RANSAC
Identifying Correct Matches

- Let’s say that we want to match points from one image to another
- We detect feature points in each image (e.g., SIFT or SURF points) and then match their descriptors
- Now we have a set of matching pairs, but some are incorrect
- How do we identify the correct matches?
SIFT Points from each image

Number of features detected: 1741
Number of features detected: 2052
Tentative matches (based on feature descriptors): 442
Demonstrate the matching of a plane, from one image to another.

clear variables
close all

% rng(0);     % Reset random number generator

% Read in test images.
I1 = imread('img1.png');
I2 = imread('img3.png');

% If color, convert them to grayscale
if size(I1,3)>1I1 = rgb2gray(I1); end
if size(I2,3)>1I2 = rgb2gray(I2); end

% Extract SIFT features.

% First make sure the vl_sift code is in the path
if exist('vl_sift', 'file')==0
    run('C:\Users\William\Documents\Research\vlfeat-0.9.20\toolbox\vl_setup');
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% First image
I1 = single(I1); % Convert to single precision floating point
figure(1), imshow(I1);%

% These parameters limit the number of features detected
peak_thresh = 0; % increase to limit; default is 0
edge_thresh = 10; % decrease to limit; default is 10
[f1,d1] = vl_sift(I1, ...    
    'PeakThresh', peak_thresh, ...
    'edgethresh', edge_thresh);
fprintf('Number of frames (features) detected: %d\n\n', size(f1,2));

% Show all SIFT features detected
h = vl_plotframe(f1);
set(h,'color','y','linewidth',1.5);
Find SIFT points in images (2 of 3)

\begin{verbatim}
\% Second image
I2 = single(I2);
figure(2), imshow(I2,[]);

[f2,d2] = vl_sift(I2, ...
    'PeakThresh', peak_thresh, ...
    'edgethresh', edge_thresh);
fprintf('Number of frames (features) detected: %d\n', size(f2,2));

% Show all SIFT features detected
h   = vl_plotframe(f2); set(h,'color','y','linewidth',1.5);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Match features
% The index of the original match and the closest descriptor is stored in
% each column of matches and the distance between the pair is stored in
% scores.

% Define threshold for matching. Descriptor D1 is matched to a descriptor
% D2 only if the distance d(D1,D2) multiplied by THRESH is not greater than
% the distance of D1 to all other descriptors
thresh = 1.5;  % default = 1.5; increase to limit matches
[matches, scores] = vl_ubcmatch(d1, d2, thresh);
fprintf('Number of matching frames (features): %d\n', size(matches,2));

indices1 = matches(1,:);  % Get matching features
f1match = f1(:,indices1);
d1match = d1(:,indices1);

indices2 = matches(2,:);
f2match = f2(:,indices2);
d2match = d2(:,indices2);
\end{verbatim}
Find SIFT points in images (3 of 3)

```matlab
% Show matches
figure(3), imshow([I1,I2],[]);
o = size(I1,2);
line([f1match(1,:);f2match(1,:)+o], ... 
    [f1match(2,:);f2match(2,:)]);
for i=1:size(f1match,2)
    x = f1match(1,i);
    y = f1match(2,i);
    text(x,y,sprintf('%d',i), 'Color', 'r');
end
for i=1:size(f2match,2)
    x = f2match(1,i);
    y = f2match(2,i);
    text(x+o,y,sprintf('%d',i), 'Color', 'r');
end
```
Fitting a Transformation

• We want to find a transformation between the two images

• What transformation to use?
  – Depends on your application ... for a planar object, or a rotating camera, you could use a homography (projective transform)

• We can calculate this transformation from corresponding points
  – To fit a homography, this requires at least 4 corresponding image points
  – However, incorrect correspondences will corrupt our estimate
Robust Regression

• Ordinary least-squares fitting minimizes an objective function, which is the sum of squared errors
• If there are outliers, they can have very large errors, so they “skew” the fitting
• One solution is to use a modified objective function (see next slide), which down-weights the influence of large errors
  – Least median of squares (LMS) can also be used
• This works well if there aren’t that many outliers (e.g., 10% or 20% or so)
These are called “M-estimators” because they are (M)inimizing a loss function.

Also, see MATLAB’s “robustdemo”.

Large Numbers of Outliers

• If the outliers are a large fraction of the data, then M-estimators don’t work well

• Instead we need a type of clustering, or voting method
  – The Hough transform method in Lowe’s original SIFT paper is an example of this

• Another method is called RANSAC (Random Sample Consensus)
  – RANSAC has been widely used in many engineering applications
Algorithm (RANSAC)

1) Randomly select a minimal set of points and solve for the transformation
2) Count the number of points that agree with this transformation
3) If this transformation has the highest number of inliers so far, save it
4) Repeat for $N$ trials and return the best transformation
Example: line fitting

We will randomly select subsets of 2 points (since 2 is the minimum number of points to fit a line)
Example: line fitting
Model fitting
Measure distances
Count inliers

\( c = 3 \)
Another trial

\[ c = 3 \]
The best model

\[ c = 15 \]
RANSAC Algorithm

• “Random Sample Consensus”
  – Can be used to find inliers for any type of model fitting

• Widely used in computer vision

• Requires two parameters:
  – The number of trials $N$ (how many do we need?)
  – The agreement threshold (how close does an inlier have to be?)

