Image alignment

Subhransu Maji

CMPSCI 670: Computer Vision

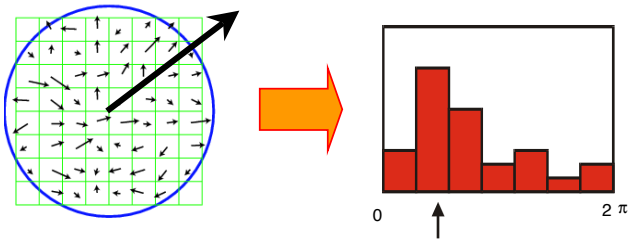
October 18, 2016

Recap: from feature detection to description

- Scaled and rotated versions of the same neighborhood will give rise to blobs that are related by the same transformation
- What to do if we want to compare the appearance of these image regions?
  - Normalization: transform these regions into same-size circles
  - Problem: rotational ambiguity

Recap: Eliminating rotation ambiguity

- To assign a unique orientation to circular image windows:
  - Create histogram of local gradient directions in the patch
  - Assign canonical orientation at peak of smoothed histogram



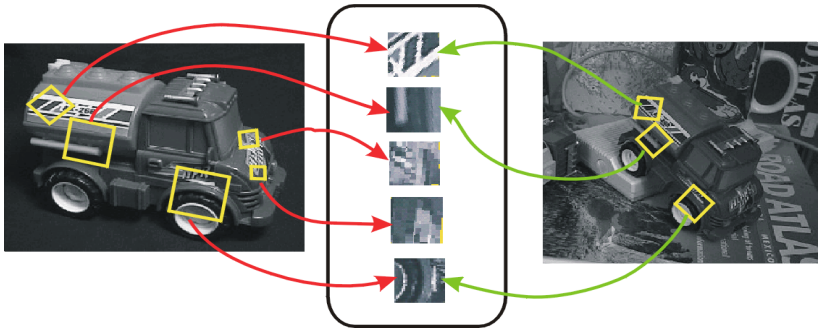
Recap: Local features

- Detected features with characteristic scales and orientations

David G. Lowe. **"Distinctive image features from scale-invariant keypoints."** *IJCV* 60 (2), pp. 91-110, 2004.

Feature descriptors

- Find **regions** and compare their **feature descriptors**



Feature descriptors

- Simplest descriptor: vector of raw intensity values
- How to compare two such vectors?
  - Sum of squared differences (SSD) — this is a distance measure

$$\text{SSD}(\mathbf{u}, \mathbf{v}) = \sum_i (u_i - v_i)^2$$

    - Not invariant to intensity change

  - Normalized correlation — this is a similarity measure

$$\rho(\mathbf{u}, \mathbf{v}) = \frac{(\mathbf{u} - \bar{\mathbf{u}})}{\|\mathbf{u} - \bar{\mathbf{u}}\|} \cdot \frac{(\mathbf{v} - \bar{\mathbf{v}})}{\|\mathbf{v} - \bar{\mathbf{v}}\|} = \frac{\sum_i (u_i - \bar{\mathbf{u}})(v_i - \bar{\mathbf{v}})}{\sqrt{\left(\sum_j (u_j - \bar{\mathbf{u}})^2\right)\left(\sum_j (v_j - \bar{\mathbf{v}})^2\right)}}$$

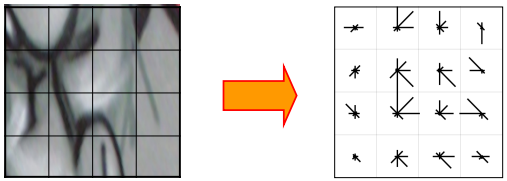
    - Invariant to affine (translation + scaling) intensity change

Problem with intensity vectors

- Small deformations can affect the matching score a lot

Feature descriptors: SIFT

- Descriptor computation:
  - Divide patch into 4x4 sub-patches
  - Compute histogram of gradient orientations (8 reference angles) inside each sub-patch
  - Resulting descriptor: 4x4x8 = 128 dimensions



David G. Lowe. "Distinctive image features from scale-invariant keypoints." *IJCV* 60 (2), pp. 91-110, 2004.

