Image Registration

CS4243 Computer Vision and Pattern Recognition

Leow Wee Kheng

Department of Computer Science
School of Computing
National University of Singapore

Outline

1. Image Registration
2. 2D Linear Transformation
3. Registration Methods
4. Bilinear Interpolation
5. Image Mosaicking
6. Alpha Blending
7. Summary
8. Further Reading
9. Reference

Image Registration

Transform an image to align its pixels with those in another image.

- Map the coordinate (x, y) of an image to a new coordinate (x', y').
- Transformation can be linear or nonlinear.

Example: Align two images and combine them to produce a larger one.

2D Similarity Transformation

Scaling changes the point $\mathbf{p} = (x, y)$ by a constant factor s:

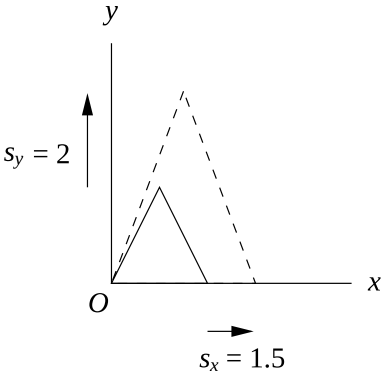
$$\begin{aligned} x' &= s\,x \\ y' &= s\,y \end{aligned} \tag{1}$$

In matrix form,

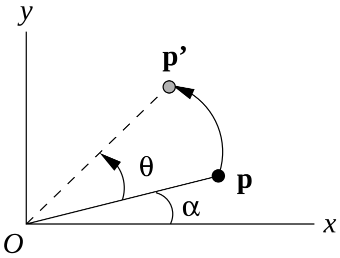
$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} s & 0 \\ 0 & s \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} \tag{2}$$

In general, the scaling factors for x and y can be different:

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} s_x & 0 \\ 0 & s_y \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} \tag{3}$$



Rotation is normally performed about the origin.



Let ρ denote the magnitude of the vector $\mathbf{p} = [\,x \quad y\,]^\top$. Then,

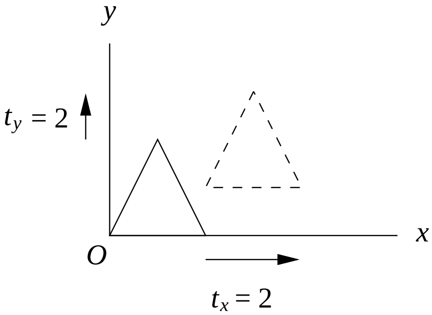
$$\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} \rho\cos\alpha \\ \rho\sin\alpha \end{bmatrix} \tag{4}$$

After rotating about the origin by an angle θ, point $\mathbf{p}$ becomes $\mathbf{p}' = [\, x' \quad y' \,]^\top$:

$$\begin{aligned}
\begin{bmatrix} x' \\ y' \end{bmatrix} &= \begin{bmatrix} \rho\cos(\alpha+\theta) \\ \rho\sin(\alpha+\theta) \end{bmatrix} = \begin{bmatrix} \rho\,(\cos\alpha\cos\theta - \sin\alpha\sin\theta) \\ \rho\,(\sin\alpha\cos\theta + \cos\alpha\sin\theta) \end{bmatrix} \\
&= \begin{bmatrix} x\cos\theta - y\sin\theta \\ x\sin\theta + y\cos\theta \end{bmatrix} \\
&= \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}
\end{aligned} \tag{5}$$

Translation of point $\mathbf{p} = [\,x \quad y\,]^\top$ by the vector $\mathbf{T} = [\,t_x \quad t_y\,]^\top$ is given by

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} t_x \\ t_y \end{bmatrix} = \begin{bmatrix} x + t_x \\ y + t_y \end{bmatrix} \qquad (6)$$



Homogeneous coordinates of the 2D point

$$\mathbf{p} = \begin{bmatrix} x \\ y \end{bmatrix}$$

are

$$\begin{bmatrix} cx \\ cy \\ c \end{bmatrix}$$

for any non-zero c.

The 2D vector $\mathbf{p}$ becomes a 3D vector.

