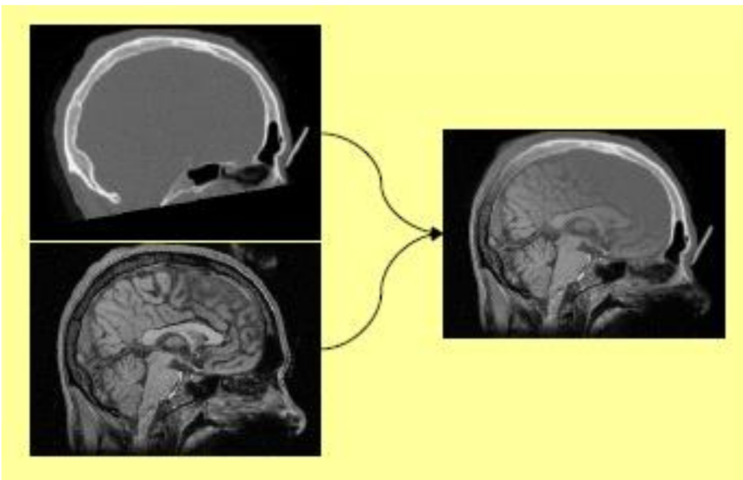
# Image Alignment

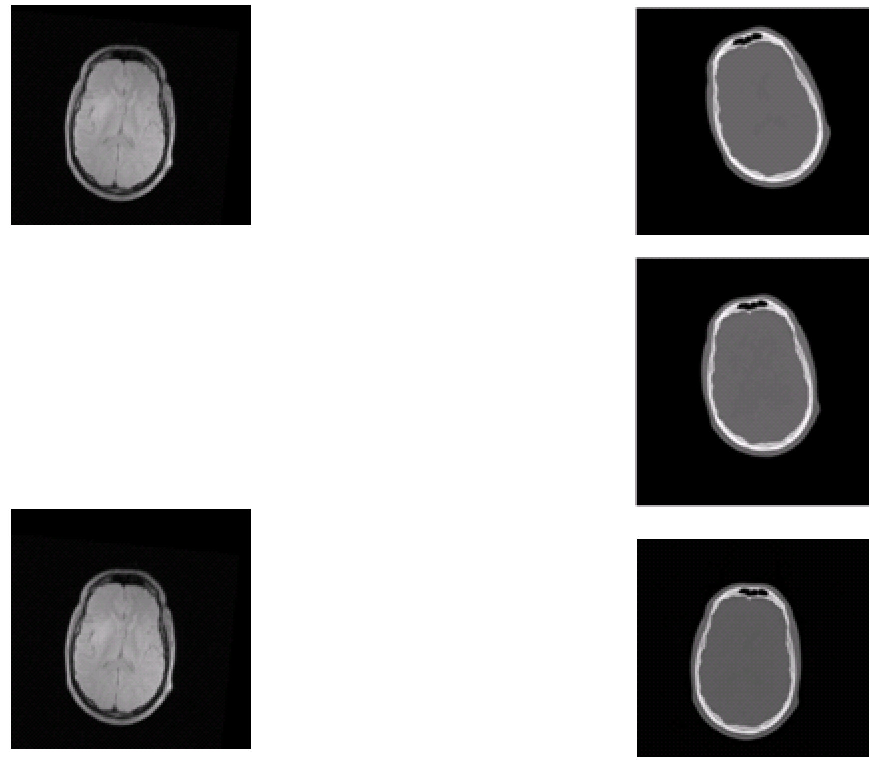
CS 413 Special Topics in Computer Science
Introduction to Computer Vision
Spring 2014
Mohamed E. Hussein

# Slide Sources

- Kristen Grauman's slides, Univ. of Texas at Austin

# Image Alignment: Medical Image Registration

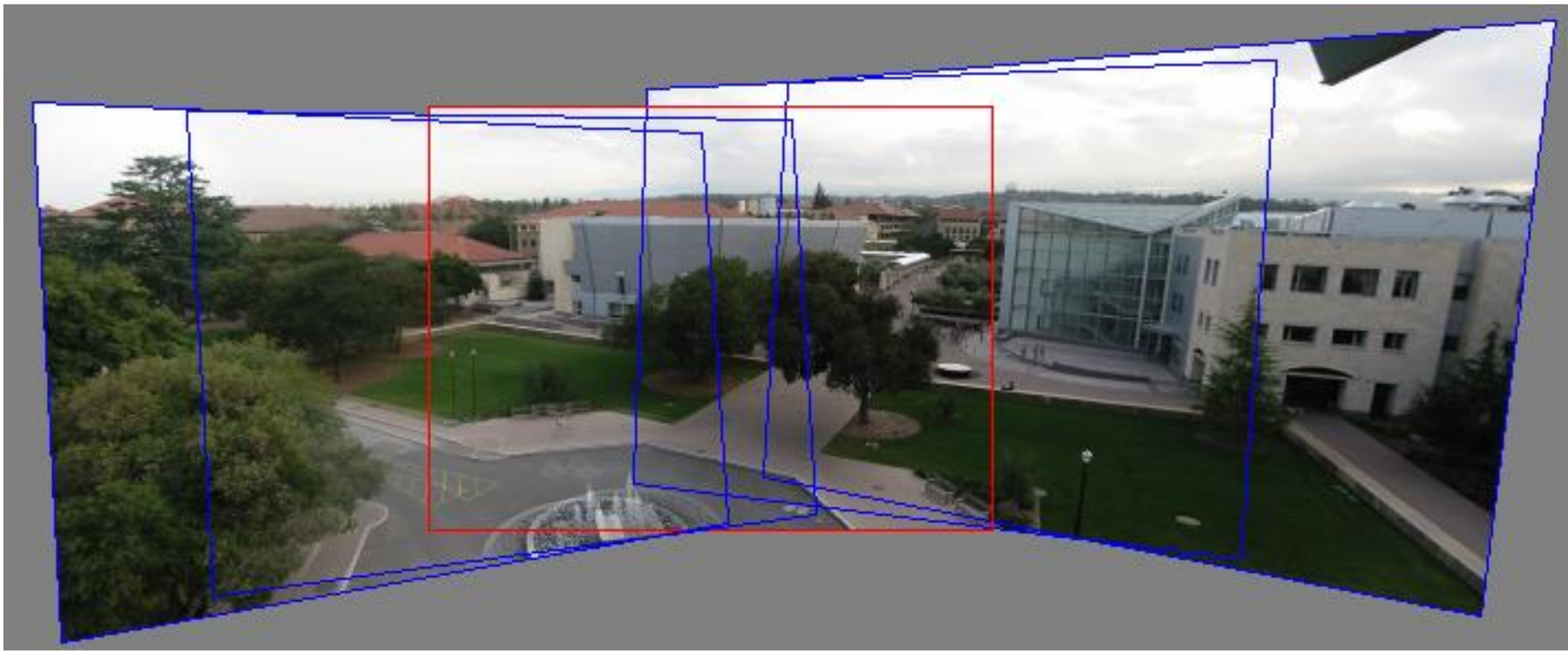




Alignment and Registration are used interchangeably to mean the same thing: mapping one image coordinate system to to another so that common pixels coincide.

*Kristen Grauman, UT-Austin*

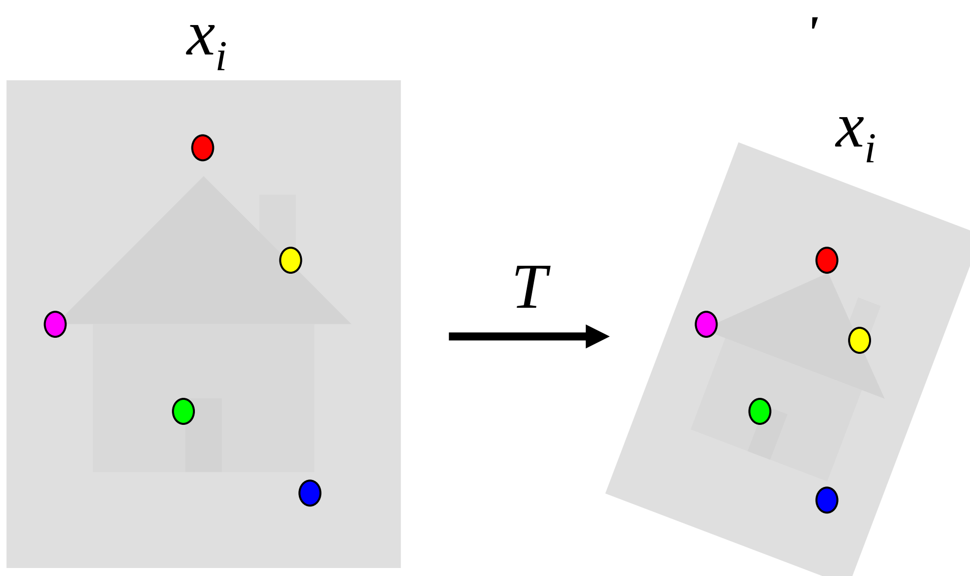
# Image Alignment: Mosaics



Kristen Grauman, UT-Austin

# Alignment problem

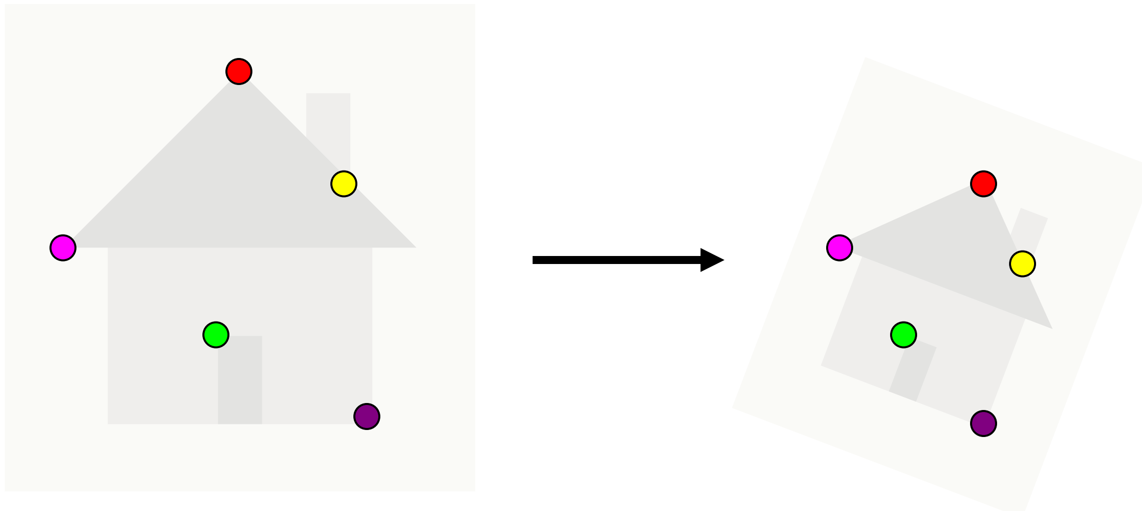
- In alignment, we will fit the parameters of some **transformation** according to a set of matching feature pairs ("correspondences").




