Image Processing

Cosimo Distante

Lecture 14: Alignment
Today: Alignment

- Homographies
- Rotational Panoramas
- RANSAC
- Global alignment
- Warping
Motivation: Recognition

Figures from David Lowe
Motivation: medical image registration
Motivation: Mosaics

• Getting the whole picture
  – Consumer camera: $50^\circ \times 35^\circ$

Slide from Brown & Lowe 2003
Motivation: Mosaics

• Getting the whole picture
  – Consumer camera: 50° x 35°
  – Human Vision: 176° x 135°

Slide from Brown & Lowe 2003
Motivation: Mosaics

• Getting the whole picture
  – Consumer camera: 50° x 35°
  – Human Vision: 176° x 135°

• Panoramic Mosaic = up to 360 x 180°

Slide from Brown & Lowe 2003
Motion models
Motion models

- What happens when we take two images with a camera and try to align them?
- translation?
- rotation?
- scale?
- affine?
- perspective?
- ... see interactive demo (VideoMosaic)
Image Warping
Image Warping

• image filtering: change range of image
  • \( g(x) = h(f(x)) \)

• image warping: change domain of image
  • \( g(x) = f(h(x)) \)
Image Warping

• image filtering: change *range* of image
  • \( g(x) = h(f(x)) \)

• image warping: change *domain* of image
  • \( g(x) = f(h(x)) \)
Parametric (global) warping

• Examples of parametric warps:

    translation
    rotation
    aspect
    affine
    perspective
    cylindrical

Szeliski
Image Warping

• Given a coordinate transform $x' = h(x)$ and a source image $f(x)$, how do we compute a transformed image $g(x') = f(h(x))$?
Forward Warping

• Send each pixel $f(x)$ to its corresponding location $x' = h(x)$ in $g(x')$
• What if pixel lands “between” two pixels?
Forward Warping

• Send each pixel $f(x)$ to its corresponding location $x' = h(x)$ in $g(x')$

• What if pixel lands “between” two pixels?
• Answer: add “contribution” to several pixels, normalize later (splatting)
Inverse Warping

- Get each pixel $g(x')$ from its corresponding location $x' = h(x)$ in $f(x)$
- What if pixel comes from “between” two pixels?
Inverse Warping

• Get each pixel $g(x')$ from its corresponding location $x' = h(x)$ in $f(x)$

• What if pixel comes from “between” two pixels?

• Answer: resample color value from interpolated (prefiltered) source image

$g(x') = \text{resample color value from interpolated (prefiltered) source image}$
Inverse warping

Get each pixel $g(x',y')$ from its corresponding location

$$(x,y) = T^{-1}(x',y')$$
in the first image.

Q: what if pixel comes from “between” two pixels?
A: Interpolate color value from neighbors
   - nearest neighbor, bilinear…

>> help interp2
Bilinear interpolation

Sampling at $f(x,y)$:

\[
  f(x, y) = (1 - a)(1 - b) \ f[i, j] \\
  + a(1 - b) \ f[i + 1, j] \\
  + ab \ f[i + 1, j + 1] \\
  + (1 - a)b \ f[i, j + 1]
\]

Slide from Alyosha Efros, CMU
Interpolation

• Possible interpolation filters:
  – nearest neighbor
  – bilinear
  – bicubic (interpolating)
  – sinc / FIR

• Needed to prevent “jaggies” and “texture crawl”
2D coordinate transformations

• translation: \( x' = x + t \) \( x = (x,y) \)
• rotation: \( x' = R x + t \)
• similarity: \( x' = s R x + t \)
• affine: \( x' = A x + t \)
• perspective: \( \underline{x'} \cong \underline{H x} \) \( \underline{x} = (x,y,1) \)
  \( \underline{x} \) is a homogeneous coordinate

• These all form a nested group (closed w/ inv.)
Example: discovering translation

Given matched points in \( \{A\} \) and \( \{B\} \), estimate the translation of the object

\[
\begin{bmatrix}
    x_i^B \\
    y_i^B
\end{bmatrix} = \begin{bmatrix}
    x_i^A \\
    y_i^A
\end{bmatrix} + \begin{bmatrix}
    t_x \\
    t_y
\end{bmatrix}
\]
Example: discovering rot/trans/scale

