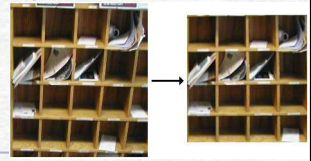
Geometric

- Geometric transformations
- Pixel co-ordinate transformationns
- Brightness interpolation



1

Geometric transformations

- Computer graphics
  - Introduce distortion

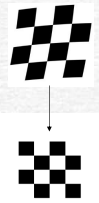
Examples

- Image processing
  - Image mosaicing
  - Matching/Registering image
  - Eliminating distortion
  - Simplifying further processing
    - e.g. OCR

2

Geometric transformations - Specification

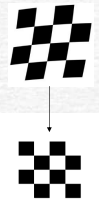
- Problem:
  - Distorted image $f(i,j)$
  - Corrected image $f'(i',j')$
- Mapping: $i = T_i(i',j')$ $j = T_j(i',j')$



3

Geometric transformations - Specification

- Application steps:
  - Define the transformation
    - Known in advance
    - Determine through correspondences
      - Image to known
      - Image to image
  - Apply the transformation
    - For every point in the output image
      - Determine where it came from using T
      - Interpolate a value for the output point



4

Output ➔ Input

- A backwards transformation?
  - Why aren't we mapping from the input image to the output?
    - For each point maintain the new continuous coordinates
  - For each output point
    - Find the nearest transformed point(s)
    - Interpolate a value from this point(s)
- We would have to
  - Store all real coordinates
  - Search for nearest values for each output pixel
  - Interpolate in corrected image space

5

Affine transformations

- Definition

$$\begin{bmatrix} i \\ j \end{bmatrix} = \begin{bmatrix} a_{00} & a_{01} & a_{02} \\ a_{10} & a_{11} & a_{12} \end{bmatrix} \begin{bmatrix} i' \\ j' \\ 1 \end{bmatrix}$$

- Known transformations...
  - E.g. Translation

$$\begin{bmatrix} i \\ j \end{bmatrix} = \begin{bmatrix} 1 & 0 & m \\ 0 & 1 & n \end{bmatrix} \begin{bmatrix} i' \\ j' \\ 1 \end{bmatrix}$$

- Unknown transformations...
  - Requires at least 3 observations

```
Mat affine_matrix( 2, 3, CV_32FC1 );
…
warpAffine( image, result, affine_matrix, image.size() );
```


6

Simple affine transformations

- Rotation

$$\begin{bmatrix} i \\ j \end{bmatrix} = \begin{bmatrix} \cos\phi & \sin\phi & 0 \\ -\sin\phi & \cos\phi & 0 \end{bmatrix} \begin{bmatrix} i' \\ j' \\ 1 \end{bmatrix}$$

- Change of scale

$$\begin{bmatrix} i \\ j \end{bmatrix} = \begin{bmatrix} a & 0 & 0 \\ 0 & b & 0 \end{bmatrix} \begin{bmatrix} i' \\ j' \\ 1 \end{bmatrix}$$

- Skewing

$$\begin{bmatrix} i \\ j \end{bmatrix} = \begin{bmatrix} 1 & \tan\phi & 0 \\ 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} i' \\ j' \\ 1 \end{bmatrix}$$

7

More affine transformations

- Panoramic distortion

$$\begin{bmatrix} i \\ j \end{bmatrix} = \begin{bmatrix} a & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} i' \\ j' \\ 1 \end{bmatrix}$$

8

Unknown affine transformations

- Given…

$$\begin{bmatrix} i \\ j \end{bmatrix} = \begin{bmatrix} a_{00} & a_{01} & a_{02} \\ a_{10} & a_{11} & a_{12} \end{bmatrix} \begin{bmatrix} i' \\ j' \\ 1 \end{bmatrix}$$

- If we have 3 observations…

$$(i_1, j_1) \leftrightarrow (i'_1, j'_1)$$
$$(i_2, j_2) \leftrightarrow (i'_2, j'_2)$$
$$(i_3, j_3) \leftrightarrow (i'_3, j'_3)$$

