean warp is rather limited w.r.t. the types of camera transformations it can handle. In a real image mosaicing application a projective transformation (a *homography*) should be applied.

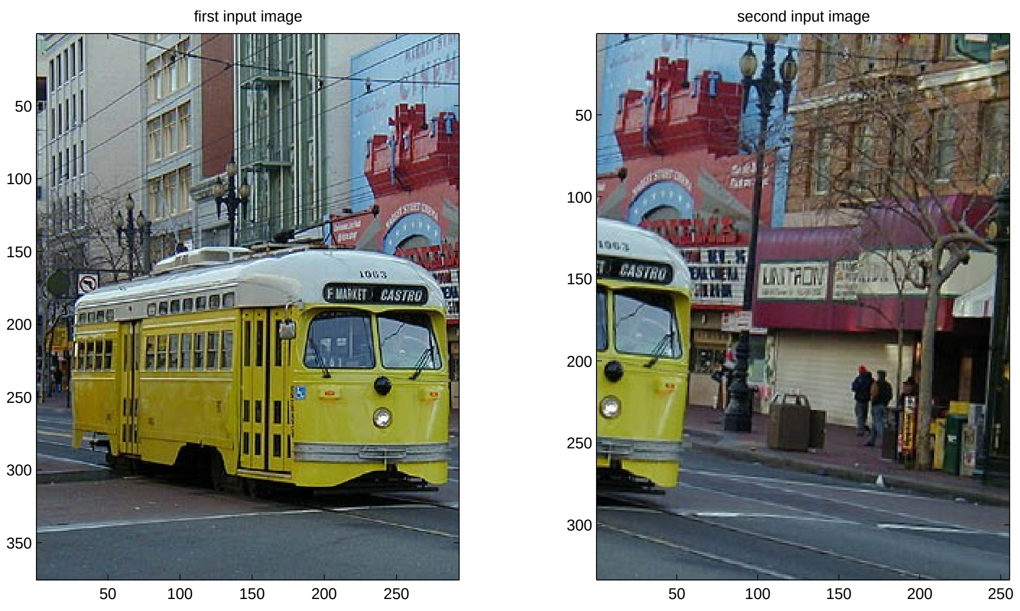

Figure 1: Left: First input image for `mosaic`. Right: Second input image for `mosaic`.

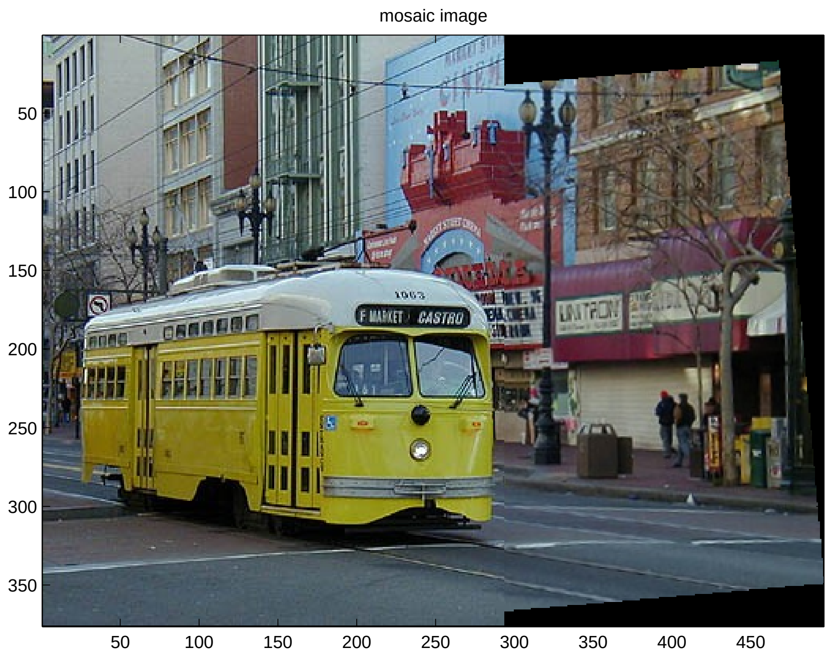

Figure 2: Image mosaic produced by `mosaic`.