Given matched points in \{A\} and \{B\}, estimate the transformation matrix

\[
\begin{bmatrix}
x_i^B \\
y_i^B
\end{bmatrix} = T \begin{bmatrix}
x_i^A \\
y_i^A
\end{bmatrix} + \begin{bmatrix}
t_x \\
t_y
\end{bmatrix}
\]

\[
T = \begin{bmatrix}
a & b \\
c & d
\end{bmatrix}
\]
Parametric (global) transformations

Transformation $T$ is a coordinate-changing machine:
$p' = T(p)$

What does it mean that $T$ is global?
- $T$ is the same for any point $p$
$T$ can be described by just a few numbers (parameters)

For linear transformations, we can represent $T$ as a matrix

\[
p' = Tp = \begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} x \\ y \end{bmatrix}
\]
## Basic 2D Transformations

### Basic 2D transformations as 3x3 matrices

- **Translate**
  
  \[
  \begin{bmatrix}
  x' \\
  y' \\
  1
  \end{bmatrix} = \begin{bmatrix}
  1 & 0 & t_x \\
  0 & 1 & t_y \\
  0 & 0 & 1
  \end{bmatrix} \begin{bmatrix}
  x \\
  y \\
  1
  \end{bmatrix}
  \]

- **Scale**
  
  \[
  \begin{bmatrix}
  x' \\
  y' \\
  1
  \end{bmatrix} = \begin{bmatrix}
  s_x & 0 & 0 \\
  0 & s_y & 0 \\
  0 & 0 & 1
  \end{bmatrix} \begin{bmatrix}
  x \\
  y \\
  1
  \end{bmatrix}
  \]

- **Rotate**
  
  \[
  \begin{bmatrix}
  x' \\
  y' \\
  1
  \end{bmatrix} = \begin{bmatrix}
  \cos \Theta & -\sin \Theta & 0 \\
  \sin \Theta & \cos \Theta & 0 \\
  0 & 0 & 1
  \end{bmatrix} \begin{bmatrix}
  x \\
  y \\
  1
  \end{bmatrix}
  \]

- **Shear**
  
  \[
  \begin{bmatrix}
  x' \\
  y' \\
  1
  \end{bmatrix} = \begin{bmatrix}
  1 & s h_x & 0 \\
  s h_y & 1 & 0 \\
  0 & 0 & 1
  \end{bmatrix} \begin{bmatrix}
  x \\
  y \\
  1
  \end{bmatrix}
  \]

*Source: Alyosha Efros*
2-D Rotation

\[ x' = x \cos(\theta) - y \sin(\theta) \]
\[ y' = x \sin(\theta) + y \cos(\theta) \]
2-D Rotation

This is easy to capture in matrix form:

\[
\begin{bmatrix}
x' \\
y'
\end{bmatrix} =
\begin{bmatrix}
\cos(\theta) & -\sin(\theta) \\
\sin(\theta) & \cos(\theta)
\end{bmatrix}
\begin{bmatrix}
x \\
y
\end{bmatrix}
\]

Even though \(\sin(\theta)\) and \(\cos(\theta)\) are nonlinear functions of \(\theta\),

- \(x'\) is a linear combination of \(x\) and \(y\)
- \(y'\) is a linear combination of \(x\) and \(y\)

What is the inverse transformation?

- Rotation by \(-\theta\)
- For rotation matrices \(R^{-1} = R^T\)
Affine Transformations

Affine transformations are combinations of
  • Linear transformations, and
  • Translations

Properties of affine transformations:
  • Lines map to lines
  • Parallel lines remain parallel
  • Ratios are preserved
  • Closed under composition

\[
\begin{bmatrix}
x' \\
y'
\end{bmatrix} = \begin{bmatrix} a & b & c \\ d & e & f \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}
\]

or

\[
\begin{bmatrix}
x' \\
y' \\
1
\end{bmatrix} = \begin{bmatrix} a & b & c \\ d & e & f \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}
\]
Projective Transformations

Properties of projective transformations:
- Lines map to lines
- Parallel lines do not necessarily remain parallel
- Ratios are not preserved
- Closed under composition
- Models change of basis
- Projective matrix is defined up to a scale (8 DOF)

\[
\begin{bmatrix}
  x' \\
  y' \\
  w'
\end{bmatrix} =
\begin{bmatrix}
  a & b & c \\
  d & e & f \\
  g & h & i
\end{bmatrix}
\begin{bmatrix}
  x \\
  y \\
  w
\end{bmatrix}
\]

Projective transformations:
- Affine transformations, and
- Projective warps
Fitting and Alignment

Fitting:
Find the parameters of a model that best fit the data.