- We can reorganise..

$$\begin{bmatrix} i_1 \\ j_1 \\ i_2 \\ j_2 \\ i_3 \\ j_3 \end{bmatrix} = \begin{bmatrix} i'_1 & j'_1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & i'_1 & j'_1 & 1 \\ i'_2 & j'_2 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & i'_2 & j'_2 & 1 \\ i'_3 & j'_3 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & i'_3 & j'_3 & 1 \end{bmatrix} \begin{bmatrix} a_{00} \\ a_{01} \\ a_{02} \\ a_{10} \\ a_{11} \\ a_{12} \end{bmatrix}$$

- And take the inverse to compute the a coefficients

9

Unknown affine transformations


```
Point2f source [3], destination [3];
…
affine_matrix = getAffineTransform(source,destination);
```

- More observations
  - Better estimate of the a coefficients
  - Must use the psuedo inverse

10

Perspective transformations

- Perspective projection
  - Planar surface
  - Not parallel to the image plane.
    - Cannot be corrected with the affine transformation
  - Need a perspective transformation

$$\begin{bmatrix} i.w \\ j.w \\ w \end{bmatrix} = \begin{bmatrix} p_{00} & p_{01} & p_{02} \\ p_{10} & p_{11} & p_{12} \\ p_{20} & p_{21} & 1 \end{bmatrix} \begin{bmatrix} i' \\ j' \\ 1 \end{bmatrix}$$

11

Perspective transformations

- From $\begin{bmatrix} i.w \\ j.w \\ w \end{bmatrix} = \begin{bmatrix} p_{00} & p_{01} & p_{02} \\ p_{10} & p_{11} & p_{12} \\ p_{20} & p_{21} & 1 \end{bmatrix} \begin{bmatrix} i' \\ j' \\ 1 \end{bmatrix}$

- We know $i.w = p_{00}.i' + p_{01}.j' + p_{02}$

$$w = p_{20}.i' + p_{21}.j' + 1$$

- Hence $i = p_{00}.i' + p_{01}.j' + p_{02} - p_{20}.i.i' - p_{21}.i.j'$
- And similarly $j = p_{10}.i' + p_{11}.j' + p_{12} - p_{20}.j.i' - p_{21}.j.j'$
- If we observe 4 mappings…

12

Perspective transformations

- Then

$$\begin{bmatrix} i_1 \\ j_1 \\ i_2 \\ j_2 \\ i_3 \\ j_3 \\ i_4 \\ j_4 \end{bmatrix} = \begin{bmatrix} i'_1 & j'_1 & 1 & 0 & 0 & 0 & -i_1 i'_1 & -i_1 j'_1 \\ 0 & 0 & 0 & i'_1 & j'_1 & 1 & -j_1 i'_1 & -j_1 j'_1 \\ i'_2 & j'_2 & 1 & 0 & 0 & 0 & -i_2 i'_2 & -i_2 j'_2 \\ 0 & 0 & 0 & i'_2 & j'_2 & 1 & -j_2 i'_2 & -j_2 j'_2 \\ i'_3 & j'_3 & 1 & 0 & 0 & 0 & -i_3 i'_3 & -i_3 j'_3 \\ 0 & 0 & 0 & i'_3 & j'_3 & 1 & -j_3 i'_3 & -j_3 j'_3 \\ i'_4 & j'_4 & 1 & 0 & 0 & 0 & -i_4 i'_4 & -i_4 j'_4 \\ 0 & 0 & 0 & i'_4 & j'_4 & 1 & -j_4 i'_4 & -j_4 j'_4 \end{bmatrix} \begin{bmatrix} p_{00} \\ p_{01} \\ p_{02} \\ p_{10} \\ p_{11} \\ p_{12} \\ p_{20} \\ p_{21} \end{bmatrix}$$

- Multiply by the inverse of the square matirx…
- More observations gives
  - A more accurate transformation
  - Matrix becomes non-square…
  - Pseudo inverse required