4 Other Warping Functions

Below is listed some of the more commonly used warping functions ordered ascending by their degrees of freedom – i.e. the number of warping parameters. For simplicity, these are not written on matrix form. Refer to [1] for a review of additional warping functions.

Euclidean / Procrustes	$\begin{aligned} x' &= ax + by + t_x \quad , \quad a = s\cos\alpha \\ y' &= -bx + ay + t_y \quad , \quad b = s\sin\alpha \end{aligned}$
Affine / 1st order polynomial	$\begin{aligned} x' &= a_0 + a_1x + a_2y \\ y' &= b_0 + b_1y + b_2y \end{aligned}$
Bilinear	$\begin{aligned} x' &= a_0 + a_1xy + a_2x + a_3y \\ y' &= b_0 + b_1xy + b_2x + b_3y \end{aligned}$
Perspective	$\begin{aligned} x' &= (a_0 + a_1x + a_2y)/(c_0x + c_1y + 1) \\ y' &= (b_0 + b_1x + b_2y)/(c_0x + c_1y + 1) \end{aligned}$
2nd order polynomial / Biquadratic	$\begin{aligned} x' &= a_0 + a_1x + a_2y + a_3x^2 + a_4y^2 + a_5xy \\ y' &= b_0 + b_1x + b_2y + b_3x^2 + b_4y^2 + b_5xy \end{aligned}$
General polynomial	$\begin{aligned} x' &= \sum_i \sum_j a_{ij}x^iy^j \\ y' &= \sum_i \sum_j b_{ij}x^iy^j \end{aligned}$

Problems

This section presents a set of practical and theoretical problems. The former requires that you have a Matlab installation[2] and the files: `euclideanwarp.m, mosaic.m, ginput2.m, left.jpg, right.jpg, checkers.jpg`[3]. Further the current Matlab dir should be where the above files reside (use Matlab commands `pwd` and `cd`).

P0: Run the script `euclideanwarp`. Try changing the values for `s, a, tx, ty`. In which order are the three Euclidean operations applied?

P1: Run the script `mosaic`. Try doing a mosaic with two nearby points and two points far apart. Which one do you prefer? Why?

P2: Show that $\mathbf{Tx} = \mathbf{Zt}$ (you must not use the equality $\mathbf{x}' = \mathbf{x}'$).

P3: Modify `euclideanwarp` to load the image `checkers.jpg` and experiment with the interpolation schemes supported by the Matlab function `interp2()`, which are `'*nearest'`, `'*bilinear'`, `'*cubic'`, `'*spline'`. What is the observed difference?

P4: Why is homogeneous points used in section 2 as opposed to normal two dimensional points?

P5: Construct expressions on the form $\mathbf{x}' = \mathbf{Tx}$ and $\mathbf{x}' = \mathbf{Zt}$ for the bilinear warp.

P6: What is the minimum number of points you would need to determine the parameters for a bilinear warp – e.g. for an image mosaicing application? How many would you prefer? Why?

[2]Version 5.3 or greater.

[3]Placed at `http://www.imm.dtu.dk/~mbs/downloads/warping.zip` (use `unzip` to unpack).

P7: Copy the file `mosiac.m` to `mosiac2.m` and modify it to perform bilinear mosaicing. Remember that the parameters for backwards transformation can be calculated by interchanging (x, y) and (x', y') in equation (8).

P8: Modify `mosiac2.m` to take twice as many points as you answered in problem 6. How does this affect the warp?

References

[1] C. A. Glasbey and K. V. Mardia. A review of image-warping methods. *Journal of Applied Statistics*, 25(2):155–172, 1998.