*Kristen Grauman, UT-Austin*

Kristen Grauman

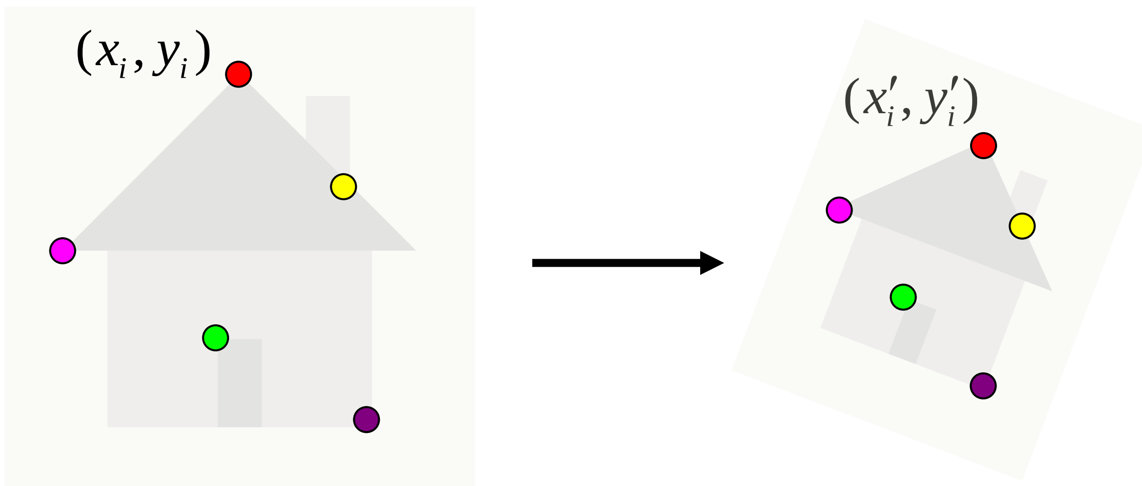
# Image alignment



- 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

# Fitting an affine transformation

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

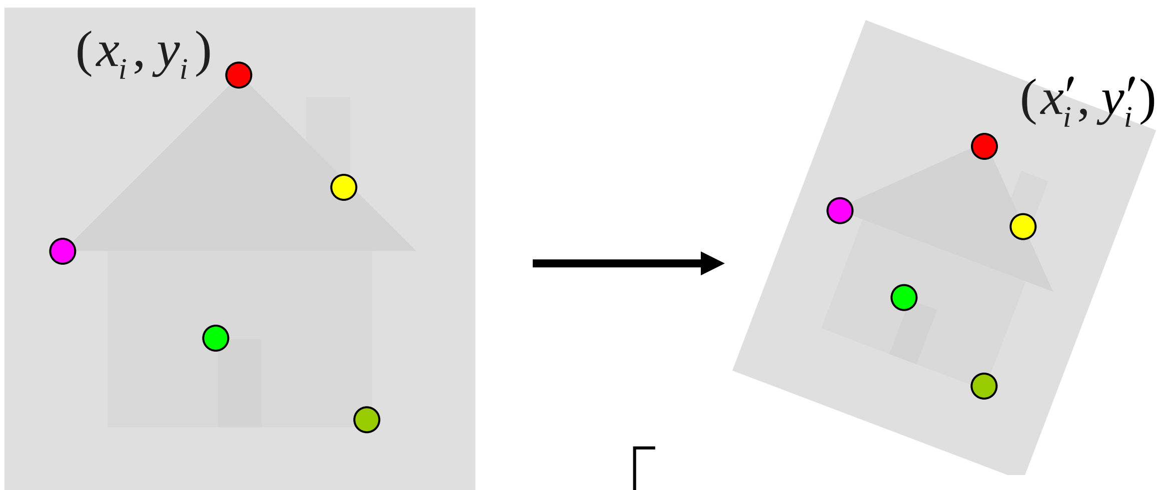



$$\begin{bmatrix} x_i' \\ y_i' \end{bmatrix} = \begin{bmatrix} m_1 & m_2 \\ m_3 & m_4 \end{bmatrix} \begin{bmatrix} x_i \\ y_i \end{bmatrix} + \begin{bmatrix} t_1 \\ t_2 \end{bmatrix}$$

*Kristen Grauman, UT-Austin*

# Fitting an affine transformation

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




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

$$\begin{bmatrix} \quad & \quad \\ & \end{bmatrix} \begin{bmatrix} m_1 \\ m_2 \\ m_3 \\ m_4 \\ t_1 \\ t_2 \end{bmatrix} = \begin{bmatrix} \quad \\ \quad \end{bmatrix}$$

Kristen Grauman, UT-Austin

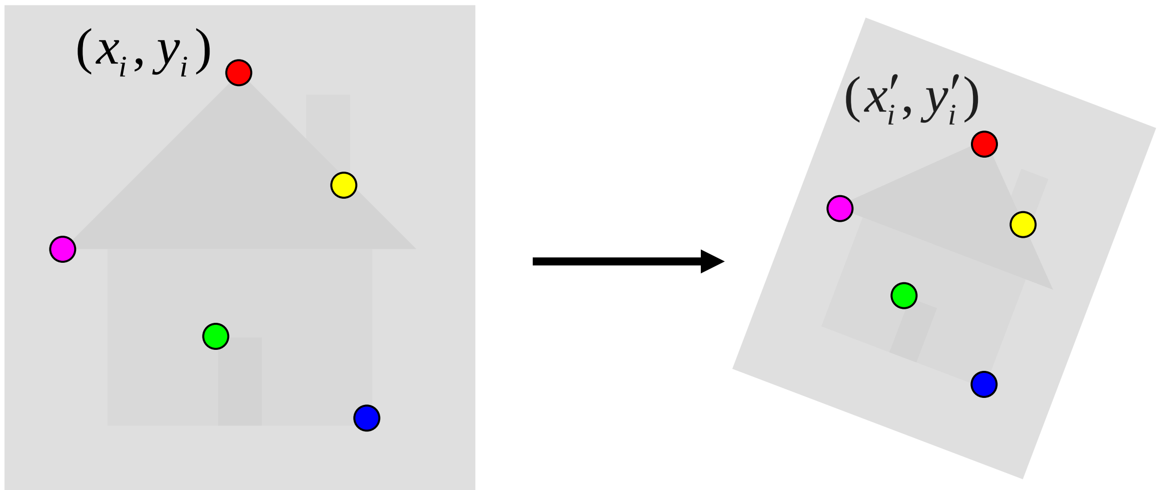
# Fitting an affine transformation

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

- 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 $(x_{new}, y_{new})$?
- Where do the matches come from?
  - Matching local features

# Fitting an affine transformation

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




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

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

# Fitting using homogeneous coordinates

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} a & b & c \\ d & e & f \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

- The set of equations becomes

$$\begin{bmatrix} & & \cdots & & & \\ x_i & y_i & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & x_i & y_i & 1 \\ & & \cdots & & & \end{bmatrix} \begin{bmatrix} a \\ b \\ c \\ d \\ e \\ f \end{bmatrix} = \begin{bmatrix} \cdots \\ x_i' \\ y_i' \\ \cdots \end{bmatrix}$$

# Recall: 2D Projective Transformations

A 2D projective transformation is also known as ***Homography***.

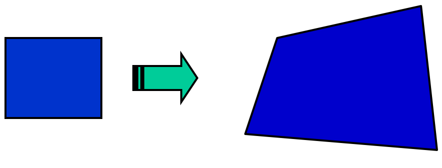
$$\begin{bmatrix} x' \\ y' \\ w' \end{bmatrix} = \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix} \begin{bmatrix} x \\ y \\ w \end{bmatrix}$$

Note that, generally, $w' \neq w$ in a homography.

Projective transformations are combinations of:

- Affine transformations, and
- Projective warps

Parallel lines do not necessarily remain parallel



# Recall: 2D Projective Transformations

- Since multiplying by a scalar does not change the point in homogeneous coordinates, a homography matrix can be arbitrarily scaled without changing the transformation.
- The typical 'canonical' form of a homography matrix is obtained by scaling the matrix such that the bottom-right most element, $i$, becomes 1.

$$\begin{bmatrix} x' \\ y' \\ w' \end{bmatrix} = \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ w \end{bmatrix}$$

Here, the homography matrix is put in the 'canonical' form.

# Fitting 2D Projective Transformations

To be able to use pixel coordinates of the two images directly, the equation is reformulated as:

$$w'\begin{bmatrix} x^* \\ y^* \\ 1 \end{bmatrix} = \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & 1 \end{bmatrix}\begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

, where $x^* = \frac{x'}{w'}, y^* = \frac{y'}{w'}$.

