More Mosaic Madness

© Jeffrey Martin (jeffrey-martin.com)

with a lot of slides stolen from
Steve Seitz and Rick Szeliski

Slides taken from
Alexei Efros
Least squares: Robustness to noise

Problem: squared error heavily penalizes outliers

outlier!
Fitting

Goal: Choose a parametric model to fit a certain quantity from data

Techniques:
• Least square methods
• RANSAC
• Hough transform
Basic philosophy (voting scheme)

- Data elements are used to vote for one (or multiple) models

- Robust to outliers and missing data

- **Assumption 1:** Noise features will not vote consistently for any single model ("few" outliers)

- **Assumption 2:** there are enough features to agree on a good model ("few" missing data)
RANSAC

(RANdom SAMple Consensus):
Learning technique to estimate parameters of a model by random sampling of observed data

Fischler & Bolles in ’81.

\[ \pi : I \rightarrow \{P, O\} \]

such that:
\[ f(P, \beta) < \delta \]

\[ \min_{\pi} |O| \]

Model parameters

\[ f(P, \beta) = \left\| \beta - \left(P^T P\right)^{-1} P^T \right\| \]

Silvio
Sample set = set of points in 2D

Algorithm:

1. Select random sample of minimum required size to fit model
2. Compute a putative model from sample set
3. Compute the set of inliers to this model from whole data set

Repeat 1-3 until model with the most inliers over all samples is found
Algorithm:

1. Select random sample of minimum required size to fit model \([?] = [2]\)
2. Compute a putative model from sample set
3. Compute the set of inliers to this model from whole data set

Repeat 1-3 until model with the most inliers over all samples is found

Silvio
Sample set = set of points in 2D

Algorithm:

1. Select random sample of minimum required size to fit model $[?] = [2]$

2. Compute a putative model from sample set

3. Compute the set of inliers to this model from whole data set

Repeat 1-3 until model with the most inliers over all samples is found
Sample set = set of points in 2D

Algorithm:

1. Select random sample of minimum required size to fit model $|?] = 2$
2. Compute a putative model from sample set
3. Compute the set of inliers to this model from whole data set

Repeat 1-3 until model with the most inliers over all samples is found

Silvio
RANSAC

(RANdom SAmple Consensus):

Fischler & Bolles in ‘81.

Algorithm:

1. Select random sample of minimum required size to fit model
2. Compute a putative model from sample set
3. Compute the set of inliers to this model from whole data set

Repeat 1-3 until model with the most inliers over all samples is found.
How many samples?

- **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: $e$)

- **Initial number of points** $s$
  - Typically minimum number needed to fit the model

- **Distance threshold** $\delta$
  - Choose $\delta$ so probability for inlier is $p$ (e.g. 0.95)
  - Zero-mean Gaussian noise with std. dev. $\sigma$: $\delta^2 = 3.84\sigma^2$

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

<table>
<thead>
<tr>
<th>$s$</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>
\[ e = \text{probability that a point is an outlier} \]
\[ s = \text{number of points in a sample} \]
\[ N = \text{number of samples (we want to compute this)} \]
\[ p = \text{desired probability that we get a good sample} \]

Solve the following for \( N \):

\[ 1 - (1 - (1 - e)^s)^N = p \]

Where in the world did that come from? ...
e = probability that a point is an outlier
s = number of points in a sample
N = number of samples (we want to compute this)
p = desired probability that we get a good sample

\[ 1 - \left(1 - (1 - e)^s\right)^N = p \]

**Probability that choosing one point yields an inlier**
$e =$ probability that a point is an outlier
$s =$ number of points in a sample
$N =$ number of samples (we want to compute this)
$p =$ desired probability that we get a good sample

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

Probability of choosing $s$ inliers in a row (sample only contains inliers)
e = probability that a point is an outlier
s = number of points in a sample
N = number of samples (we want to compute this)
p = desired probability that we get a good sample

\[ 1 - \left(1 - (1 - e)^s\right)^N = p \]

Probability that one or more points in the sample were outliers (sample is contaminated).
$e =$ probability that a point is an outlier  
$s =$ number of points in a sample  
$N =$ number of samples (we want to compute this)  
$p =$ desired probability that we get a good sample  

$$1 - (1 - (1 - e)^s)^N = p$$

Probability that $N$ samples were contaminated.
\[ 1 - \left( 1 - \left( 1 - e \right)^s \right)^N = p \]

**Probability that at least one sample was not contaminated**
(at least one sample of \( s \) points is composed of only inliers).
Choose $N$ so that, with probability $p$, at least one random sample is free from outliers. e.g. $p=0.99$

\[ (1 - (1 - e)^s)^N = 1 - p \]

\[ N = \frac{\log(1 - p)}{\log \left( 1 - (1 - e)^s \right)} \]

<table>
<thead>
<tr>
<th>$e$</th>
<th>proportion of outliers</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></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></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></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></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></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></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></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>
- \( n = 12 \) points
- Minimal sample size \( s = 2 \)
- 2 outliers: \( e = \frac{1}{6} \Rightarrow 20\% \)
- So \( N = 5 \) gives us a 99\% chance of getting a pure-inlier sample
  - Compared to \( N = 66 \) by trying every pair of points

from Hartley & Zisserman
• We have seen that we don’t have to exhaustively sample subsets of points, we just need to randomly sample N subsets.