A euclideanwarp.m

```
% load input image
I = double(imread('right.jpg'));
[h w d] = size(I);

% show input image
figure; image(I/255); axis image;
title('input image');

% make transformation matrix (T)
s  = 1;
a  = 5*pi/180;
tx = 0;
ty = 0;
T  = [ s*cos(a) s*sin(a) tx ; -s*sin(a) s*cos(a) ty ; 0 0 1 ];

% warp incoming corners to determine the size of the output image
cp = T*[ 1 1 w w ; 1 h 1 h ; 1 1 1 1 ];
Xpr = min( cp(1,:) ) : max( cp(1,:) ); % min x : max x
Ypr = min( cp(2,:) ) : max( cp(2,:) ); % min y : max y
[Xp,Yp] = ndgrid(Xpr,Ypr);
[wp hp] = size(Xp); % = size(Yp)

% do backwards transform (from out to in)
n = wp*hp;
X = T \ [ Xp(:) Yp(:) ones(n,1) ]';  % warp

% re-sample pixel values with bilinear interpolation
clear Ip;
xI = reshape( X(1,:),wp,hp)';
yI = reshape( X(2,:),wp,hp)';
Ip(:,:,1) = interp2(I(:,:,1), xI, yI, '*bilinear'); % red
Ip(:,:,2) = interp2(I(:,:,2), xI, yI, '*bilinear'); % green
Ip(:,:,3) = interp2(I(:,:,3), xI, yI, '*bilinear'); % blue

% show the warping result
figure; image(Ip/255); axis image;
title('warped image');
```

B mosaic.m

```
% load input images
I1 = double(imread('left.jpg'));
[h1 w1 d1] = size(I1);
I2 = double(imread('right.jpg'));
[h2 w2 d2] = size(I2);

% show input images and prompt for correspondences
figure; subplot(1,2,1); image(I1/255); axis image; hold on;
title('first input image');
[X1 Y1] = ginput2(2); % get two points from the user
subplot(1,2,2); image(I2/255); axis image; hold on;
title('second input image');
[X2 Y2] = ginput2(2); % get two points from the user

% estimate parameter vector (t)
Z  = [ X2'  Y2' ; Y2' -X2' ; 1 1 0 0  ; 0 0 1 1 ]';
xp = [ X1 ; Y1 ];
t  = Z \ xp; % solve the linear system
a  = t(1); % = s cos(alpha)
b  = t(2); % = s sin(alpha)
tx = t(3);
ty = t(4);

% construct transformation matrix (T)
T = [a b tx ; -b a ty ; 0 0 1];

% warp incoming corners to determine the size of the output image (in to out)
cp = T*[ 1 1 w2 w2 ; 1 h2 1 h2 ; 1 1 1 1 ];
Xpr = min( [ cp(1,:) 0 ] ) : max( [cp(1,:) w1] ); % min x : max x
Ypr = min( [ cp(2,:) 0 ] ) : max( [cp(2,:) h1] ); % min y : max y
[Xp,Yp] = ndgrid(Xpr,Ypr);
[wp hp] = size(Xp); % = size(Yp)

% do backwards transform (from out to in)
X = T \ [ Xp(:) Yp(:) ones(wp*hp,1) ]';  % warp

% re-sample pixel values with bilinear interpolation
clear Ip;
xI = reshape( X(1,:),wp,hp)';
yI = reshape( X(2,:),wp,hp)';
Ip(:,:,1) = interp2(I2(:,:,1), xI, yI, '*bilinear'); % red
Ip(:,:,2) = interp2(I2(:,:,2), xI, yI, '*bilinear'); % green
Ip(:,:,3) = interp2(I2(:,:,3), xI, yI, '*bilinear'); % blue

% offset and copy original image into the warped image
offset =  -round( [ min( [ cp(1,:) 0 ] ) min( [ cp(2,:) 0 ] ) ] );
Ip(1+offset(2):h1+offset(2),1+offset(1):w1+offset(1),:) = double(I1(1:h1,1:w1,:));

% show the result
figure; image(Ip/255); axis image;
title('mosaic image');
```