- The set of equations becomes

$$\begin{bmatrix} x & y & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & x & y & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & x & y \end{bmatrix}\begin{bmatrix} a \\ b \\ c \\ d \\ e \\ f \\ g \\ h \end{bmatrix} = \begin{bmatrix} w'x^* \\ w'y^* \\ w' - 1 \end{bmatrix}$$

- Each matching pair, $p_i$ and $p_i'$, gives three equations, but, adds a new unknown, $w_i'$.

# Fitting 2D Projective Transformations

- Note that the $w_i'$ unknowns are in the rhs, so equations need to be re-written to get all unknowns to the lhs

$$\begin{bmatrix} x & y & 1 & 0 & 0 & 0 & 0 & 0 & -x^* \\ 0 & 0 & 0 & x & y & 1 & 0 & 0 & -y^* \\ 0 & 0 & 0 & 0 & 0 & 0 & x & y & -1 \end{bmatrix} \begin{bmatrix} a \\ b \\ c \\ d \\ e \\ f \\ g \\ h \\ w' \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ -1 \end{bmatrix}$$

- It is much easier to get rid of the $w'$ variables all together.

# Fitting 2D Projective Transformations

- For two vectors $\vec{a}$ and $\vec{b}$, if $\vec{a} = \vec{b}$, then $\vec{a} \times \vec{b} = \underline{0}$, where $\underline{0}$ is a vector of all zeros.

$$\vec{a} \times \vec{b} = [\vec{a}]_\times \vec{b} = \begin{bmatrix} 0 & -a_3 & a_2 \\ a_3 & 0 & -a_1 \\ -a_2 & a_1 & 0 \end{bmatrix} \vec{b}$$

- Back to the homography equation: let $p^*$ and $p$ be a pair of matching points, and let $H$ be the homography matrix, then

$$w' p^* = Hp \Longrightarrow w' p^* \times Hp = \underline{0} \Longrightarrow p^* \times Hp = \underline{0}$$

- Now, we have successfully removed the $w'$ variable
- We can get only two equation per pair of points
  - Why?
  - How many pair of points to we need to estimate?
  - Construct the resulting system of equations

# An aside: Least Squares Example

Say we have a set of data points (X1,X1’), (X2,X2’), (X3,X3’), etc. (e.g. person’s height vs. weight)

We want a nice compact formula (a line) to predict X’s from Xs: aX + b = X’

We want to find a and b

How many (X,X’) pairs do we need?

Each point gives us one equation, so, we need two points to solve for two unknowns

$$aX_1 + b = X_1'$$
$$aX_2 + b = X_2'$$

$$\begin{bmatrix} X_1 & 1 \\ X_2 & 1 \end{bmatrix} \begin{bmatrix} a \\ b \end{bmatrix} = \begin{bmatrix} X_1' \\ X_2' \end{bmatrix}$$

$$Ax = b$$

# An aside: Least Squares Example

What if the data is noisy?

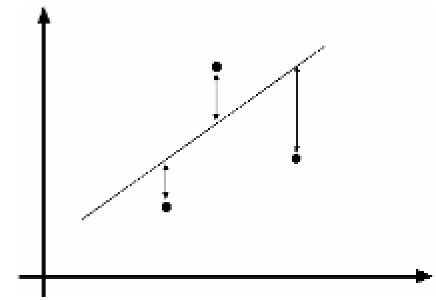
Use more points than needed and minimize fitting error

$$\begin{bmatrix} X_1 & 1 \\ X_2 & 1 \\ X_3 & 1 \\ \dots & \dots \end{bmatrix} \begin{bmatrix} a \\ b \end{bmatrix} = \begin{bmatrix} X_1' \\ X_2' \\ X_3' \\ \dots \end{bmatrix}$$

Overconstrained system of equations

Find $x$ that achieves

$$\min \|Ax - b\|^2$$



This is called the Least Squares solution

Recall that

$$Ax = b \Rightarrow A^T A x = A^T b \Rightarrow x = (A^T A)^{-1} A^T b$$

It is proven that this is the least squares solution for $Ax = b$.

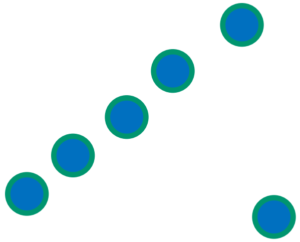


Source: Alyosha Efros

# Outliers

**Outliers** can hurt the quality of our parameter estimates, e.g.,

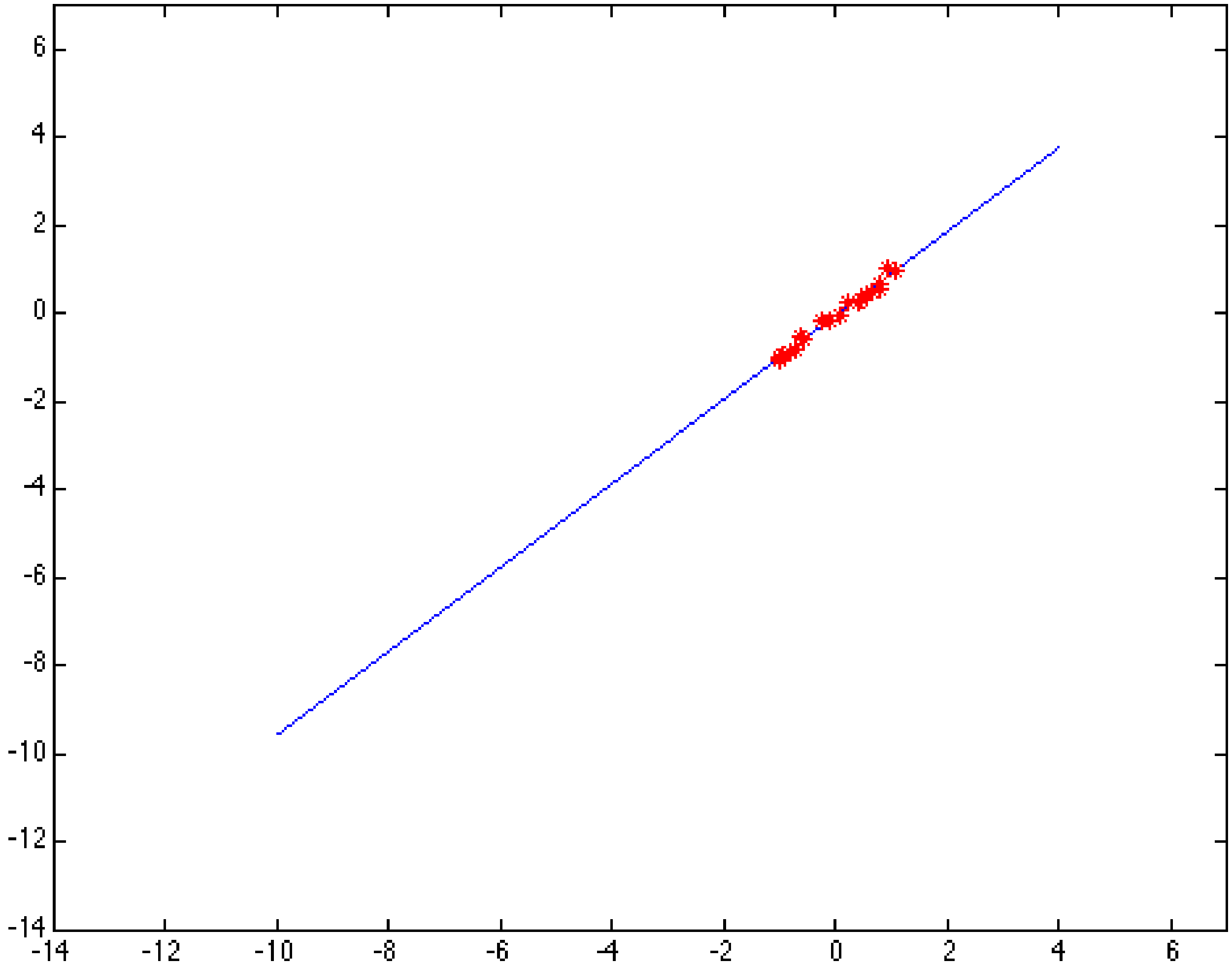
- an erroneous pair of matching points from two images
- an edge point that is noise, or doesn't belong to the line we are fitting.