Feature descriptors: SIFT

- Descriptor computation:
  - Divide patch into 4x4 sub-patches
  - Compute histogram of gradient orientations (8 reference angles) inside each sub-patch
  - Resulting descriptor: 4x4x8 = 128 dimensions
- Advantage over raw vectors of pixel values
  - Gradients less sensitive to illumination change
  - Pooling of gradients over the sub-patches achieves robustness to small shifts, but still preserves some spatial information

More on image representations later

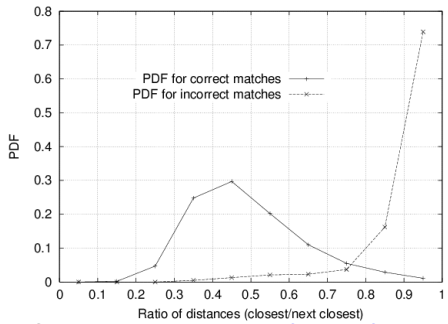
David G. Lowe. "Distinctive image features from scale-invariant keypoints." *IJCV* 60 (2), pp. 91-110, 2004.

Problem: Ambiguous putative matches

Source: Y. Furukawa

Rejection of unreliable matches

- How can we tell which putative matches are more reliable?
- Heuristic: compare distance of **nearest** neighbor to that of **second** nearest neighbor
  - Ratio of closest distance to second-closest distance will be *high* for features that are *not* distinctive



Threshold of 0.8 provides good separation

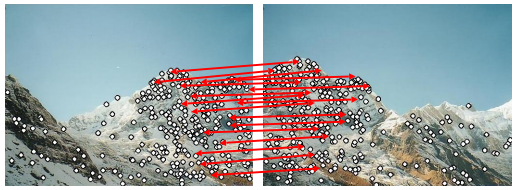
David G. Lowe. "Distinctive image features from scale-invariant keypoints." *IJCV* 60 (2), pp. 91-110, 2004.

A framework for alignment

- Matching local features
  - Local information used, can contain outliers
  - But hopefully enough of these matches are good

A framework for alignment

- Matching local features
  - Local information used, can contain outliers
  - But hopefully enough of these matches are good
- Consensus building
  - Aggregate the good matches and find a transformation that explains these matches



Families of transformation

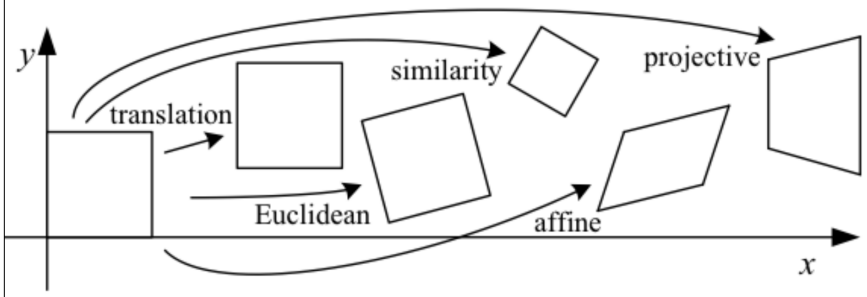


Figure 2.4 Basic set of 2D planar transformations.

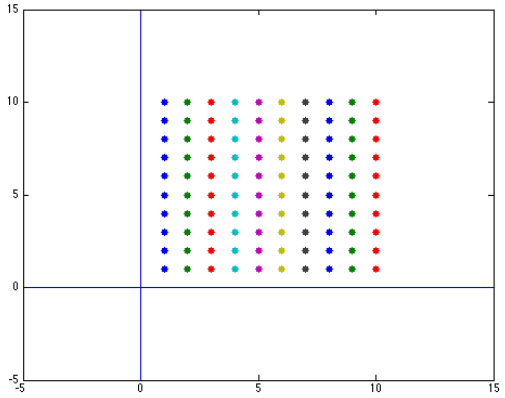
From Computer Vision: Algorithms and Applications, by Rick Szeliski

How do we move pixels?

- Think about moving coordinates, not pixels.