Alignment:
Find the parameters of the transformation that best aligns matched points.
Fitting and Alignment

• Challenges
  – Design a suitable *goodness of fit* measure
    • Similarity should reflect application goals
    • Encode robustness to outliers and noise

  – Design an *optimization* method
    • Avoid local optima
    • Find best parameters quickly

  – Typically want to solve for a global transformation that accounts for *the most* true correspondences
    • Noise (typically 1-3 pixels)
    • Outliers (often 50%)
    • Many-to-one matches or multiple objects
Fitting and Alignment: Methods

• Global optimization / search for parameters
  – Least squares fit
  – Robust least squares
  – Iterative closest point (ICP)

• Hypothesize and test
  – RANSAC
Fitting and Alignment: Methods

• Global optimization / search for parameters
  – Least squares fit
  – Robust least squares
  – Iterative closest point (ICP)

• Hypothesize and test
  – Generalized Hough transform
  – RANSAC
Simple example: Fitting a line
Least squares line fitting

- **Data:** \((x_1, y_1), \ldots, (x_n, y_n)\)
- **Line equation:** \(y_i = mx_i + b\)
- **Find** \((m, b)\) **to minimize**

\[
E = \sum_{i=1}^{n} (y_i - mx_i - b)^2
\]

\[
E = \sum_{i=1}^{n} \left( \begin{bmatrix} x_i \\ 1 \end{bmatrix} \begin{bmatrix} m \\ b \end{bmatrix} - y_i \right)^2 = \left\| \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix} \begin{bmatrix} 1 \\ \vdots \\ 1 \end{bmatrix} \begin{bmatrix} m \\ b \end{bmatrix} - \begin{bmatrix} y_1 \\ \vdots \\ y_n \end{bmatrix} \right\|^2 = \|Ap - y\|^2
\]

\[
\]

\[
\frac{dE}{dp} = 2A^T Ap - 2A^T y = 0
\]

\[
A^T Ap = A^T y \implies p = \left( A^T A \right)^{-1} A^T y \quad \text{(Closed form solution)}
\]

**Matlab:** \(p = A \setminus y\);

Modified from S. Lazebnik
Example: solving for translation

Given matched points in \{A\} and \{B\}, estimate the translation of the object

\[
\begin{bmatrix}
  x_i^B \\
  y_i^B
\end{bmatrix}
= \begin{bmatrix}
  x_i^A \\
  y_i^A
\end{bmatrix} + \begin{bmatrix}
  t_x \\
  t_y
\end{bmatrix}
\]
Example: solving for translation

Least squares solution
1. Write down objective function
2. Derived solution
   a) Compute derivative
   b) Compute solution
3. Computational solution
   a) Write in form $Ax = p$
   b) Solve using closed-form solution

\[
\begin{bmatrix}
  x_i^B \\
  y_i^B
\end{bmatrix} = \begin{bmatrix}
  x_i^A \\
  y_i^A
\end{bmatrix} + \begin{bmatrix}
  t_x \\
  t_y
\end{bmatrix}
\]

\[
\begin{bmatrix}
  1 & 0 \\
  0 & 1 \\
  \vdots & \vdots \\
  1 & 0 \\
  0 & 1
\end{bmatrix} \begin{bmatrix}
  t_x \\
  t_y
\end{bmatrix} = \begin{bmatrix}
  x_1^B - x_1^A \\
  y_1^B - y_1^A \\
  \vdots \\
  x_n^B - x_n^A \\
  y_n^B - y_n^A
\end{bmatrix}
\]
- Two broad approaches:
  - Direct (pixel-based) alignment
    - Search for alignment where most pixels agree
  - Feature-based alignment
    - Search for alignment where extracted features agree
    - Can be verified using pixel-based alignment

Source: L. Lazebnik
Fitting an affine transformation

Affine model approximates perspective projection of planar objects.