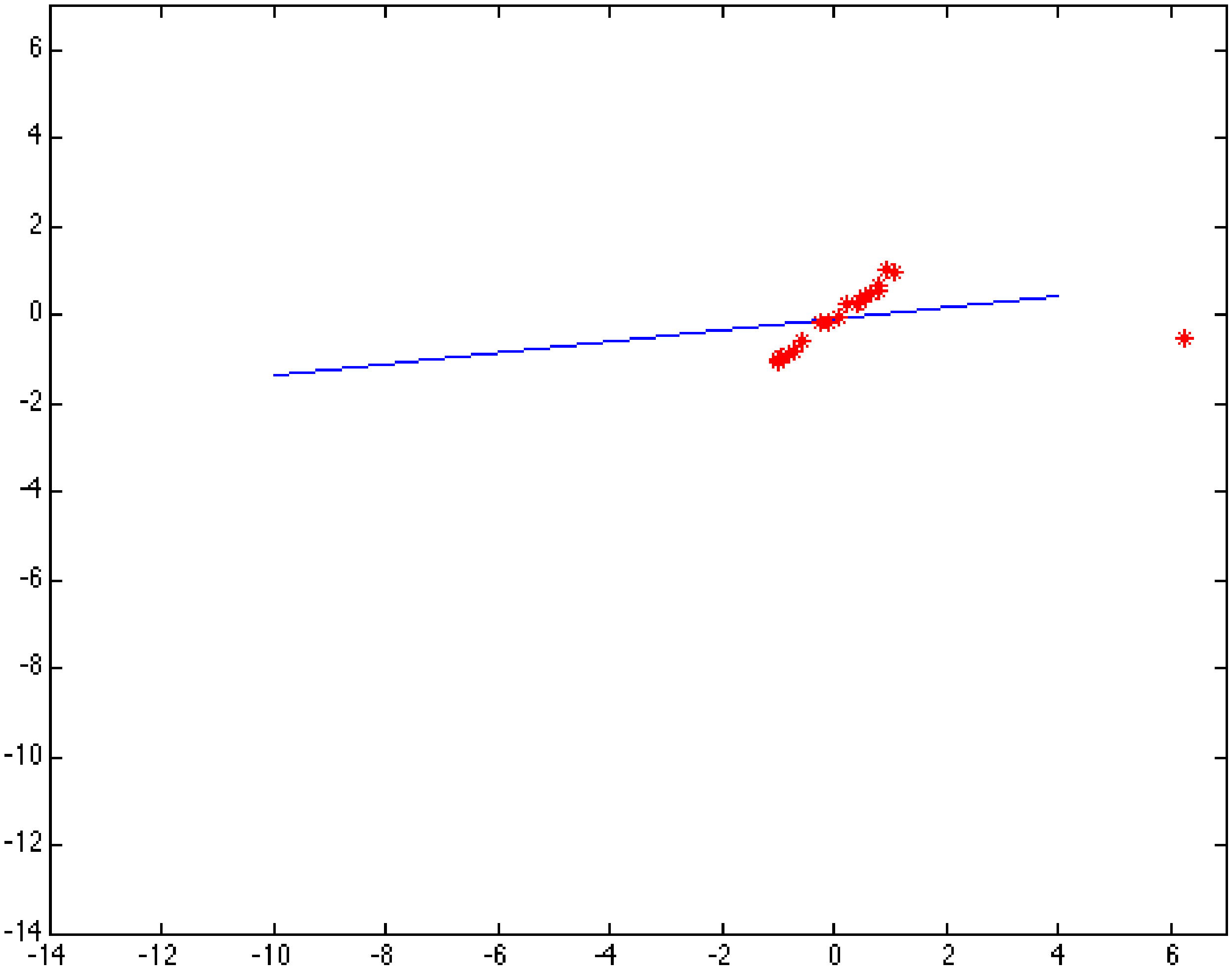
Kristen Grauman

# Outliers affect least squares fit




*Kristen Grauman, UT-Austin*

# 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.

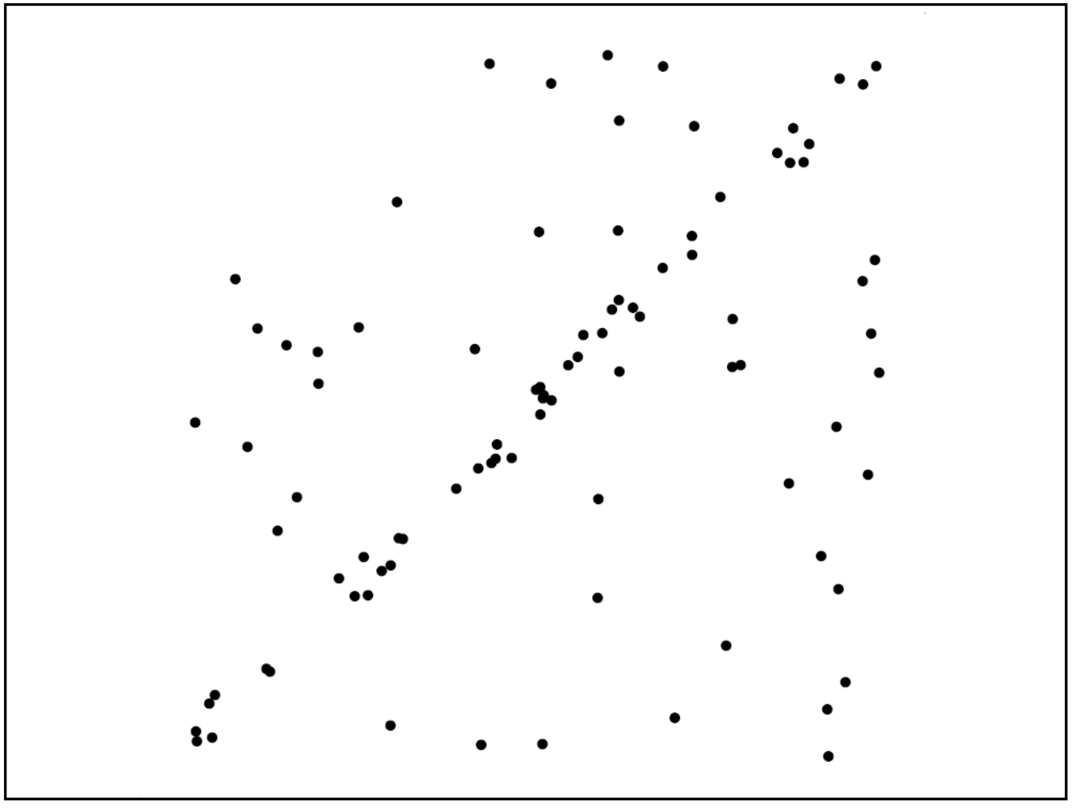
# RANSAC: General form

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 estimate of transformation on all of the inliers