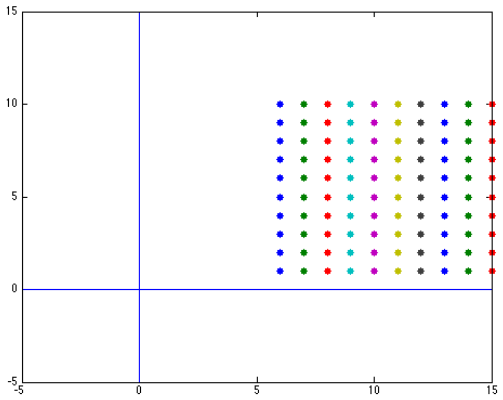
Points in 2D

```
figure(1);
[ox oy]=meshgrid(1:10,1:10);
plot(ox,oy,'.');
line([0 0],[-5 15]);
line([-5 15],[0 0]);
axis([-5 15 -5 15]);
```



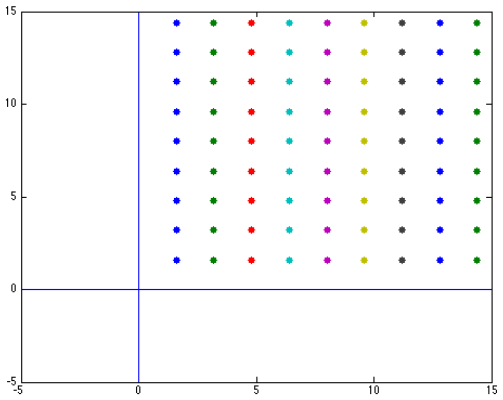
Translation

```
figure(2);
nx = ox + 5;
ny = oy;

plot(nx,ny,'.');
line([0 0],[-5 15]);
line([-5 15],[0 0]);
axis([-5 15 -5 15]);
```



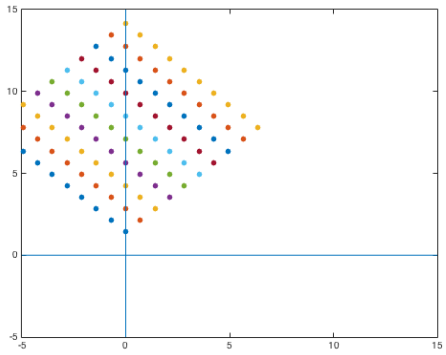
Scaling

```
figure(3);
oxy = [ox(:)'; oy(:)'];
A = [1.6 0; 0 1.6];
nxy = A*oxy;

nx = nxy(1,:);
ny = nxy(2,:);
nx = reshape(nx, [10 10]);
ny = reshape(ny, [10 10]);
plot(nx,ny,'.');
line([0 0],[-5 15]);
line([-5 15],[0 0]);
axis([-5 15 -5 15]);
```



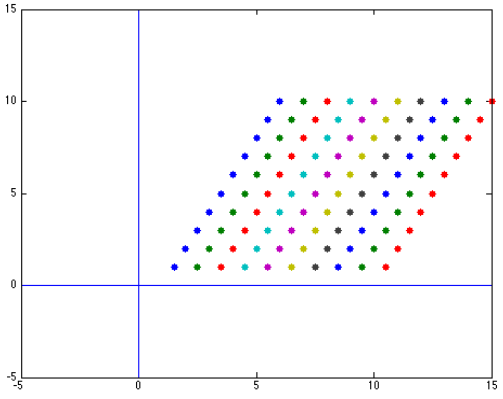
Rotation

```
figure(4);
oxy = [ox(:)'; oy(:)'];
A = [.707 -.707;.707 .707];
nxy = A*oxy;

nx = nxy(1,:);
ny = nxy(2,:);
nx = reshape(nx,[10 10]);
ny = reshape(ny,[10 10]);
plot(nx,ny,'.');
line([0 0],[-5 15]);
line([-5 15],[0 0]);
axis([-5 15 -5 15]);
```



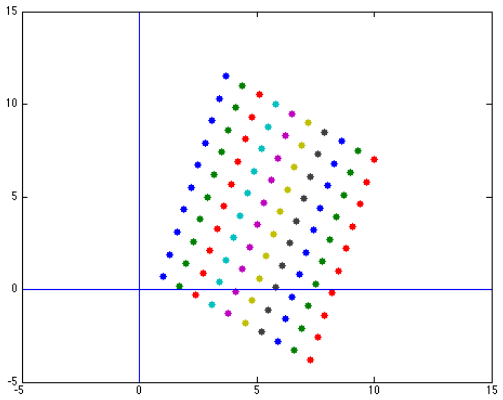
Shear

```
figure(5);
oxy = [ox(:)'; oy(:)'];
A = [1 0.5; 0 1];
nxy = A*oxy;

nx = nxy(1,:);
ny = nxy(2,:);
nx = reshape(nx,[10 10]);
ny = reshape(ny,[10 10]);
plot(nx,ny,'.');
line([0 0],[-5 15]);
line([-5 15],[0 0]);
axis([-5 15 -5 15]);
```