Figures from David Lowe, ICCV 1999
Fitting an affine transformation

• Assuming we know the correspondences, how do we get the transformation?

\[
\begin{bmatrix}
  x'_i \\
  y'_i
\end{bmatrix} = \begin{bmatrix}
  m_1 & m_2 \\
  m_3 & m_4
\end{bmatrix} \begin{bmatrix}
  x_i \\
  y_i
\end{bmatrix} + \begin{bmatrix}
  t_1 \\
  t_2
\end{bmatrix}
\]
Fitting an affine transformation

- Assuming we know the correspondences, how do we get the transformation?

\[
\begin{bmatrix}
x'_i \\
y'_i
\end{bmatrix} = \begin{bmatrix} m_1 & m_2 \\ m_3 & m_4 \end{bmatrix} \begin{bmatrix} x_i \\
y_i
\end{bmatrix} + \begin{bmatrix} t_1 \\
t_2
\end{bmatrix}
\]
Fitting an affine transformation

\[
\begin{bmatrix}
    x_i & y_i & 0 & 0 & 1 & 0 \\
    0 & 0 & x_i & y_i & 0 & 1 \\
    \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\
\end{bmatrix}
\begin{bmatrix}
    m_1 \\
    m_2 \\
    m_3 \\
    m_4 \\
    t_1 \\
    t_2 \\
\end{bmatrix}
=
\begin{bmatrix}
    x'_i \\
    y'_i \\
    \vdots \\
\end{bmatrix}
\]

- How many matches (correspondence pairs) do we need to solve for the transformation parameters?
- Once we have solved for the parameters, how do we compute the coordinates of the corresponding point for \((x_{new}, y_{new})\)?
Panoramas

Obtain a wider angle view by combining multiple images.
How to stitch together a panorama?

• Basic Procedure
  – Take a sequence of images from the same position
    • Rotate the camera about its optical center
  – Compute transformation between second image and first
  – Transform the second image to overlap with the first
  – Blend the two together to create a mosaic
  – (If there are more images, repeat)

• …but **wait**, why should this work at all?
  – What about the 3D geometry of the scene?
  – Why aren’t we using it?

Source: Steve Seitz
Panoramas: generating synthetic views

Can generate any synthetic camera view as long as it has the same center of projection!

Source: Alyosha Efros
Image reprojection

The mosaic has a natural interpretation in 3D
- The images are reprojected onto a common plane
- The mosaic is formed on this plane
- Mosaic is a *synthetic wide-angle camera*

Source: Steve Seitz
Homography

How to relate two images from the same camera center?
  – how to map a pixel from PP1 to PP2?

Think of it as a 2D image warp from one image to another.

A projective transform is a mapping between any two PPs with the same center of projection
  • rectangle should map to arbitrary quadrilateral
  • parallel lines aren’t
  • but must preserve straight lines

called Homography

\[
\begin{bmatrix}
wx' \\
w & w
\end{bmatrix} = \begin{bmatrix}
* & * & * \\
* & * & * \\
* & * & *
\end{bmatrix} \begin{bmatrix}
x \\
y \\
1
\end{bmatrix}
\]

Source: Alyosha Efros
To apply a given homography \( H \)

- Compute \( p' = Hp \) (regular matrix multiply)
- Convert \( p' \) from homogeneous to image coordinates

\[
\begin{bmatrix}
w x' \\
w y' \\
w \\
p'
\end{bmatrix} = \begin{bmatrix}
* & * & * \\
* & * & * \\
* & * & * \\
H & &
\end{bmatrix} \begin{bmatrix}
x \\
y \\
1
\end{bmatrix}
\]
To **compute** the homography given pairs of corresponding points in the images, we need to set up an equation where the parameters of $H$ are the unknowns...
Number of measurements required

• At least as many independent equations as degrees of freedom required

• Example:

\[
\lambda \begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}
\]

2 independent equations / point
8 degrees of freedom
4x2 ≥ 8
Approximate solutions

- **Minimal solution**
  4 points yield an exact solution for $H$

- **More points**
  - No exact solution, because measurements are inexact ("noise")
  - Search for "best" according to some cost function
  - Algebraic or geometric/statistical cost
Gold Standard algorithm

• Cost function that is optimal for some assumptions
• Computational algorithm that minimizes it is called “Gold Standard” algorithm
• Other algorithms can then be compared to it
Direct Linear Transformation (DLT)

\[ \mathbf{x}_{i}' \propto H \mathbf{x}_i = 0 \]

\[ \mathbf{x}'_i = (x'_i, y'_i, w'_i)^T \]

\[ H \mathbf{x}_i = \begin{pmatrix} h^T \mathbf{x}_i \end{pmatrix} \]

\[ x'_i \times H \mathbf{x}_i = \begin{pmatrix} y'_i h^T x_i - w'_i h^T x_i \\ w'_i h^T x_i - x'_i h^T x_i \\ x'_i h^T x_i - y'_i h^T x_i \end{pmatrix} \]