• However, typically, we don’t even have to sample N sets!

• **Early termination**: terminate when inlier ratio reaches expected ratio of inliers

\[ T = (1 - e) \times (\text{total number of data points}) \]
Rotation about vertical axis

What if our camera rotates on a tripod? What’s the structure of $H$?
Do we have to project onto a plane?
Full Panoramas

What if you want a $360^\circ$ field of view?

mosaic Projection Cylinder
Cylindrical projection

- Map 3D point \((X,Y,Z)\) onto cylinder
  \[
  (\tilde{x}, \tilde{y}, \tilde{z}) = \frac{1}{\sqrt{X^2+Z^2}}(X, Y, Z)
  \]
- Convert to cylindrical coordinates
  \[
  (\sin \theta, h, \cos \theta) = (\tilde{x}, \tilde{y}, \tilde{z})
  \]
- Convert to cylindrical image coordinates
  \[
  (\bar{x}, \bar{y}) = (f\theta, fh) + (\bar{x}_c, \bar{y}_c)
  \]
Cylindrical Projection
Inverse Cylindrical projection

\[ \theta = \frac{(x_{cyl} - x_c)}{f} \]
\[ h = \frac{(y_{cyl} - y_c)}{f} \]
\[ \hat{x} = \sin \theta \]
\[ \hat{y} = h \]
\[ \hat{z} = \cos \theta \]
\[ x = \frac{f \hat{x}}{\hat{z}} + x_c \]
\[ y = \frac{f \hat{y}}{\hat{z}} + y_c \]
Cylindrical panoramas

Steps

- Reproject each image onto a cylinder
- Blend
- Output the resulting mosaic
Cylindrical image stitching

What if you don’t know the camera rotation?

- Solve for the camera rotations
  - Note that a rotation of the camera is a translation of the cylinder!
Assembling the panorama

Stitch pairs together, blend, then crop
Problem: Drift

Vertical Error accumulation
- small (vertical) errors accumulate over time
- apply correction so that sum = 0 (for 360° pan.)

Horizontal Error accumulation
- can reuse first/last image to find the right panorama radius
Full-view (360°) panoramas
Spherical projection

- Map 3D point \((X,Y,Z)\) onto sphere
  \[
  (\hat{x}, \hat{y}, \hat{z}) = \frac{1}{\sqrt{X^2 + Y^2 + Z^2}}(X,Y,Z)
  \]
- Convert to spherical coordinates
  \[
  (\sin \theta \cos \phi, \sin \phi, \cos \theta \cos \phi) = (\hat{x}, \hat{y}, \hat{z})
  \]
- Convert to spherical image coordinates
  \[
  (\tilde{x}, \tilde{y}) = (f \theta, f h) + (\tilde{x}_c, \tilde{y}_c)
  \]
Spherical Projection
Inverse Spherical projection

\[ \theta = \frac{(x_{sp} - x_c)}{f} \]
\[ \varphi = \frac{(y_{sp} - y_c)}{f} \]
\[ \hat{x} = \sin \theta \cos \varphi \]
\[ \hat{y} = \sin \varphi \]
\[ \hat{z} = \cos \theta \cos \varphi \]
\[ x = f \hat{x}/\hat{z} + x_c \]
\[ y = f \hat{y}/\hat{z} + y_c \]
3D rotation

Rotate image before placing on unrolled sphere

\[
\begin{align*}
\theta &= \frac{(x_{sph} - x_c)}{f} \\
\varphi &= \frac{(y_{sph} - y_c)}{f} \\
\hat{x} &= \sin \theta \cos \varphi \\
\hat{y} &= \sin \varphi \\
\hat{z} &= \cos \theta \cos \varphi \\
x &= f \frac{\hat{x}}{\hat{z}} + x_c \\
y &= f \frac{\hat{y}}{\hat{z}} + y_c
\end{align*}
\]
Full-view Panorama
Other projections are possible

You can stitch on the plane and then warp the resulting panorama
  • What’s the limitation here?
Or, you can use these as stitching surfaces
  • But there is a catch…
Cylindrical reprojection

Focal length – the dirty secret…

Image 384x300  f = 180 (pixels)  f = 280  f = 380
What’s your focal length, buddy?