Arbitrary linear transformation

```
figure(6);
oxy = [ox(:)'; oy(:)'];
A = [0.7 0.3; -0.5 1.2];
nxy = A*oxy;

nx = nxy(1,:);
ny = nxy(2,:);
nx = reshape(nx,[10 10]);
ny = reshape(ny,[10 10]);
plot(nx,ny,'.');
line([0 0],[-5 15]);
line([-5 15],[0 0]);
axis([-5 15 -5 15]);
```



Families of linear transforms

Identity:

$$\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

Uniform scaling:

$$\begin{bmatrix} s & 0 \\ 0 & s \end{bmatrix}$$

Scaling in x:

$$\begin{bmatrix} s_x & 0 \\ 0 & 1 \end{bmatrix}$$

Rotation by θ radians:

$$\begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix}$$

Shearing in x:

$$\begin{bmatrix} 1 & sh_x \\ 0 & 1 \end{bmatrix}$$

Arbitrary linear transformation:

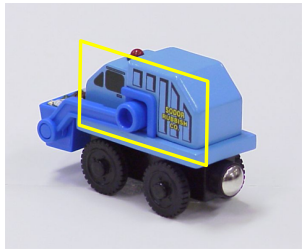
$$\begin{bmatrix} a & b \\ c & d \end{bmatrix}$$

Affine transformations

- Affine = linear + translation

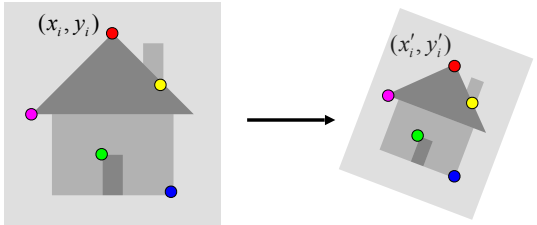
$$\mathbf{x} \rightarrow \mathbf{M}\mathbf{x} + \mathbf{t}$$

- Approximates viewpoint changes for roughly planar objects and roughly orthographic cameras
- Can be used to initialize fitting for more complex models



Fitting an affine transformation

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



$$\begin{bmatrix} x_i' \\ y_i' \end{bmatrix} = \begin{bmatrix} m_1 & m_2 \\ m_3 & m_4 \end{bmatrix} \begin{bmatrix} x_i \\ y_i \end{bmatrix} + \begin{bmatrix} t_1 \\ t_2 \end{bmatrix}$$

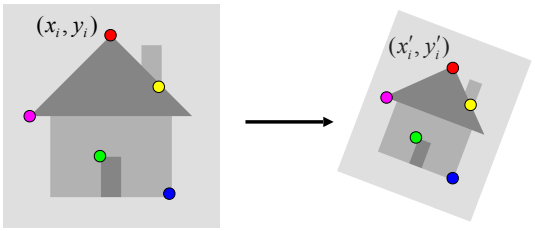
$$\mathbf{x}_i' = \mathbf{M}\mathbf{x}_i + \mathbf{t}$$

Want to find **M**, **t** to minimize

$$\sum_{i=1}^{n} || \mathbf{x}_i' - \mathbf{M}\mathbf{x}_i - \mathbf{t} ||^2$$

Fitting an affine transformation

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



$$\begin{bmatrix} x_i' \\ y_i' \end{bmatrix} = \begin{bmatrix} m_1 & m_2 \\ m_3 & m_4 \end{bmatrix} \begin{bmatrix} x_i \\ y_i \end{bmatrix} + \begin{bmatrix} t_1 \\ t_2 \end{bmatrix}$$

$$\begin{bmatrix} & & \cdots & & & \\ x_i & y_i & 0 & 0 & 1 & 0 \\ 0 & 0 & x_i & y_i & 0 & 1 \\ & & \cdots & & & \end{bmatrix} \begin{bmatrix} m_1 \\ m_2 \\ m_3 \\ m_4 \\ t_1 \\ t_2 \end{bmatrix} = \begin{bmatrix} \cdots \\ x_i' \\ y_i' \\ \cdots \end{bmatrix}$$