Keep the transformation with the largest number of inliers

# RANSAC for line fitting example

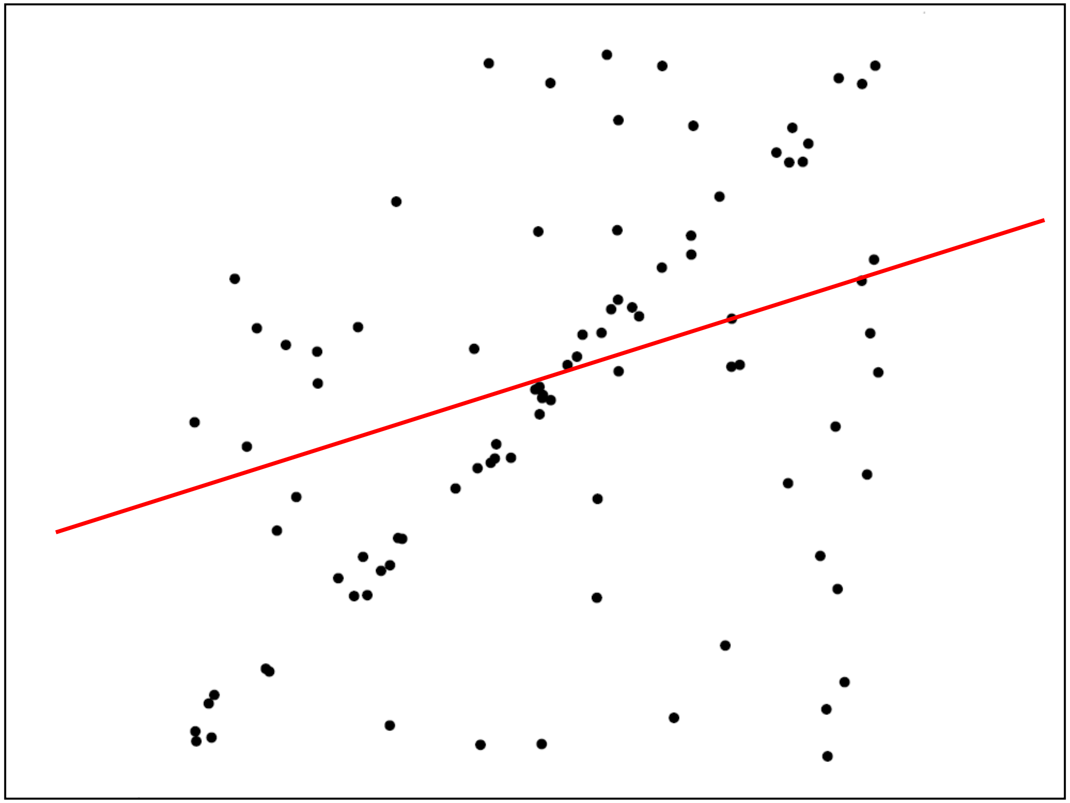


Source: R. Raguram

Lana Lazebnik

*Kristen Grauman, UT-Austin*

# RANSAC for line fitting example



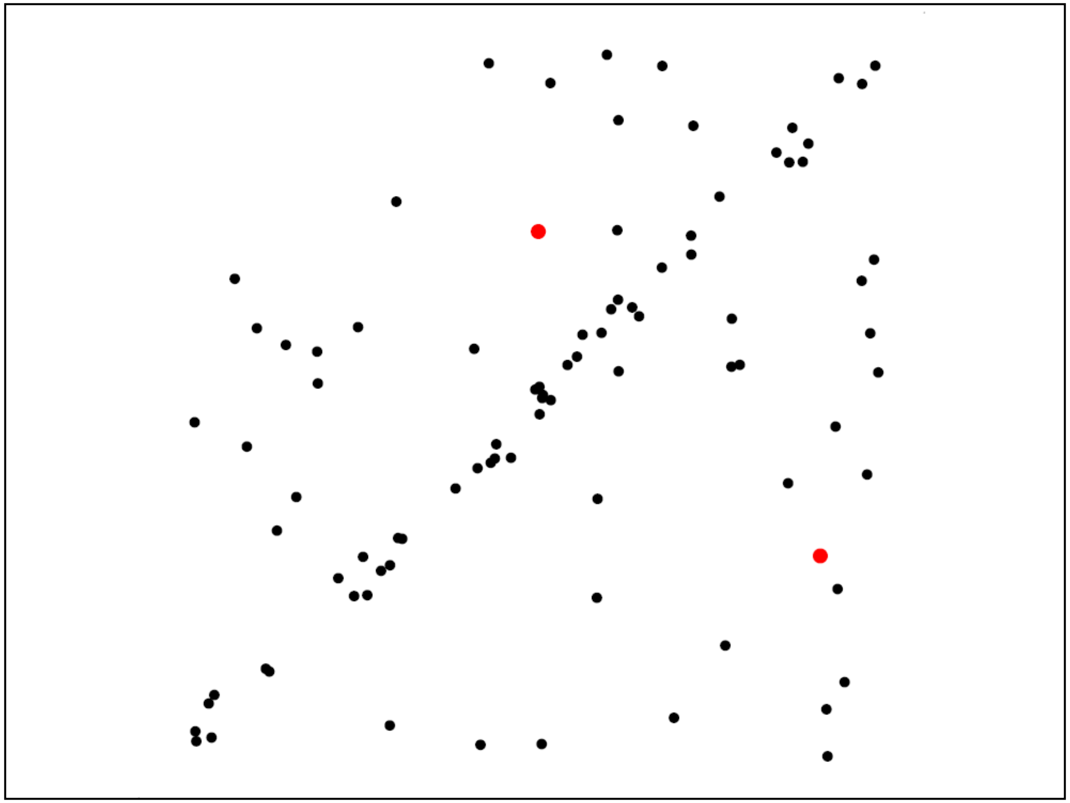
**Least-squares fit**

Source: R. Raguram

*Kristen Grauman, UT-Austin*

Lana Lazebnik

# RANSAC for line fitting example

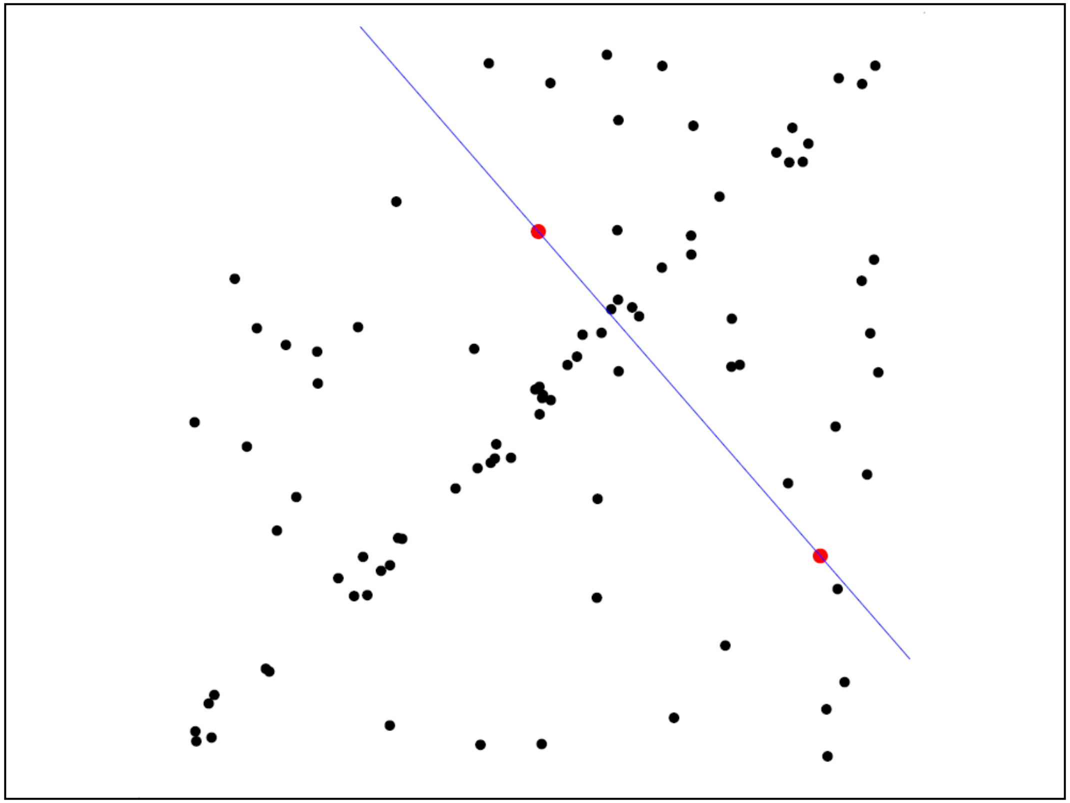


1. Randomly select minimal subset of points

Source: R. Raguram
*Kristen Grauman, UT-Austin*

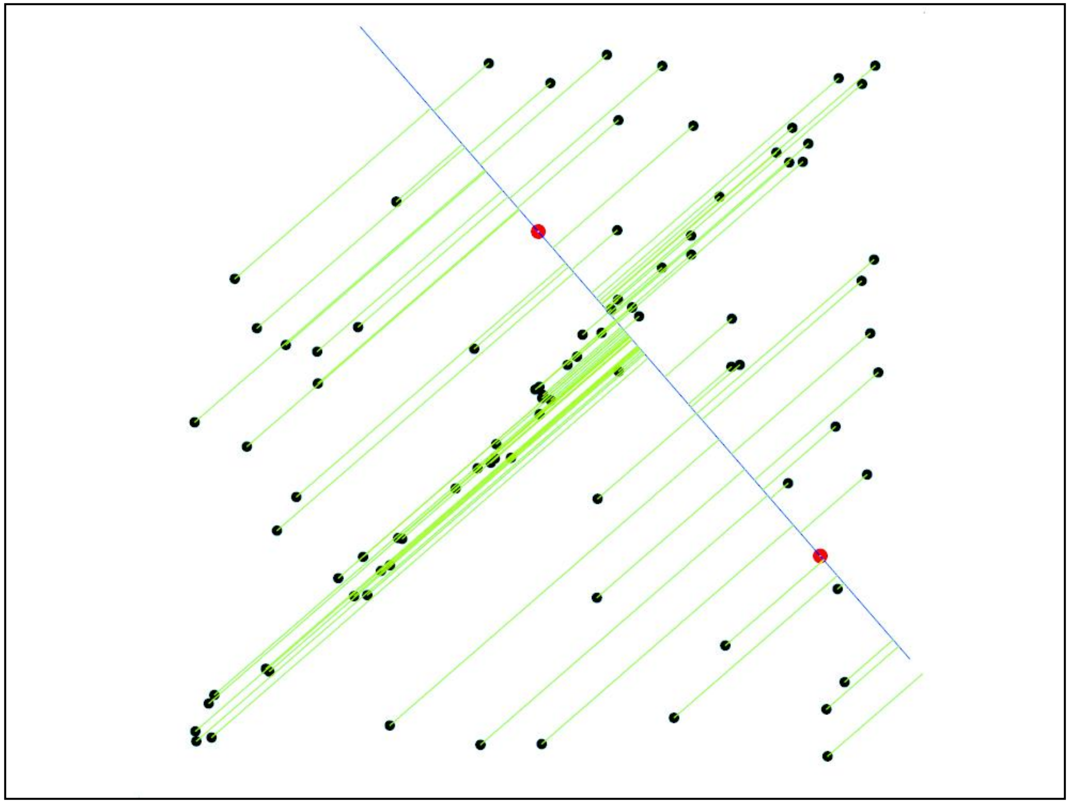
Lana Lazebnik

# RANSAC for line fitting example



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

# RANSAC for line fitting example



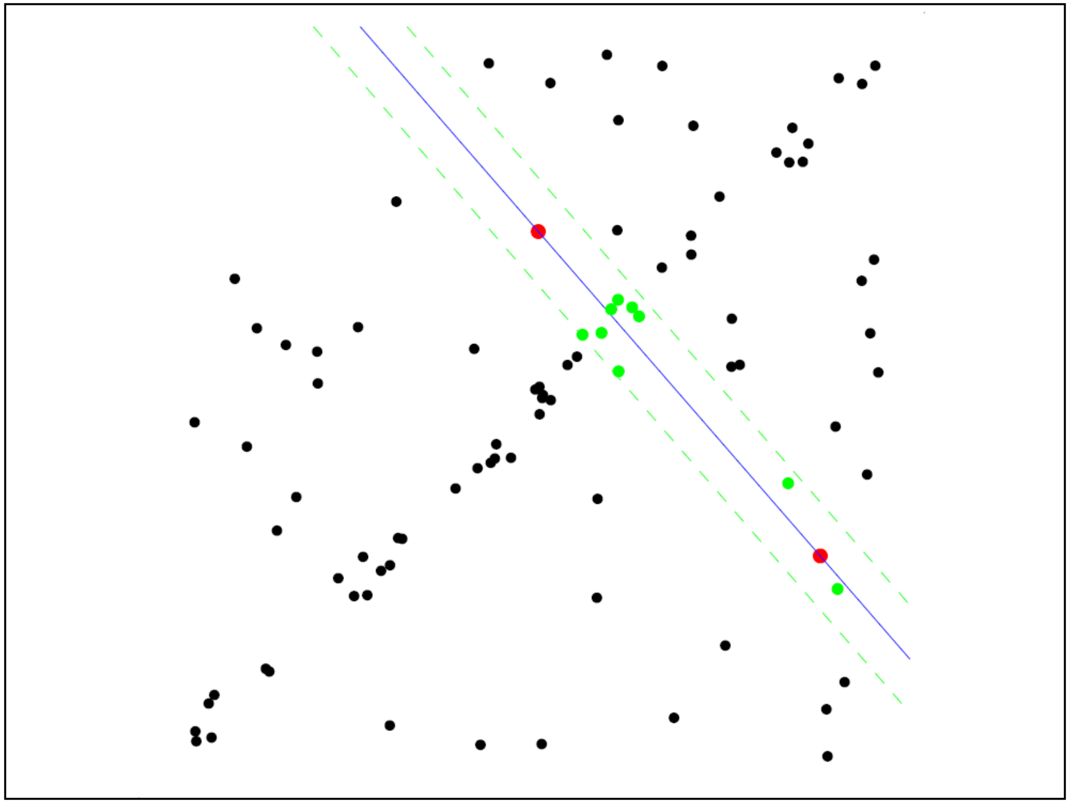
1. Randomly select minimal subset of points
2. Hypothesize a model
3. Compute error function