Given a point $[\,x \quad y \quad z\,]^\top$ in homogeneous coords,
its 2D Cartesian coords are $[\,x/z \quad y/z\,]^\top$, provided $z \neq 0$.
If $z = 0$, then this is a point at infinity.

Homogeneous coordinates apply to 3D points as well, by adding a 4th component.

Can combine rotation, scaling, and translation into a single matrix using homogeneous coordinates:

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

$$= \begin{bmatrix} s\cos\theta & -s\sin\theta & t_x \\ s\sin\theta & s\cos\theta & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \tag{7}$$

2D Affine Transformation

Affine transform is a generalization of linear transformation:

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \tag{8}$$

for some parameters a_{ij}.

In short-hand notation:

$$\mathbf{p}' = \mathbf{A}\,\mathbf{p} \tag{9}$$

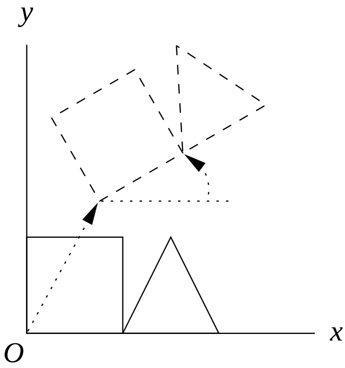
$\mathbf{A}$ is the affine transformation matrix.

Registration Methods

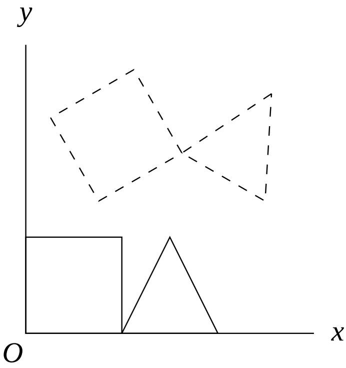
Given two images, how to register one with the other?

Basic idea:

1. Determine the corresponding points between the images.
   - Manually mark corresponding points, or
   - Detect and match features between views (see lecture on feature detection and matching).
2. Determine the transformation between corresponding points.
   - Assume that all pairs of corresponding points are related by the same transformation.
   - Compute parameters of transformation given corresponding points.



(a) same rotation



(b) different rotation

- In general, need to apply non-linear method.

Let's try affine transformation which is simpler to work with.

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Affine transformation (Eq. 8) has 6 parameters.

- Need 3 pairs of corresponding points.
- Usually use more than 3 pairs to obtain best fitting affine parameters.

Method 1

Suppose we have n pairs of corresponding points $\mathbf{p}_i$ and $\mathbf{p}'_i$.

From Eq. 8,

$$\begin{aligned} x'_i &= a_{11}\,x_i + a_{12}\,y_i + a_{13} \\ y'_i &= a_{21}\,x_i + a_{22}\,y_i + a_{23} \end{aligned} \tag{10}$$

for $i = 1, \ldots, n$.

Now, we have two sets of linear equations of the form

$$\mathbf{M}\,\mathbf{a} = \mathbf{b} \tag{11}$$

First set:

$$\begin{bmatrix} x_1 & y_1 & 1 \\ \vdots & \vdots & \vdots \\ x_n & y_n & 1 \end{bmatrix} \begin{bmatrix} a_{11} \\ a_{12} \\ a_{13} \end{bmatrix} = \begin{bmatrix} x'_1 \\ \vdots \\ x'_n \end{bmatrix} \tag{12}$$

Second set:

$$\begin{bmatrix} x_1 & y_1 & 1 \\ \vdots & \vdots & \vdots \\ x_n & y_n & 1 \end{bmatrix} \begin{bmatrix} a_{21} \\ a_{22} \\ a_{23} \end{bmatrix} = \begin{bmatrix} y'_1 \\ \vdots \\ y'_n \end{bmatrix} \tag{13}$$

- Number of equations > number of unknowns. No exact solution.
- Can compute best fitting a_{ij} for each set independently.
- Use linear least square fit to compute.
- There's a variation of this method (Lab 2).

In

$$\mathbf{M}\,\mathbf{a} = \mathbf{b}, \tag{14}$$

$\mathbf{M}$ is not square and so has no inverse.