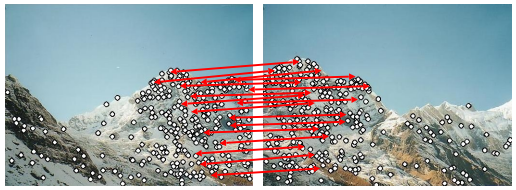
Fitting an affine transformation

- Linear system with six unknowns
- Each match gives us two linearly independent equations: need at least three to solve for the transformation parameters

$$\begin{bmatrix} & & \cdots & & & \\ x_i & y_i & 0 & 0 & 1 & 0 \\ 0 & 0 & x_i & y_i & 0 & 1 \\ & & \cdots & & & \end{bmatrix} \begin{bmatrix} m_1 \\ m_2 \\ m_3 \\ m_4 \\ t_1 \\ t_2 \end{bmatrix} = \begin{bmatrix} \cdots \\ x_i' \\ y_i' \\ \cdots \end{bmatrix}$$

A framework for alignment

- Matching local features
  - Local information used, can contain outliers
  - But hopefully enough of these matches are good
- Consensus building
  - Aggregate the good matches and find a transformation that explains these matches

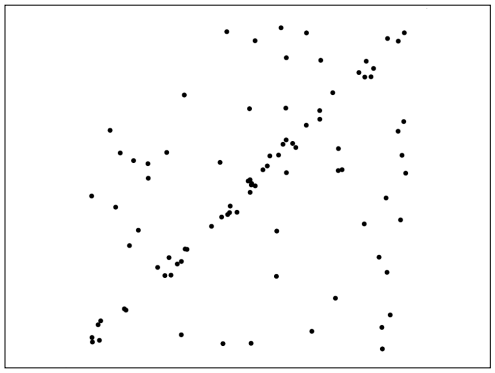


RANSAC

- **Ran**dom **Sa**mple **C**onsensus
  - Choose a small subset of points uniformly at random
  - Fit a model to that subset
  - Find all remaining points that are "close" to the model and reject the rest as outliers
  - Do this many times and choose the best model
- For rigid transformation we can estimate the parameters of the transformation, e.g., rotation angle, scaling, translation, etc, from putative correspondence matches
- Lets see how **RANSAC** works for a simple example.

M. A. Fischler, R. C. Bolles. Random Sample Consensus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography. Comm. of the ACM, Vol 24, pp 381-395, 1981.

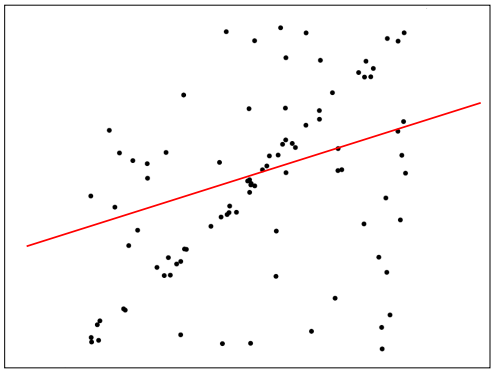
RANSAC for line fitting example



Source: R. Raguram

RANSAC for line fitting example

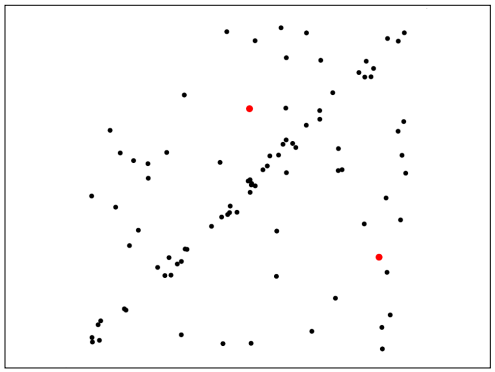
$$\min_{a,b} \sum_i (ax_i + b - y_i)^2$$