Source: R. Raguram

*Kristen Grauman, UT-Austin*

Lana Lazebnik

# RANSAC for line fitting example



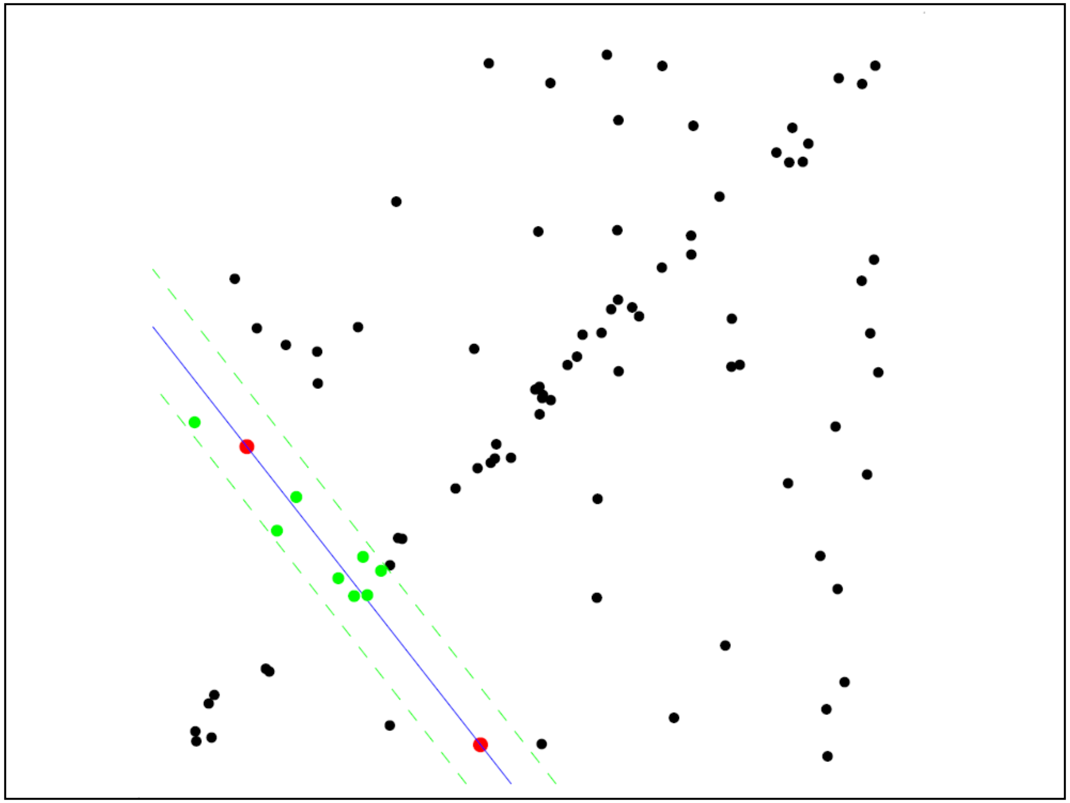
1. Randomly select minimal subset of points
2. Hypothesize a model
3. Compute error function
4. Select points consistent with model

Source: R. Raguram

*Kristen Grauman, UT-Austin*

Lana Lazebnik

# RANSAC for line fitting example

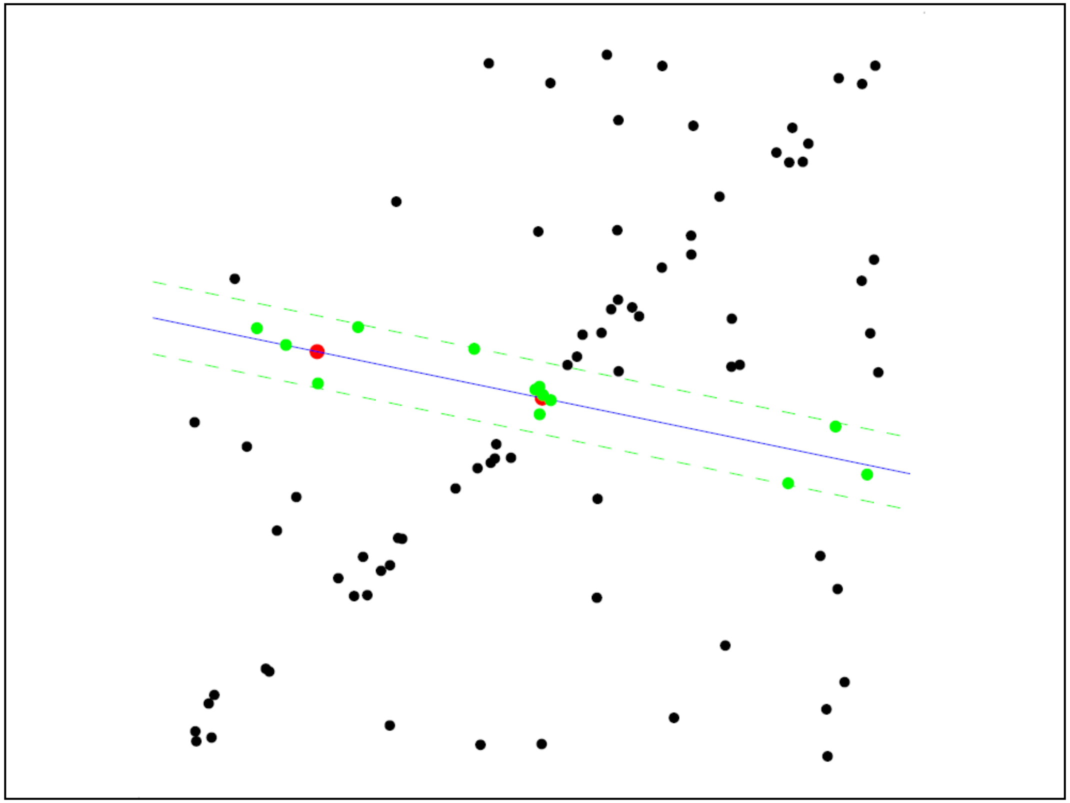


1. Randomly select minimal subset of points
2. Hypothesize a model
3. Compute error function
4. Select points consistent with model
5. Repeat *hypothesize-and-verify* loop

Source: R. Raguram
*Kristen Grauman, UT-Austin*

Lana Lazebnik

# RANSAC for line fitting example



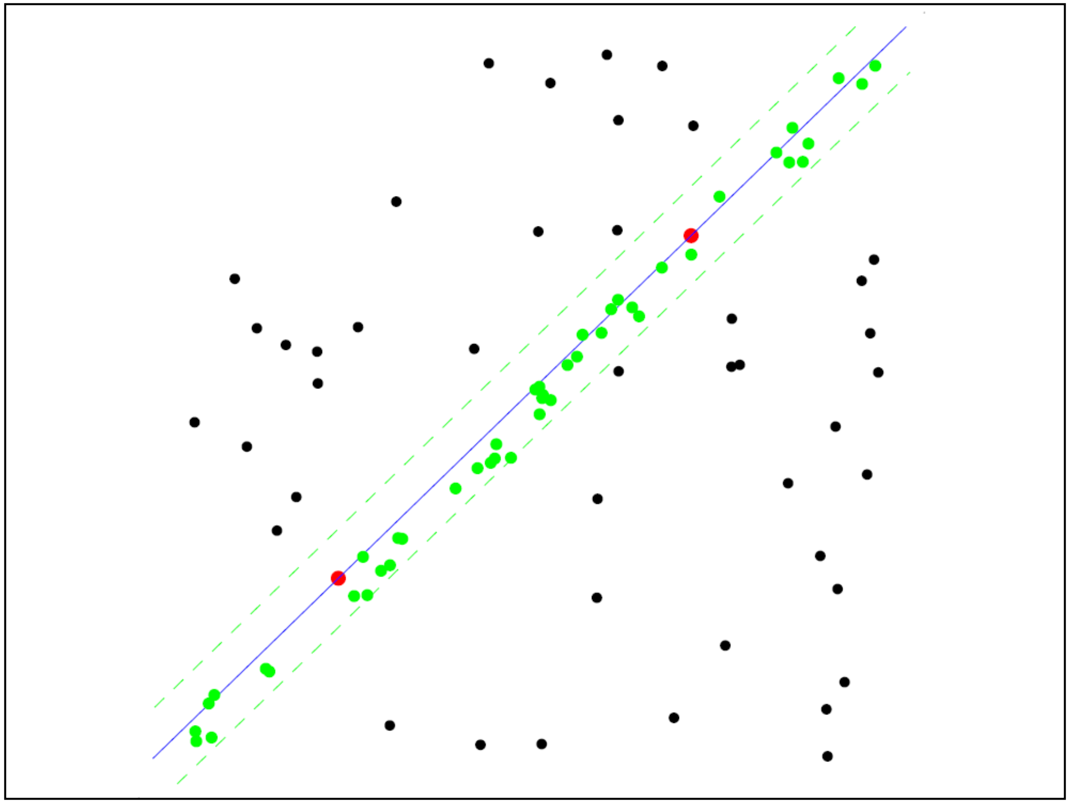
1. Randomly select minimal subset of points
2. Hypothesize a model
3. Compute error function
4. Select points consistent with model
5. Repeat *hypothesize-and-verify* loop