But, $\mathbf{M}^\top\mathbf{M}$ is square and has inverse (typically). So,

$$\begin{aligned} \mathbf{M}^\top\mathbf{M}\,\mathbf{a} &= \mathbf{M}^\top\mathbf{b} \\ \mathbf{a} &= (\mathbf{M}^\top\mathbf{M})^{-1}\mathbf{M}^\top\mathbf{b} \end{aligned} \tag{15}$$

- $(\mathbf{M}^\top\mathbf{M})^{-1}\mathbf{M}^\top$ is the pseudo-inverse of $\mathbf{M}$.
- Pseudo-inverse gives the least squared error solution.
- In practice, pseudo-inverse can be very large matrix.
  So, don't use it directly.
- Numerical software such as NumPy, Matlab, Numerical Recipes provide functions for computing the linear least square solution (Lab 2).

Method 2

Put the x' and y' parts in the same matrix equation:

$$\begin{bmatrix} x_1 & y_1 & 1 & 0 & 0 & 0 \\ & & \vdots & & & \\ x_n & y_n & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & x_1 & y_1 & 1 \\ & & \vdots & & & \\ 0 & 0 & 0 & x_n & y_n & 1 \end{bmatrix} \begin{bmatrix} a_{21} \\ a_{22} \\ a_{23} \\ a_{21} \\ a_{22} \\ a_{23} \end{bmatrix} = \begin{bmatrix} x'_1 \\ \vdots \\ x'_n \\ y'_1 \\ \vdots \\ y'_n \end{bmatrix} \tag{16}$$

- This system of linear equations can be easily solved in NumPy.
- Actually, the x' and y' parts are still independent of each other.

Beware!

Suppose you sum the x' and y' parts, you will get

$$x'_i + y'_i = a_{11}\, x_i + a_{12}\, y_i + a_{13} + a_{21}\, x_i + a_{22}\, y_i + a_{23}. \tag{17}$$

That is correct. But, if you form the matrix equation like this

$$\begin{bmatrix} x_1 & y_1 & 1 & x_1 & y_1 & 1 \\ & & \vdots & & & \\ x_n & y_n & 1 & x_n & y_n & 1 \end{bmatrix} \begin{bmatrix} a_{11} \\ a_{12} \\ a_{13} \\ a_{21} \\ a_{22} \\ a_{23} \end{bmatrix} = \begin{bmatrix} x'_1 + y'_1 \\ x'_2 + y'_2 \\ \vdots \\ x'_n + y'_n \end{bmatrix} \tag{18}$$

you can't get the correct results. Reasons:

- There are only 3 independent columns in the matrix!
- The matrix has a rank of 3, instead of the required 6.

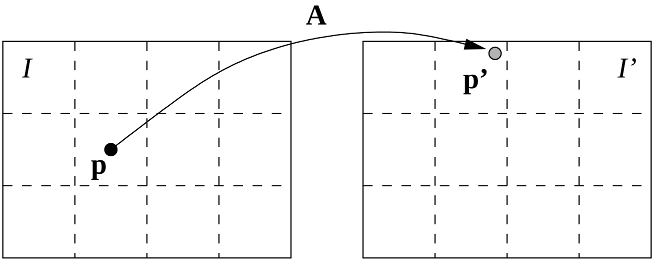
Bilinear interpolation

Suppose the matrix $\mathbf{A}$ maps $\mathbf{p}$ in image I to $\mathbf{p}'$ in image I'. Then,

$$\mathbf{p}' = \mathbf{A}\,\mathbf{p} \tag{19}$$

and

$$I'(\mathbf{p}') = I(\mathbf{p}) \tag{20}$$



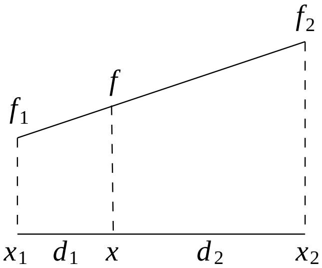
- dashed boxes: pixels
- black dot: center of pixel, integer-valued coordinates
- gray dot: off-centered, real-valued coordinates

Note:

- Cannot use $I(\mathbf{p})$ for $I'(\mathbf{p}')$:
  - In general, $\mathbf{p}'$ has real-valued coordinates even when $\mathbf{p}$ has integer-valued coordinates.
  - But, image pixel locations are integer-valued.
  - Rounding $\mathbf{p}'$ to integer causes error in $I'(\mathbf{p}')$.
- However, can use $I'(\mathbf{p}')$ for $I(\mathbf{p})$:
  - Can estimate $I'(\mathbf{p}')$ from neighboring pixel values using bilinear interpolation.