Least-squares fit

Source: R. Raguram

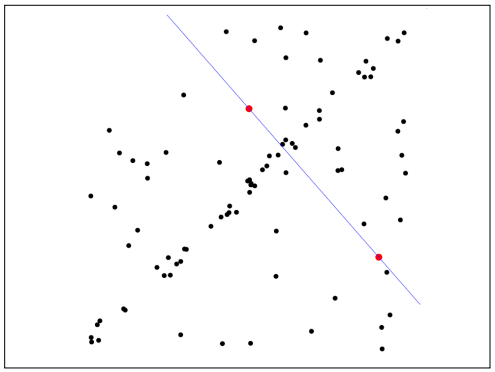
RANSAC for line fitting example



1. Randomly select minimal subset of points

Source: R. Raguram

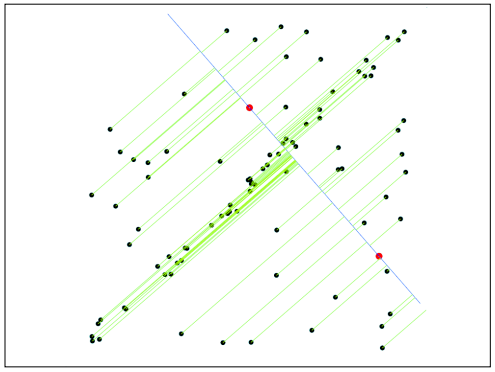
RANSAC for line fitting example



1. Randomly select minimal subset of points
2. Hypothesize a model

Source: R. Raguram

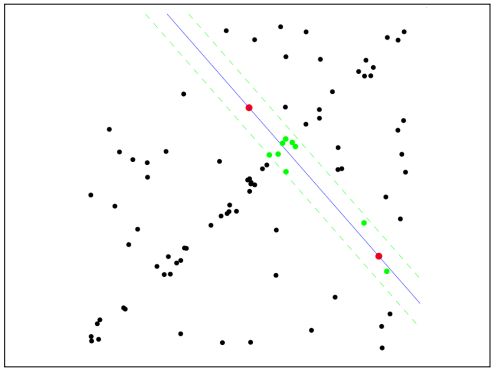
RANSAC for line fitting example



1. Randomly select minimal subset of points
2. Hypothesize a model
3. Compute error function

Source: R. Raguram

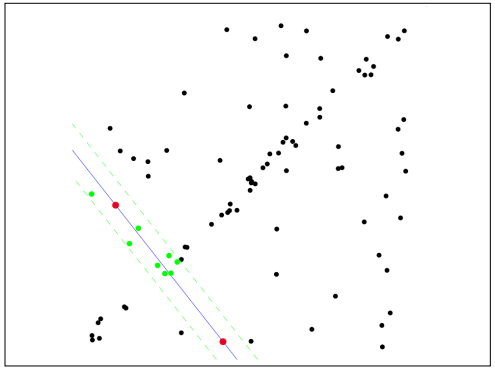
RANSAC for line fitting example



1. Randomly select minimal subset of points
2. Hypothesize a model
3. Compute error function
4. Select points consistent with model

Source: R. Raguram

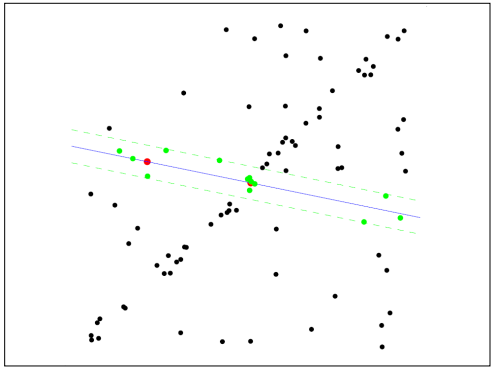
RANSAC for line fitting example



1. Randomly select minimal subset of points
2. Hypothesize a model
3. Compute error function
4. Select points consistent with model
5. Repeat *hypothesize-and-verify* loop

Source: R. Raguram

RANSAC for line fitting example

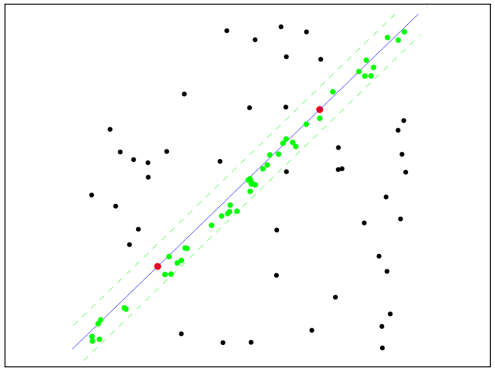


1. Randomly select minimal subset of points
2. Hypothesize a model
3. Compute error function
4. Select points consistent with model
5. Repeat *hypothesize-and-verify* loop