Source: R. Raguram

Lana Lazebnik

# RANSAC for line fitting example

**Uncontaminated sample**



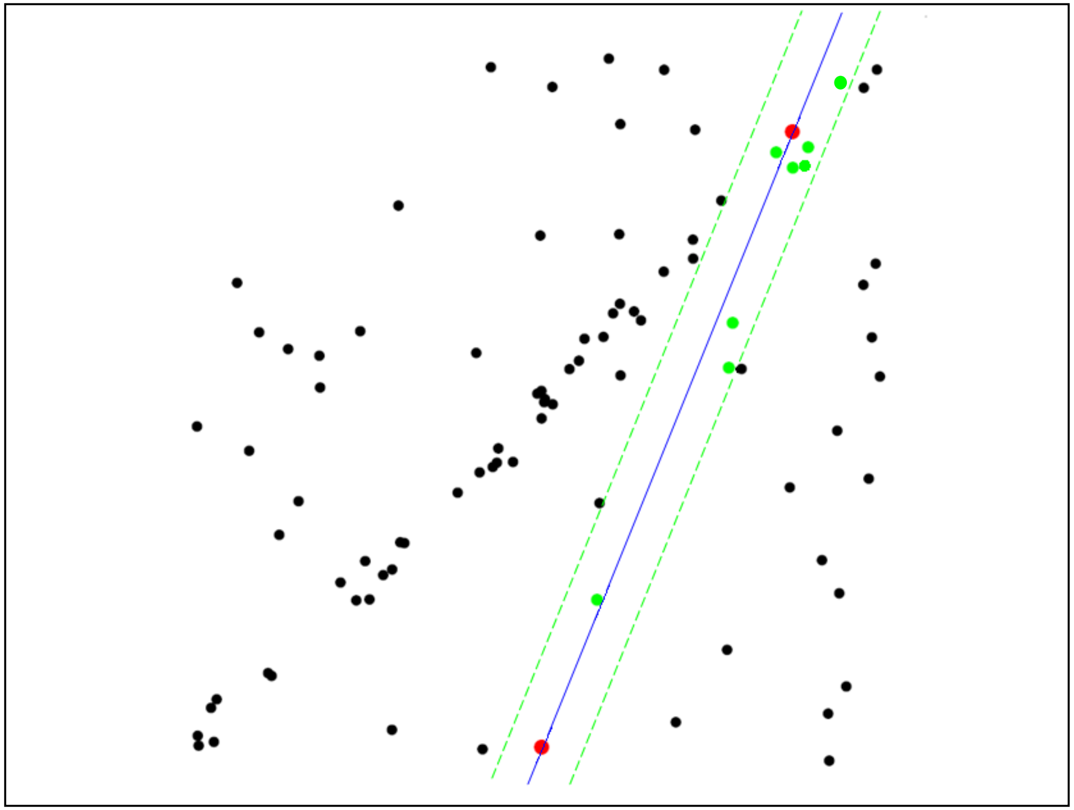
1. Randomly select minimal subset of points
2. Hypothesize a model
3. Compute error function
4. Select points consistent with model
5. Repeat *hypothesize-and-verify* loop

Source: R. Raguram

*Kristen Grauman, UT-Austin*

Lana Lazebnik

# 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
*Kristen Grauman, UT-Austin*

Lana Lazebnik

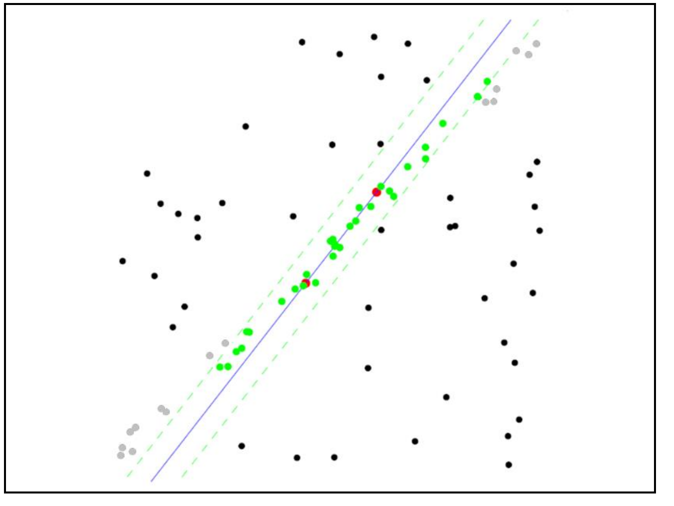
# RANSAC for line fitting

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, accept the line and refit using all inliers

# RANSAC pros and cons

- Pros
  - Simple and general
  - Applicable to many different problems
  - Often works well in practice
- Cons
  - Lots of parameters to tune
  - Doesn’t work well for low inlier ratios (too many iterations, or can fail completely)
  - Can’t always get a good initialization of the model based on the minimum number of samples
  - Hard to work with multiple models
    - E.g. multiple lines in an image