Linear Interpolation

First, consider the 1D case: linear interpolation.



$$\frac{f - f_1}{x - x_1} = \frac{f_2 - f}{x_2 - x} \tag{21}$$

i.e.,

$$\frac{f - f_1}{d_1} = \frac{f_2 - f}{d_2} \tag{22}$$

Rearranging terms yields

$$f = \frac{d_1 f_2 + d_2 f_1}{d_1 + d_2} \tag{23}$$

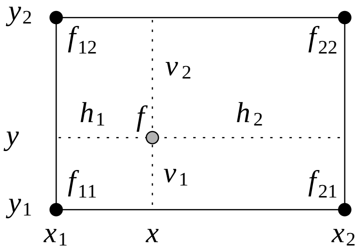
If $[x_1, x_2]$ is a unit interval, then

$$f = \alpha f_2 + (1 - \alpha) f_1 \tag{24}$$

where $\alpha = d_1$.

Bilinear Interpolation

Now, consider the 2D case: bilinear interpolation.



First, apply linear interpolation to obtain $f(x_1, y)$ and $f(x_2, y)$.

$$\begin{aligned} f(x_1, y) &= \frac{v_1 f(x_1, y_2) + v_2 f(x_1, y_1)}{v_1 + v_2} \\ f(x_2, y) &= \frac{v_1 f(x_2, y_2) + v_2 f(x_2, y_1)}{v_1 + v_2} \end{aligned} \tag{25}$$

Then, apply linear interpolation between $f(x1, y)$ and $f(x2, y)$.

$$\begin{aligned} f(x, y) &= \frac{h_1 f(x_2, y) + h_2 f(x_1, y)}{h_1 + h_2} \\ &= \frac{h_1 v_1 f_{22} + h_1 v_2 f_{21} + h_2 v_1 f_{12} + h_2 v_2 f_{11}}{(h_1 + h_2)(v_1 + v_2)} \end{aligned} \tag{26}$$

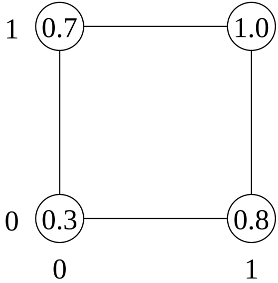
where $f_{ij} = f(x_i, y_j)$.

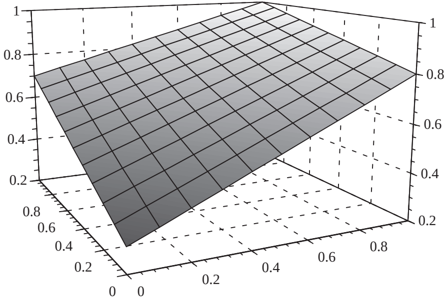
For a unit square, with $\alpha = h_1, \beta = v_1$,

$$f(x, y) = \alpha\beta f_{22} + \alpha(1 - \beta) f_{21} + (1 - \alpha)\beta f_{12} + (1 - \alpha)(1 - \beta) f_{11} \tag{27}$$

 Bilinear Interpolation

Example

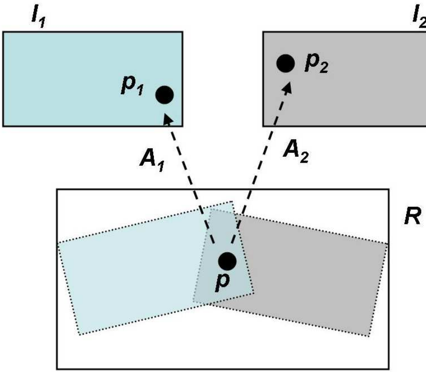




Note:
In general, can have trilinear interpolation in 3D, multilinear interpolation in multi-D.

Image Mosaicking

Combine small overlapping images into single large image.