\[ \begin{bmatrix} 0^T & -w'_i \mathbf{x}_i^T & y'_i \mathbf{x}_i^T \\ w'_i \mathbf{x}_i^T & 0^T & -x'_i \mathbf{x}_i^T \\ -y'_i \mathbf{x}_i^T & x'_i \mathbf{x}_i^T & 0^T \end{bmatrix} \begin{pmatrix} h^1 \\ h^2 \\ h^3 \end{pmatrix} = 0 \]

\[ A_i h = 0 \]
Direct Linear Transformation (DLT)

- Equations are linear in $h$
  $$A_i h = 0$$
- Only 2 out of 3 are linearly independent (indeed, 2 eq/pt)
  $$\begin{bmatrix}
  0^T & -w'_i x^T & y'_i x^T \\
  0^T & -w'_i x^T & y'_i x^T \\
  w'_i & 0 & -x'_i \\
  y'_i & 0 & -x'_i
  \end{bmatrix}
  \begin{pmatrix}
  h^1 \\
  h^2 \\
  h^3
  \end{pmatrix} = 0$$
  (only drop third row if $w'_i \neq 0$)
- Holds for any homogeneous representation, e.g. $(x'_i, y'_i, 1)$
Direct Linear Transformation (DLT)

• Solving for \( H \)

\[
\begin{bmatrix}
A_1 \\
A_2 \\
A_3 \\
A_4 \\
\end{bmatrix} h = 0
\]

size \( A \) is 8x9 or 12x9, but rank 8

Trivial solution is \( h = 0^T_9 \) is not interesting

1-D null-space yields solution of interest

pick for example the one with \( \| h \| = 1 \)
Direct Linear Transformation (DLT)

- Over-determined solution

$$\begin{bmatrix} A_1 \\ A_2 \\ \vdots \\ A_n \end{bmatrix} h = 0$$

No exact solution because of inexact measurement i.e. "noise"

Find approximate solution

- Additional constraint needed to avoid 0, e.g. $$\|h\| = 1$$
- $$Ah = 0$$ not possible, so minimize $$\|Ah\|$$
Singular Value Decomposition

\[ A = U \Sigma V^T \]

Homogeneous least-squares

\[
\min \|AX\| \quad \text{subject to} \quad \|X\| = 1
\]

solution \( X = V_n \)
DLT algorithm

**Objective**
Given $n \geq 4$ 2D to 2D point correspondences $\{x_i \leftrightarrow x_i'\}$, determine the 2D homography matrix $H$ such that $x_i' = Hx_i$

**Algorithm**
(i) For each correspondence $x_i \leftrightarrow x_i'$ compute $A_i$. Usually only two first rows needed.

(ii) Assemble $n$ 2x9 matrices $A_i$ into a single $2nx9$ matrix $A$

(iii) Obtain SVD of $A$. Solution for $h$ is last column of $V$

(iv) Determine $H$ from $h$
Inhomogeneous solution

Since $h$ can only be computed up to scale, pick $h_j=1$, e.g. $h_9=1$, and solve for 8-vector $\tilde{h}$

\[
\begin{bmatrix}
0 & 0 & 0 & -x_i w_i' & -y_i w_i' & -w_i w_i' & x_i y_i' & y_i y_i' \\
x_i w_i' & y_i w_i' & w_i w_i' & 0 & 0 & 0 & x_i x_i' & y_i x_i'
\end{bmatrix}
\begin{bmatrix}
\tilde{h}
\end{bmatrix} =
\begin{bmatrix}
-w_i y_i' \\
-w_i x_i'
\end{bmatrix}
\]

Solve using Gaussian elimination (4 points) or using linear least-squares (more than 4 points)

However, if $h_9=0$ this approach fails
also poor results if $h_9$ close to zero
Therefore, not recommended
Recap: How to stitch together a panorama?

• Basic Procedure
  – Take a sequence of images from the same position
    • Rotate the camera about its optical center
  – Compute transformation between second image and first
  – Transform the second image to overlap with the first
  – Blend the two together to create a mosaic
  – (If there are more images, repeat)

Source: Steve Seitz
Image warping with homographies

Source: Steve Seitz
Image rectification
Analysing patterns and shapes

What is the shape of the b/w floor pattern?