Source: R. Raguram

RANSAC for line fitting example

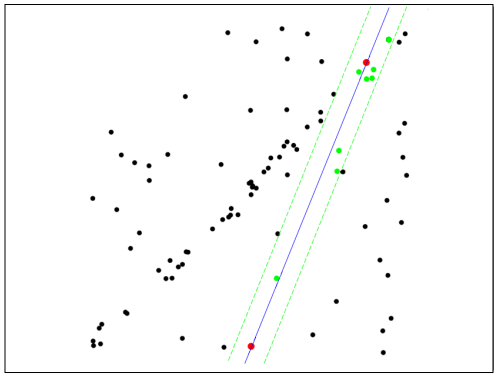
Uncontaminated sample



1. Randomly select minimal subset of points
2. Hypothesize a model
3. Compute error function
4. Select points consistent with model
5. Repeat *hypothesize-and-verify* loop

Source: R. Raguram

RANSAC for line fitting example



1. Randomly select minimal subset of points
2. Hypothesize a model
3. Compute error function
4. Select points consistent with model
5. Repeat *hypothesize-and-verify* loop

Source: R. Raguram

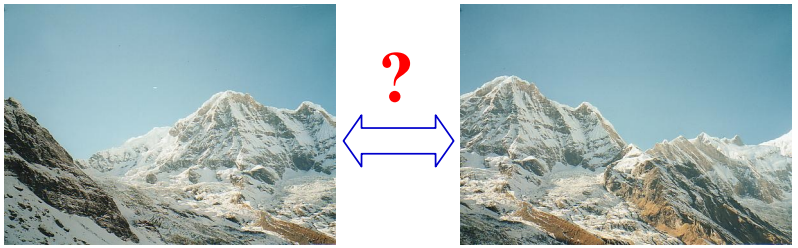
RANSAC for line fitting

Repeat ***N*** times:

- Draw ***s*** points uniformly at random
- Fit line to these ***s*** points
- Find ***inliers*** to this line among the remaining points (i.e., points whose distance from the line is less than ***t***)
- If there are ***d*** or more inliers and the number of inliers is higher than the *previous best*, accept the line and refit using all inliers.

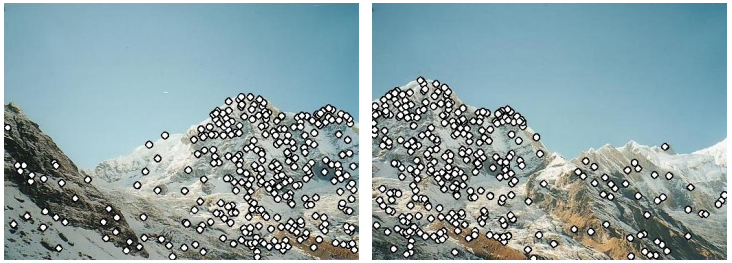
Panoramic stitching

- **Approach**
  - Local feature matching
  - RANSAC for alignment



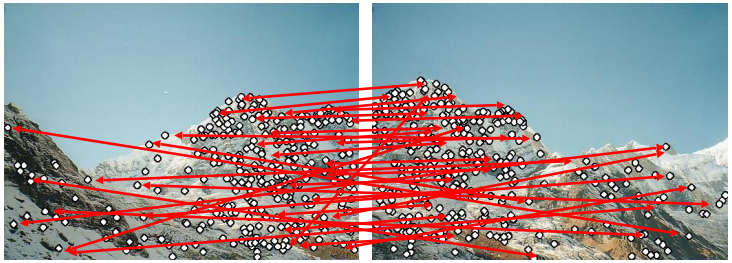
Panoramic stitching

Panoramic stitching



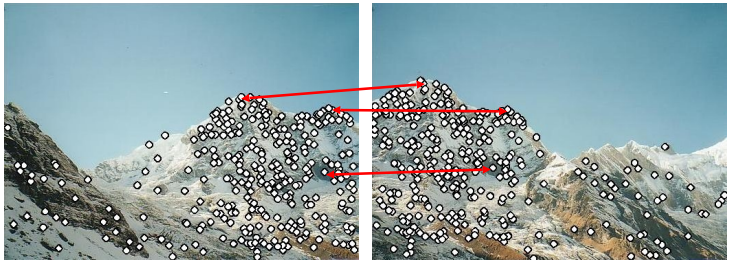
- Extract features
  - corner detector

Panoramic stitching



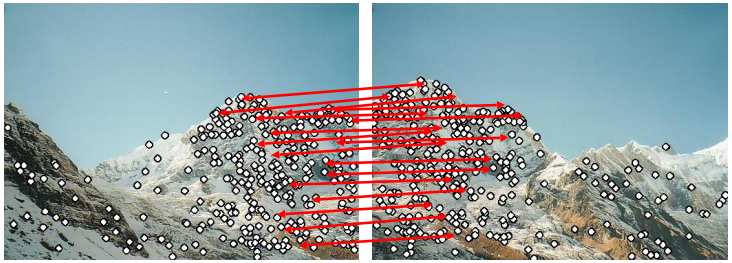
- Extract features
- Compute *putative matches*