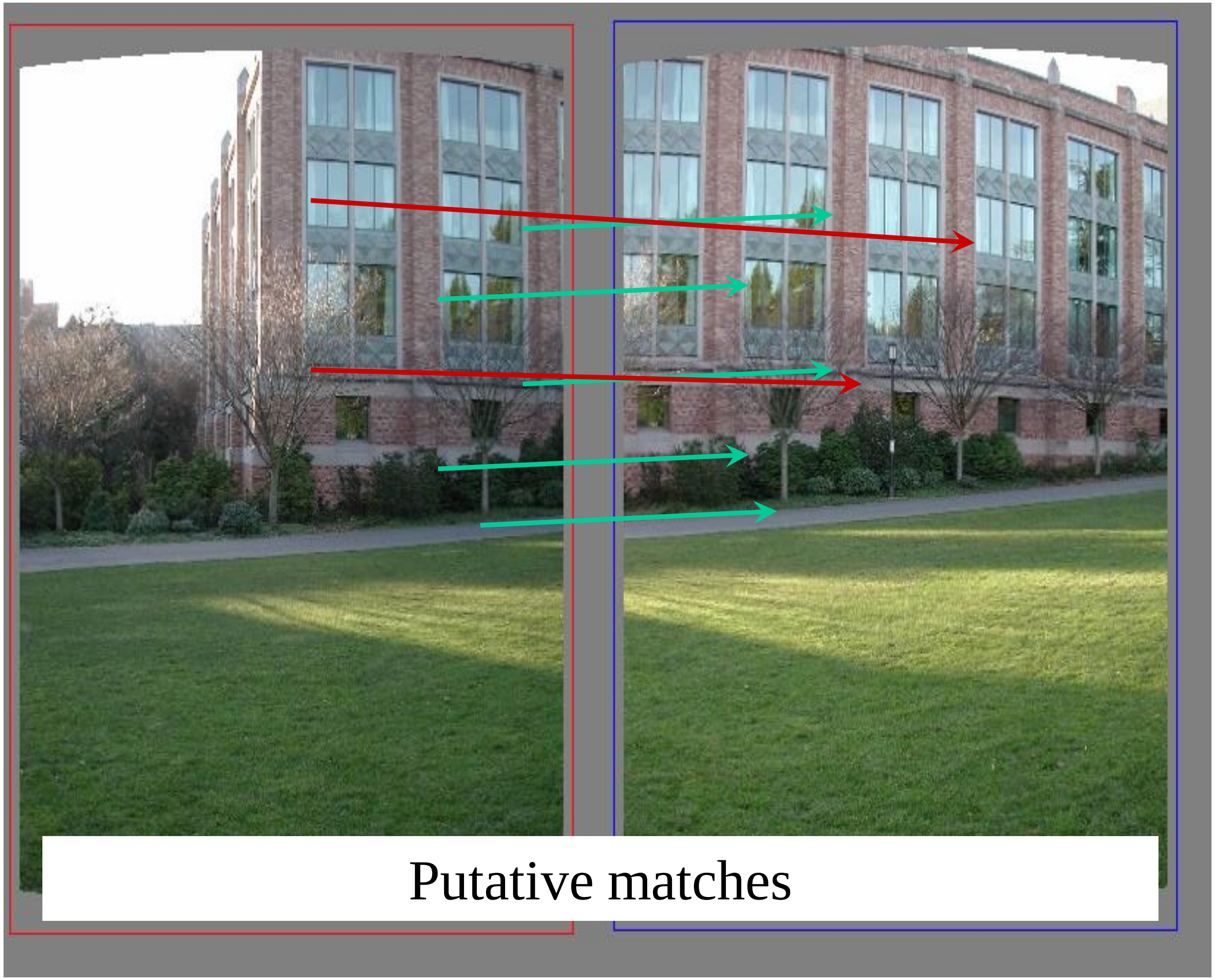
Lana Lazebnik

*Kristen Grauman, UT-Austin*

That is an example fitting a model (line)…

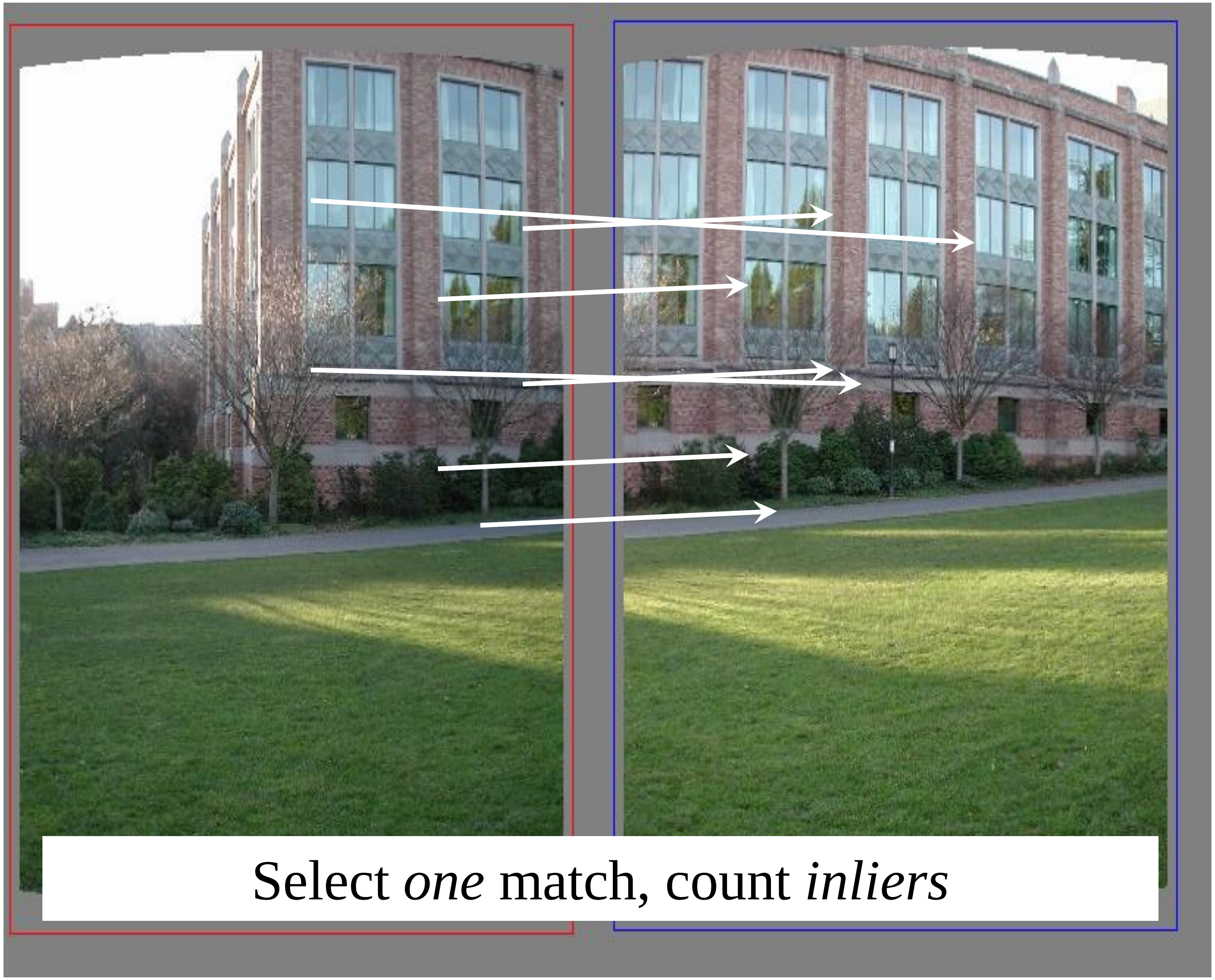
What about fitting a transformation (translation)?

# RANSAC example: Translation

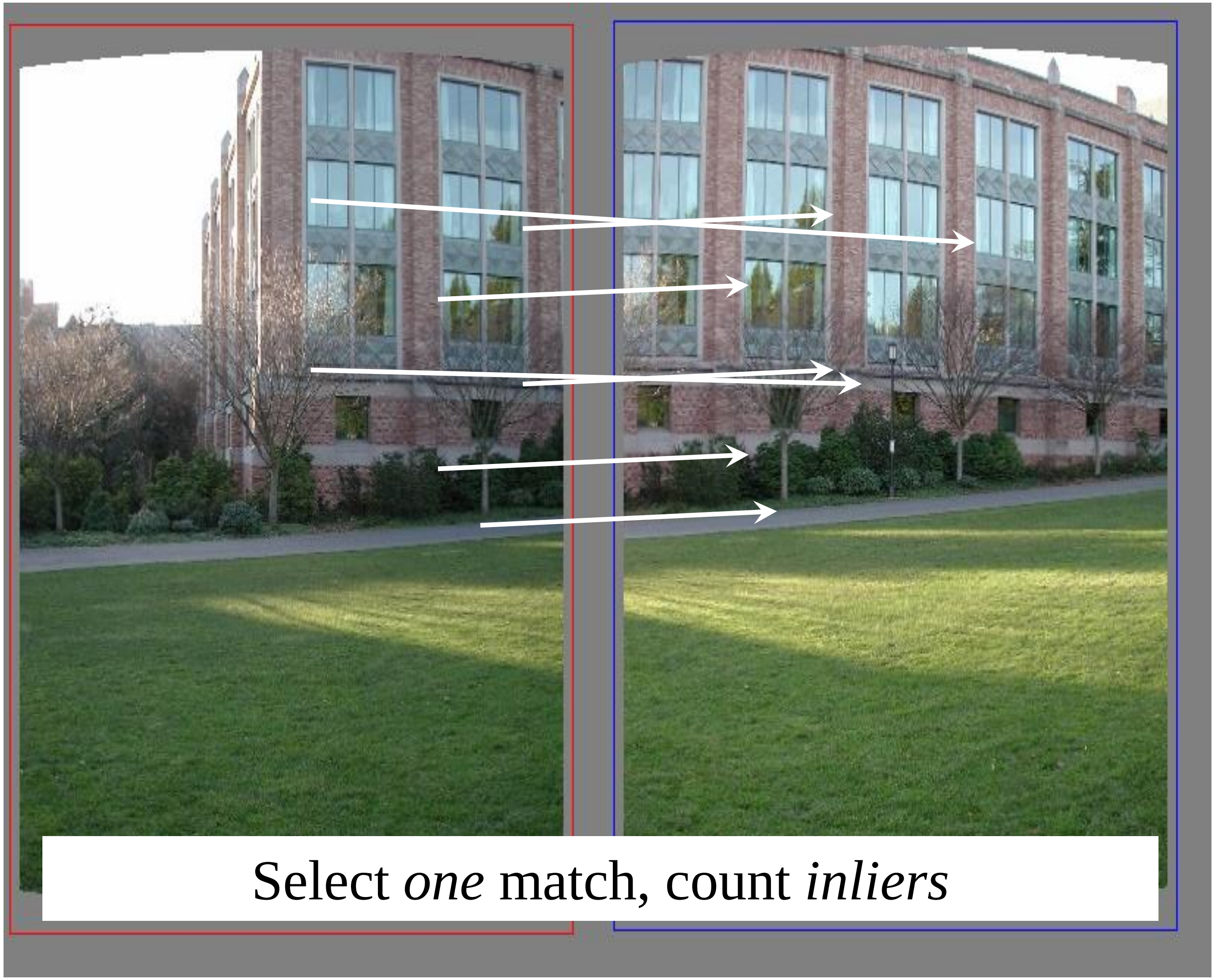



Source: Rick Szeliski

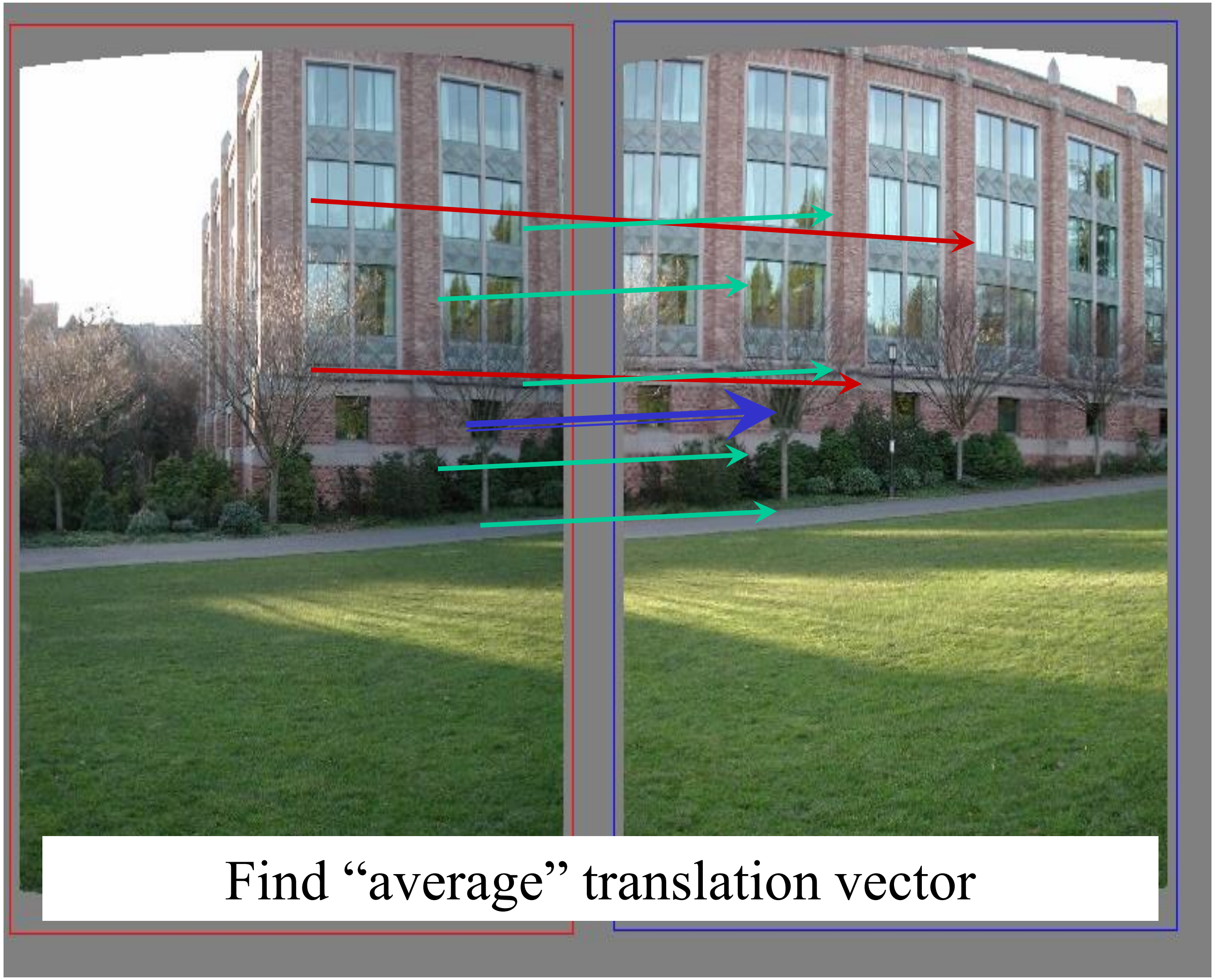
# RANSAC example: Translation



# RANSAC example: Translation



# RANSAC example: Translation



Find “average” translation vector

# Readings

- “Digital Image Processing”, Burger and Burge: 16.1.1-4
- “Computer Vision: Algorithms and Applications”, Richard Szeliski: 6.1.4