Number of samples (trials) required

• Assume we know $\varepsilon$, the probability that a point is an outlier
  – So $(1 - \varepsilon)$ is the probability that any point is an inlier

• The probability that a sample of size $s$ is all inliers is $(1 - \varepsilon)^s$
  – So $1 - (1 - \varepsilon)^s$ is the probability of getting at least one outlier in that sample

• With $N$ samples (trials), the probability that all samples have at least one outlier is $p_{\text{all outliers}} = (1 - (1 - \varepsilon)^s)^N$

• We want at least one sample to have no outliers. The probability is $p_{\text{not all outliers}} = 1 - p_{\text{all outliers}}$

• We’ll pick $p_{\text{not all outliers}}$ to be some high value, like 0.99.

• If we know $\varepsilon$, we can solve for $N$

$$N = \frac{\log(1 - p_{\text{not all outliers}})}{\log(1 - (1 - \varepsilon)^s)}$$

$N$ is the number of trials needed to get at least one sample of all inliers, with probability $p_{\text{not all outliers}}$

Try example:
$\varepsilon = 0.5$
$p = 0.99$
$s = 2$
$N = 10$
Number of samples required

<table>
<thead>
<tr>
<th>Sample size</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>

Table 1: From [Hartley and Zisserman, 2000]: The number $N$ of samples required to ensure, with a probability $p = 0.99$, that at least one sample has no outliers for a given sample size $s$ and a proportion of outliers $\epsilon$.

$$N = \frac{\log(1 - p)}{\log(1 - (1 - \epsilon)^s)}.$$
Better estimate for required number of trials

- Again, assume we know \( \varepsilon \), the probability that a point is an outlier
  - So \((1 - \varepsilon)\) is the probability that any point is an inlier

- Assume we have \( M \) total points. The estimated number of inliers is \( n_{inlier} = (1 - \varepsilon)M \)

- The number of ways we can pick a sample of \( s \) points out of \( M \) points is \( m = \binom{M}{s} \)

- The number of ways we can choose \( s \) points that are all inliers is \( n = \binom{n_{inlier}}{s} \)

- The probability that any sample of \( s \) points is all inliers is \( p = n/m \)

\[
N = \frac{\log(1 - p_{notalloutliers})}{\log(1 - p)}
\]

\( N \) is the number of trials needed to get at least one sample of all inliers, with probability \( p_{notalloutliers} \)

“\( n \text{ choose } k \)” is the number of ways to choose \( k \) items from a set of \( n \) items:

\[
\binom{n}{k} = \frac{n!}{k!(n-k)!}
\]
Estimating outlier probability

• If you don’t know the outlier probability $\varepsilon$ in advance, you can estimate it from the data:
  – Start with a very conservative guess for $\varepsilon$ (say, 0.9)
  – Compute the number of trials required ... it will be very large
  – Each time you get a fit, count the number of inliers
  – If the number of inliers is the best so far, revise your estimate of $\varepsilon$: $\varepsilon = (1 - n_{\text{inlier}})/M$
  – Re-compute the number of trials required using the new estimate of $\varepsilon$ and continue
Deciding if a point is an inlier

- You can just pick a threshold for a point’s residual error, for it to be counted as an inlier (e.g., you could measure this empirically)

- Another strategy:
  - Compute the conditional probability of the residual assuming that it is an inlier, and compare it to the conditional probability of the residual assuming that it is an outlier
  - Namely, test if $P(r|\text{inlier}) > P(r|\text{outlier})$
  - Typically, we assume inliers have Gaussian distributed noise (with standard deviation $\sigma$), and outliers have uniform distributed noise
  - This approach is called MLESAC (“maximum likelihood estimator sample consensus”)

Demo Program

• This program matches two images using a homography
  – So the two images should be images of a plane
  – Or, the two images should be taken with a rotating (but not translating) camera

• Files
  – main.m
    • Reads the images, detects SIFT features, finds tentative matches
  – fitHomographyRansac.m
    • A function that takes tentative matching features, and returns the homography (if possible)
  – imfuseWarped.m
    • A function that fuses two images by blending
Demo Program

• (Go through Matlab code)

• Demonstrate on sample images
  – img1.png, img3.png (from the graffiti dataset)

  – floor1.jpg, floor2.jpg

  – table1.jpg, table2.jpg
372 inliers

difference image

fused image