Method

Suppose that $\mathbf{A}_1$ and $\mathbf{A}_2$ are known.
They specify the transformation between the output image R and the input images I_1 and I_2, respectively.

For each pixel $\mathbf{p}$ in R, do:

- Compute: $\mathbf{p}_1 = \mathbf{A}_1\mathbf{p}$ and $\mathbf{p}_2 = \mathbf{A}_2\mathbf{p}$.
- If both $\mathbf{p}_1$ and $\mathbf{p}_2$ fall outside of I_1 and I_2, respectively, then $R(\mathbf{p})$ = default color, e.g., black.
- If both $\mathbf{p}_1$ and $\mathbf{p}_2$ fall inside of I_1 and I_2, respectively, then $R(\mathbf{p})$ = blending of $I_1(\mathbf{p}_1)$ and $I_2(\mathbf{p}_2)$.
- Otherwise, only one of $\mathbf{p}_1$ or $\mathbf{p}_2$ falls inside I_1 or I_2. So, $R(\mathbf{p}) = I_1(\mathbf{p}_1)$ or $I_2(\mathbf{p}_2)$, as appropriate.

Notes:

- $\mathbf{A}_1$ and $\mathbf{A}_2$ are solved using the methods introduced earlier.
- Usually, R is chosen to have the same viewpoint as one of the input images, e.g., that of I_1. Then $\mathbf{A}_1$ is the identity matrix $\mathbf{I}$.
- Usually $\mathbf{p}_1$ and $\mathbf{p}_2$ do not have integer coordinates. So, use bilinear interpolation to determine its color.
- Alpha blending is usually used to blend colors coming from different input images.

Example: input images

Example: mosaicked image

Alpha Blending

Usually, the images to be mosaicked together have different overall intensity and contrast.

The mosaicked image has an apparent seam.

To remove the seam, apply **alpha blending**.

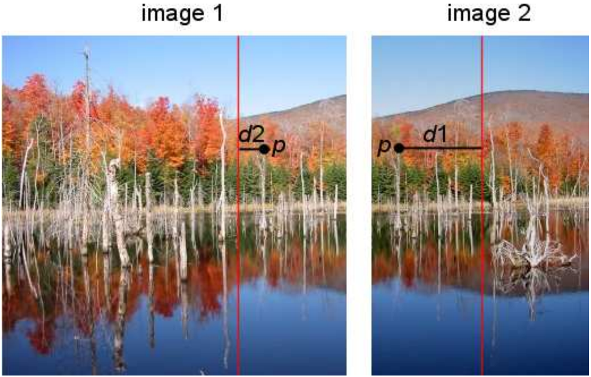
Basic idea

- Let the color in the overlapping regions change smoothly from the color in one image to the color in the other image.
- Let $C_1(p)$ denote color of pixel p in image 1.
- Let $C_2(p)$ denote color of pixel p in image 2.
- Then, color $C(p)$ of blended image is given by

$$C(p) = \alpha C_1(p) + (1 - \alpha) C_2(p) \tag{28}$$

where α is related to the distances to the overlapping boundaries, e.g.,

$$\alpha = \frac{d_1}{d_1 + d_2} \tag{29}$$



- When $d_1 = 0$, pixel is not in image 1. $C(p) = C_2(p)$.
- When $d_2 = 0$, pixel is not in image 2. $C(p) = C_1(p)$.
- Otherwise, $C(p)$ is a blend of $C_1(p)$ and $C_2(p)$.

Example

without blending

with blending

Summary

- Affine transformation is a simple linear transformation.
- Affine transformation can change shape:
  it includes scaling, rotation, translation, and shearing.
- Image mosaicking transforms images into the same coordinate frame and blend them together.
- Bilinear interpolation estimates colours at real-number coordinates.
- Alpha blending blends images seamlessly.
- Beside affine transformation, can also use homography (see lecture on multiple view methods).

Further Reading

- Affine mapping: [SS01] Section 11.3, 11.4
- Examples of image mosaicking: CS4243 website: project showcase
- Image stitching (mosaicking): [Sze10] Chapter 9.

Reference I

L. Shapiro and Stockman.
Computer Vision.
Prentice-Hall, 2001.

R. Szeliski.
Computer Vision: Algorithms and Applications.
Springer, 2010.