Panoramic stitching



- Extract features
- Compute *putative matches*
- Loop:
  - ***Hypothesize*** transformation ***T***

Panoramic stitching



- Extract features
- Compute *putative matches*
- Loop:
  - *Hypothesize* transformation *T*
  - *Verify* transformation (search for other matches consistent with *T*)

Mechanics of transformation

procedure *forwardWarp*$(f, \boldsymbol{h}, \textbf{out}\ g)$:

For every pixel $\boldsymbol{x}$ in $f(\boldsymbol{x})$

1. Compute the destination location $\boldsymbol{x}' = \boldsymbol{h}(\boldsymbol{x})$.
2. Copy the pixel $f(\boldsymbol{x})$ to $g(\boldsymbol{x}')$.

From Computer Vision: Algorithms and Applications, by Rick Szeliski

Mechanics of transformation

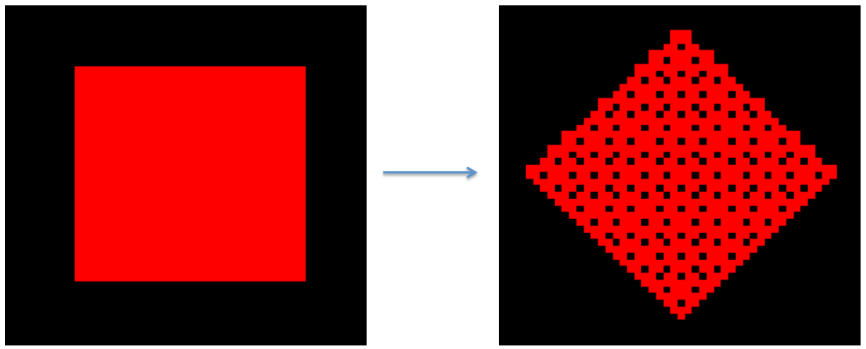
procedure *forwardWarp*$(f, \boldsymbol{h}, \textbf{out}\ g)$:

For every pixel $\boldsymbol{x}$ in $f(\boldsymbol{x})$

1. Compute the destination location $\boldsymbol{x}' = \boldsymbol{h}(\boldsymbol{x})$.
2. Copy the pixel $f(\boldsymbol{x})$ to $g(\boldsymbol{x}')$.

Example of forward warp

- Rotation by 45 degrees



Mechanics of transformation

procedure *inverseWarp*$(f, \boldsymbol{h}, \textbf{out}\ g)$:

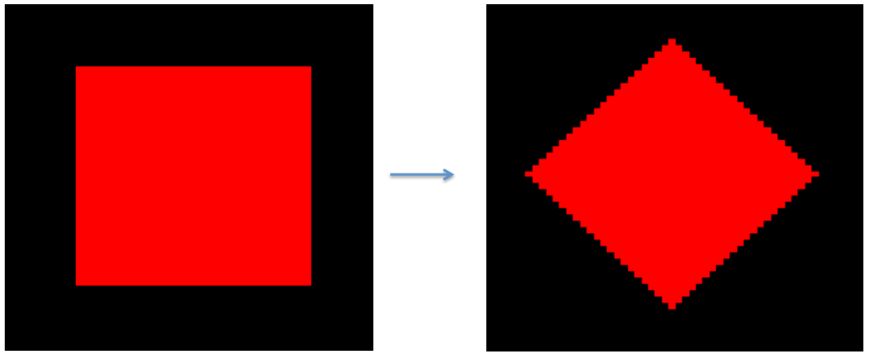
For every pixel $\boldsymbol{x}'$ in $g(\boldsymbol{x}')$

1. Compute the source location $\boldsymbol{x} = \hat{\boldsymbol{h}}(\boldsymbol{x}')$
2. Resample $f(\boldsymbol{x})$ at location $\boldsymbol{x}$ and copy to $g(\boldsymbol{x}')$

From Computer Vision: Algorithms and Applications, by Rick Szeliski

Example of inverse warp

- Rotation by 45 degrees



Panoramic stitching warping