Focal length is (highly!) camera dependant

- Can get a rough estimate by measuring FOV:

- Can use the EXIF data tag (might not give the right thing)
- Can use several images together and try to find f that would make them match
- Can use a known 3D object and its projection to solve for f
- Etc.

There are other camera parameters too:

- Optical center, non-square pixels, lens distortion, etc.
Distortion

Radial distortion of the image

- Caused by imperfect lenses
- Deviations are most noticeable for rays that pass through the edge of the lens
Radial distortion

Correct for “bending” in wide field of view lenses

\[
\hat{r}^2 = \hat{x}^2 + \hat{y}^2
\]
\[
\hat{x}' = \hat{x}/(1 + \kappa_1 \hat{r}^2 + \kappa_2 \hat{r}^4)
\]
\[
\hat{y}' = \hat{y}/(1 + \kappa_1 \hat{r}^2 + \kappa_2 \hat{r}^4)
\]
\[
x = f \hat{x}'/\hat{z} + x_c
\]
\[
y = f \hat{y}'/\hat{z} + y_c
\]

Use this instead of normal projection
Polar Projection

Extreme “bending” in ultra-wide fields of view

\[ \hat{r}^2 = \hat{x}^2 + \hat{y}^2 \]

\[(\cos \theta \sin \phi, \sin \theta \sin \phi, \cos \phi) = s (x, y, z)\]

Equations become

\[ x' = s\phi \cos \theta = s \frac{x}{r} \tan^{-1} \frac{r}{z} \]

\[ y' = s\phi \sin \theta = s \frac{y}{r} \tan^{-1} \frac{r}{z} \]
Camera calibration

Determine camera parameters from *known* 3D points or calibration object(s)

1. *internal* or *intrinsic* parameters such as focal length, optical center, aspect ratio: *what kind of camera?*

2. *external* or *extrinsic* (pose) parameters: *where is the camera in the world coordinates?*
   - World coordinates make sense for multiple cameras / multiple images

How can we do this?
Approach 1: solve for projection matrix

Place a known object in the scene

- identify correspondence between image and scene
- compute mapping from scene to image

\[
\begin{bmatrix}
  u_i \\
v_i \\
  1
\end{bmatrix}
\begin{bmatrix}
  m_{00} & m_{01} & m_{02} & m_{03} \\
  m_{10} & m_{11} & m_{12} & m_{13} \\
  m_{20} & m_{21} & m_{22} & m_{23}
\end{bmatrix}
\begin{bmatrix}
  X_i \\
  Y_i \\
  Z_i \\
  1
\end{bmatrix}
\]
Direct linear calibration

\[
\begin{bmatrix}
 u_i \\
 v_i \\
 1
\end{bmatrix}
= \begin{bmatrix}
 m_{00} & m_{01} & m_{02} & m_{03} \\
 m_{10} & m_{11} & m_{12} & m_{13} \\
 m_{20} & m_{21} & m_{22} & m_{23}
\end{bmatrix}
\begin{bmatrix}
 X_i \\
 Y_i \\
 Z_i \\
 1
\end{bmatrix}
\]

Solve for Projection Matrix $\Pi$ using least-squares (just like in homework)

Advantages:
- All specifics of the camera summarized in one matrix
- Can predict where any world point will map to in the image

Disadvantages:
- Doesn’t tell us about particular parameters
- Mixes up internal and external parameters
  - pose specific: move the camera and everything breaks
Approach 2: solve for parameters

A camera is described by several parameters

- Translation $T$ of the optical center from the origin of world coords
- Rotation $R$ of the image plane
- focal length $f$, principle point $(x'_c, y'_c)$, pixel size $(s_x, s_y)$
- blue parameters are called “extrinsics,” red are “intrinsics”

Projection equation

$$X = \begin{bmatrix} sx \\ sy \\ s \end{bmatrix} = \begin{bmatrix} * & * & * \\ * & * & * \\ * & * & * \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = \Pi X$$

- The projection matrix models the cumulative effect of all parameters
- Useful to decompose into a series of operations

$$\Pi = \begin{bmatrix} -fs_x & 0 & x'_c \\ 0 & -fs_y & y'_c \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} R_{3x3} & 0_{3x1} \\ 0_{1x3} & 1 \end{bmatrix} \begin{bmatrix} I_{3x3} & T_{3x1} \end{bmatrix}$$

- Solve using non-linear optimization
Multi-plane calibration

Images courtesy Jean-Yves Bouguet, Intel Corp.

**Advantage**

- Only requires a plane
- Don’t have to know positions/orientations
- Good code available online!