The floor (enlarged)

Automatically rectified floor

Slide from Criminisi
Analysing patterns and shapes

From Martin Kemp *The Science of Art* (manual reconstruction)

Slide from Criminisi
Analysing patterns and shapes

What is the (complicated) shape of the floor pattern?

Automatically rectified floor

*St. Lucy Altarpiece, D. Veneziano*

Slide from Criminisi
Analysing patterns and shapes

From Martin Kemp, *The Science of Art* (manual reconstruction)

Slide from Criminisi
changing camera center

Does it still work?

Source: Alyosha Efros
Planar scene (or far away)

PP3 is a projection plane of both centers of projection, so we are OK!

This is how big aerial photographs are made

Source: Alyosha Efros
Grauman
Outliers

- Outliers can hurt the quality of our parameter estimates, e.g.,
  - an erroneous pair of matching points from two images
  - an edge point that is noise, or doesn’t belong to the line we are fitting.
Example: least squares line fitting

- Assuming all the points that belong to a particular line are known
Outliers affect least squares fit
Outliers affect least squares fit
RANSAC

• RANdom Sample Consensus

• Approach: we want to avoid the impact of outliers, so let’s look for “inliers”, and use those only.

• Intuition: if an outlier is chosen to compute the current fit, then the resulting line won’t have much support from rest of the points.
RANSAC

- **RANSAC loop:**
  1. Randomly select a *seed group* of points on which to base transformation estimate (e.g., a group of matches)
  2. Compute transformation from seed group
  3. Find *inliers* to this transformation
  4. If the number of inliers is sufficiently large, re-compute least-squares estimate of transformation on all of the inliers

- Keep the transformation with the largest number of inliers
RANSAC

(Random Sample Consensus):

Fischler & Bolles in ‘81.
RANSAC

(RANdom SAmple Consensus):

Fischler & Bolles in ‘81.
RANSAC

(RANdom SAmple Consensus):

Fischler & Bolles in ’81.

This data is noisy, but we expect a good fit to a known model.
This data is noisy, but we expect a good fit to a known model.

Here, we expect to see a line, but least-squares fitting will produce the wrong result due to strong outlier presence.
RANSAC
(RANdom SAmple Consensus):

Fischler & Bolles in ‘81.

Algorithm:

1. Sample (randomly) the number of points $s$ required to fit the model
2. Solve for model parameters using samples
3. Score by the fraction of inliers within a preset threshold of the model

Repeat 1-3 until the best model is found with high confidence
RANSAC

Algorithm:
1. **Sample** (randomly) the number of points required to fit the model (s=2)
2. **Solve** for model parameters using samples
3. **Score** by the fraction of inliers within a preset threshold of the model

**Repeat** 1-3 until the best model is found with high confidence
RANSAC

Algorithm:

1. **Sample** (randomly) the number of points required to fit the model (s=2)
2. **Solve** for model parameters using samples
3. **Score** by the fraction of inliers within a preset threshold of the model

**Repeat** 1-3 until the best model is found with high confidence
RANSAC

Algorithm:
1. **Sample** (randomly) the number of points required to fit the model \( (s=2) \)
2. **Solve** for model parameters using samples
3. **Score** by the fraction of inliers within a preset threshold of the model

Repeat 1-3 until the best model is found with high confidence
Algorithm:

1. **Sample** (randomly) the number of points required to fit the model ($s=2$)
2. **Solve** for model parameters using samples
3. **Score** by the fraction of inliers within a preset threshold of the model

**Repeat** 1-3 until the best model is found with high confidence
How to choose parameters?

- **Number of samples** $N$
  - Choose $N$ so that, with probability $p$, at least one random sample is free from outliers (e.g., $p=0.99$) (outlier ratio: $e$)
- **Number of sampled points** $s$
  - Minimum number needed to fit the model
- **Distance threshold** $\delta$
  - Choose $\delta$ so that a good point with noise is likely (e.g., prob=0.95) within threshold
  - Zero-mean Gaussian noise with std. dev. $\sigma$: $t^2=3.84\sigma^2$