13

Perspective transformations

```
Point2f source [4], destination [4];
// Assign values to source and destination points.
perspective_matrix = getPerspectiveTransform( source, destination );
warpPerspective( image, result, perspective_matrix, result.size() );
```


14

More complex transformations

- Approximation by a polynomial

$$i = T_i(i', j') = a_{00} + a_{10} i' + a_{01} j' + a_{11} i' j' + a_{02} (j')^2 + a_{20} (i')^2 + a_{12} i' (j')^2 + a_{21} (i')^2 j' + a_{22} (i')^2 (j')^2 + \ldots$$

$$j = T_j(i', j') = b_{00} + b_{10} i' + b_{01} j' + b_{11} i' j' + b_{02} (j')^2 + b_{20} (i')^2 + b_{12} i' (j')^2 + b_{21} (i')^2 j' + b_{22} (i')^2 (j')^2 + \ldots$$

  - No. of correspondences.
  - Distribution of points
  - Solve simultaneous linear equations
  - Least squared error solution

- Even more complex
  - Partition the image.

15

Brightness interpolation

- Locations are not integer coordinates
- Interpolate output pixel value from
  - The nearby pixels in the original (uncorrected) image
- Loss of accuracy
  - Complexity of interpolation scheme
- Interpolation schemes:
  - Nearest neighbour
  - Bilinear interpolation
  - Bicubic interpolation

16

Nearest neighborhood interpolation

- Simple formulation: $f'(i', j') = f(round(i), round(j))$
- Error is perceptible
  - Step edges

17

Bi-linear interpolation

$$f'^{(i',j')} = (trunc(i) + 1 - i)(trunc(j) + 1 - j)f(trunc(i), trunc(j)) + (i - trunc(i))(trunc(j) + 1 - j)f(trunc(i) + 1, trunc(j)) + (trunc(i) + 1 - i)(j - trunc(j))f(trunc(i), trunc(j) + 1) + (i - trunc(i))(j - trunc(j))f(trunc(i) + 1, trunc(j) + 1)$$

- Brightness function is bilinear
- Neighbours contribute
- Error is perceptible
  - Blurring

18

Bi-cubic interpolation

- Approximate using a bi-cubic polynomial surface
- No step-like boundary problems
- Copes with linear interpolation blurring
- Often used in raster displays…

19

Brightness interpolation

```
int interpolation_scheme = INTER_LINEAR;
// Nearest neighbour interpolation: INTER_NEAREST
// Bilinear interpolation: INTER_LINEAR
// Bicubic interpolation: INTER_CUBIC
warpAffine( image, result, affine_matrix, result_size,
                                        interpolation_scheme );
warpPerspective( image, result, perspective_matrix,
                                    result_size, interpolation_scheme );
```


20

Camera models – Distortion

- Radial distortion
  - Radially symmetric from the optical axis
  - Change in magnification
    - Barrel distortion
    - Pincushion distortion
  - Caused by the lenses

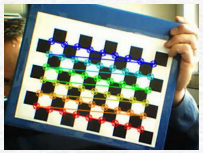
$$i' = i(1 + k_1 r^2 + k_2 r^4 + k_3 r^6)$$
$$j' = j(1 + k_1 r^2 + k_2 r^4 + k_3 r^6)$$
$$r = \sqrt{i^2 + j^2}$$

- Tangential distortion
  - Uneven magnification from one side to the other
  - Lenses not parallel to the image plane

$$i' = i + (2p_1 ij + p_2(r^2 + 2i^2))$$
$$j' = j + (2p_2 ij + p_1(r^2 + 2j^2))$$

Camera models – Removing distortion

- Calibrate using multiple images
  - Known object
  - Different positions
  - Computes camera matrix
  - & distortion parameters
- Remove distortion
  - Mainly using the distortion parameters



```
calibrateCamera( object_points, image_points,
    image_size, camera_matrix, distortion_coefficients,
    rotation_vectors, translation_vectors );
…
undistort( camera_image, corrected_image,
        camera_matrix, distortion_coefficients );
```