$$N = \log(1 - p) / \log\left(1 - (1 - e)^s\right)$$

<table>
<thead>
<tr>
<th>$e$</th>
<th>5%</th>
<th>10%</th>
<th>20%</th>
<th>25%</th>
<th>30%</th>
<th>40%</th>
<th>50%</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>2</td>
<td>3</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>11</td>
<td>17</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>4</td>
<td>7</td>
<td>9</td>
<td>11</td>
<td>19</td>
<td>35</td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td>5</td>
<td>9</td>
<td>13</td>
<td>17</td>
<td>34</td>
<td>72</td>
</tr>
<tr>
<td>5</td>
<td>4</td>
<td>6</td>
<td>12</td>
<td>17</td>
<td>26</td>
<td>57</td>
<td>146</td>
</tr>
<tr>
<td>6</td>
<td>4</td>
<td>7</td>
<td>16</td>
<td>24</td>
<td>37</td>
<td>97</td>
<td>293</td>
</tr>
<tr>
<td>7</td>
<td>4</td>
<td>8</td>
<td>20</td>
<td>33</td>
<td>54</td>
<td>163</td>
<td>588</td>
</tr>
<tr>
<td>8</td>
<td>5</td>
<td>9</td>
<td>26</td>
<td>44</td>
<td>78</td>
<td>272</td>
<td>1177</td>
</tr>
</tbody>
</table>

For $p = 0.99$

modified from M. Pollefeys
Reprojection error

\[ d(x, H^{-1}x')^2 + d(x', Hx)^2 \]

\[ d(x, \hat{x})^2 + d(x', \hat{x}')^2 \]
RANSAC example: Translation

Putative matches

Source: Rick Szeliski
RANSAC example: Translation

Select one match, count inliers
RANSAC example: Translation

Select one match, count inliers
RANSAC example: Translation

Find “average” translation vector
Feature-based alignment outline

Source: L. Lazebnik
Feature-based alignment outline

- Extract features

Source: L. Lazebnik
Feature-based alignment outline

- Extract features
- Compute *putative matches*

Source: L. Lazebnik
Feature-based alignment outline

- Extract features
- Compute *putative matches*
- Loop:
  - *Hypothesize* transformation $T$ (small group of putative matches that are related by $T$)

Source: L. Lazebnik
Feature-based alignment outline

- Extract features
- Compute *putative matches*
- Loop:
  - *Hypothesize* transformation $T$ (small group of putative matches that are related by $T$)
  - *Verify* transformation (search for other matches consistent with $T$)

Source: L. Lazebnik
Feature-based alignment outline

- Extract features
- Compute *putative matches*
- Loop:
  - *Hypothesize* transformation $T$ (small group of putative matches that are related by $T$)
  - *Verify* transformation (search for other matches consistent with $T$)

Source: L. Lazebnik
Towards large-scale mosaics...
Motion models

- Translation: 2 unknowns
- Affine: 6 unknowns
- Perspective: 8 unknowns
- 3D rotation: 3 unknowns

Szeliski
Plane perspective mosaics

- 8-parameter homographies

- Limitations:
  - local minima
  - slow convergence
  - difficult to control interactively
Rotational mosaics

– Directly optimize rotation and focal length

– Advantages:
  • ability to build full-view panoramas
  • easier to control interactively
  • more stable and accurate estimates
3D → 2D Perspective Projection

\[
\begin{bmatrix}
X_c \\
Y_c \\
Z_c
\end{bmatrix}
= \begin{bmatrix}
R
\end{bmatrix}_{3 \times 3}
\begin{bmatrix}
X \\
Y \\
Z
\end{bmatrix}
+ t
\]

\[
\begin{bmatrix}
u \\
v \\
1
\end{bmatrix}
\sim
\begin{bmatrix}
U \\
V \\
W
\end{bmatrix}
= \begin{bmatrix}
f & 0 & u_c \\
0 & f & v_c \\
0 & 0 & 1
\end{bmatrix}
\begin{bmatrix}
X_c \\
Y_c \\
Z_c
\end{bmatrix}
\]
PnP(1)
Perspective n-Point Problem

• Calibrated camera $\mathbf{K}$, $\mathbf{C} = (\mathbf{K}\mathbf{K}^T)^{-1}$
• $n$ point correspondences scene $\leftrightarrow$ image
• Known scene coordinates of $\mathbf{p}_i$, and known distances $d_{ij} = \| \mathbf{p}_i - \mathbf{p}_j \|$
• Each pair $(\mathbf{p}_i, \mathbf{p}_j)$ defines an angle $\theta$
• $\theta$ can be measured (2 lines of sight, calibrated camera)

$\Rightarrow$ constraint for the distance $\| \mathbf{c} - \mathbf{p}_j \|$
PnP (2)

searching: \[ x_i = \| p_i - c \|, \quad x_j = \| p_j - c \| \]

constraint: \[ d_{ij}^2 = x_i^2 + x_j^2 - 2x_i x_j \cos \theta_{ij} \]

\[ f_{ij}(x_i, x_j) = x_i^2 + x_j^2 - 2x_i x_j \cos \theta_{ij} - d_{ij}^2 = 0 \]

calibrated camera: \[ \cos \theta_{ij} = \frac{u_i^T C u_j}{\sqrt{u_i^T C u_i} \sqrt{u_j^T C u_j}} \]
PnP (3)

• P3P, 3 points: underdetermined, 4 solutions

\[
\begin{align*}
    f_{12}(x_1, x_2) &= 0 \\
    f_{13}(x_1, x_3) &= 0 \\
    f_{23}(x_2, x_3) &= 0
\end{align*}
\]

• P4P, 4 points:
  overdetermined, 6 equations, 4 unknowns

\[
\begin{align*}
    f_{12}(x_1, x_2) &= 0 \\
    f_{13}(x_1, x_3) &= 0 \\
    f_{23}(x_2, x_3) &= 0
\end{align*}
\]

\[
\begin{align*}
    1,2,3 \\
    1,2,4 \\
    1,3,4 \\
    2,3,4
\end{align*}
\]

4 x P3P, then find a common solution

• General problem: PnP, \( n \) points
PnP (4)

Once the $x_i$ have been solved:

1) project image points $\rightarrow$ scene
   \[ p'_i = x_i K^{-1} u_i \]

2) find a common $R, t$ for $p'_i \leftrightarrow p_i$
   (point-correspondences $\rightarrow$ solve a simple system of linear equations)
Application to faces

http://dlib.net


Matlab face frontalization http://www.openu.ac.il/home/hassner/code.html
Application to faces

// Reading image using OpenCV, you may use dlib as well.
std::vector<double> rv(3), tv(3);
cv::Mat rvec(rv), tvec(tv);
cv::Vec3d eav;

// Labelling the 3D Points derived from a 3D model of human face.
// You may replace these points as per your custom 3D head model if any
std::vector<cv::Point3f> modelPoints;
modelPoints.push_back(cv::Point3f(2.37427,110.322,21.7776)); // l eye (v 314)
modelPoints.push_back(cv::Point3f(70.0602,109.898,20.8234)); // r eye (v 0)
modelPoints.push_back(cv::Point3f(36.8301,78.3185,52.0345)); // nose (v 1879)
modelPoints.push_back(cv::Point3f(14.8498,51.0115,30.2378)); // l mouth (v 1502)
modelPoints.push_back(cv::Point3f(58.1825,51.0115,29.6224)); // r mouth (v 695)
modelPoints.push_back(cv::Point3f(-61.8886f,127.797,-89.4523f)); // l ear (v 2011)
modelPoints.push_back(cv::Point3f(127.603,126.9,-83.9129f)); // r ear (v 1138)

// labelling the position of corresponding feature points on the input image.
std::vector<cv::Point2f> srcImagePoints = {cv::Point2f(442,442), // left eye
cv::Point2f(529, 426), // right eye
cv::Point2f(501, 479), // nose
cv::Point2f(469, 534), // left lip corner
cv::Point2f(538, 521), // right lip corner
cv::Point2f(349, 457), // left ear
cv::Point2f(578, 415) // right ear};
Application to faces

cv::Mat ip(srcImagePoints);

cv::Mat op = cv::Mat(modelPoints);
cv::Scalar m = mean(cv::Mat(modelPoints));

rvec = cv::Mat(rv);
double _d[9] = {1,0,0, 0,-1,0, 0,0,-1};
Rodrigues(cv::Mat(3,3,CV_64FC1,_d),rvec);
tv[0]=0;tv[1]=0;tv[2]=1;
tvec = cv::Mat(tv);

double max_d = MAX(img.rows,img.cols);
double _cm[9] = {max_d, 0, (double)img.cols/2.0, 0, max_d, (double)img.rows/2.0, 0, 0, 1.0};
cv::Mat camMatrix = cv::Mat(3,3,CV_64FC1, _cm);

double _dc[] = {0,0,0,0};
solvePnP(op,ip,camMatrix,cv::Mat(1,4,CV_64FC1,_dc),rvec,tvec,false,CV_EPNP);
Application to faces

Head Pose Estimation using OpenCV and Dlib