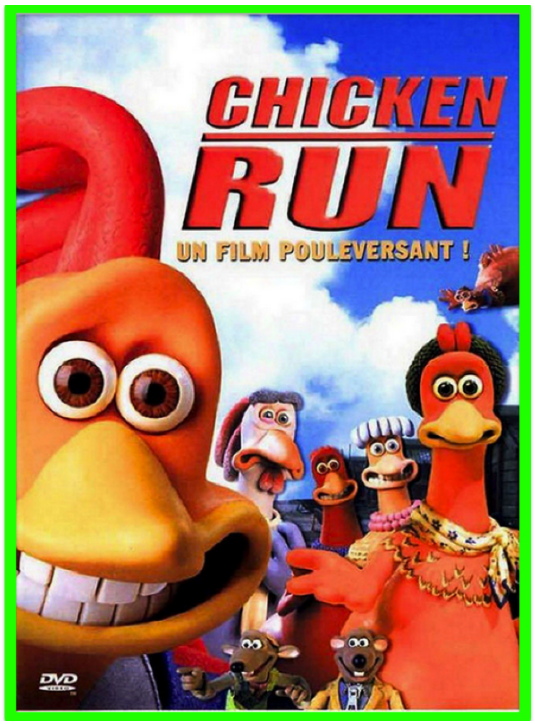
Matching Planar Objects In New Viewpoints

What Kind of Transformation Happened To My DVD?

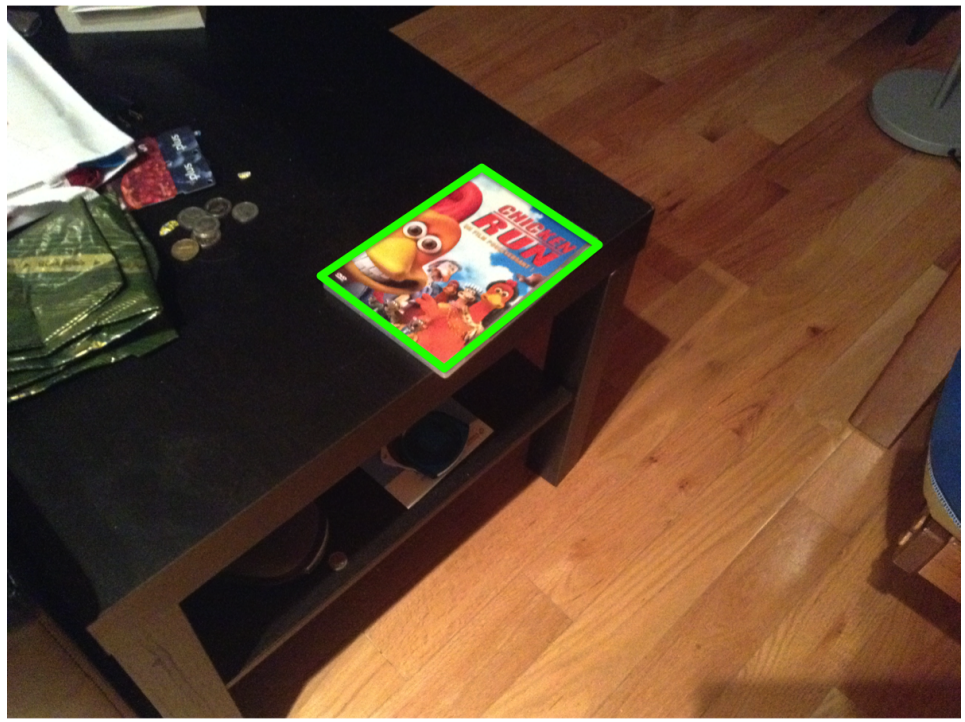
$T?$

What Kind of Transformation Happened To My DVD?

- Rectangle goes to a parallelogram (almost but not really, but let's believe that for now)



T?



All 2D Linear Transformations

Linear transformations are combinations of

- Scale,
- Rotation
- Shear
- Mirror

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

[Source: N. Snavely]

All 2D Linear Transformations

Properties of linear transformations:

- Origin maps to origin
- Lines map to lines

All 2D Linear Transformations

Properties of linear transformations:

- Origin maps to origin
- Lines map to lines
- Parallel lines remain parallel

All 2D Linear Transformations

Properties of linear transformations:

- Origin maps to origin
- Lines map to lines
- Parallel lines remain parallel
- Ratios are preserved

All 2D Linear Transformations

Properties of linear transformations:

- Origin maps to origin
- Lines map to lines
- Parallel lines remain parallel
- Ratios are preserved
- Closed under composition

All 2D Linear Transformations

Properties of linear transformations:

- Origin maps to origin
- Lines map to lines
- Parallel lines remain parallel
- Ratios are preserved
- Closed under composition

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

All 2D Linear Transformations

Properties of linear transformations:

- Origin maps to origin
- Lines map to lines
- Parallel lines remain parallel
- Ratios are preserved
- Closed under composition

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

What about the translation?

[Source: N. Snavely]

All 2D Linear Transformations

Properties of linear transformations:

- Origin maps to origin
- Lines map to lines
- Parallel lines remain parallel
- Ratios are preserved
- Closed under composition

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

What about the translation?

[Source: N. Snavely]

Affine Transformations

Affine transformations are combinations of

- Linear transformations, and
- Translations

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

same as:

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

Affine Transformations

Affine transformations are combinations of

- Linear transformations, and
- Translations

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

same as:

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

Affine Transformations

Affine transformations are combinations of

- Linear transformations, and
- Translations

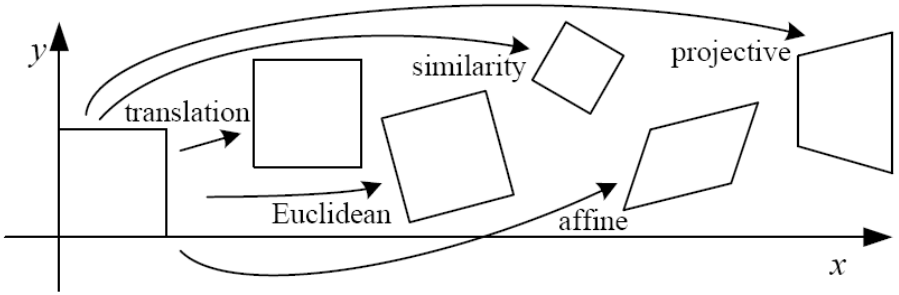
$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} a & b & e \\ c & d & f \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Properties of affine transformations:

- Origin does not necessarily map to origin
- Lines map to lines
- Parallel lines remain parallel
- Ratios are preserved
- Closed under composition
- Rectangles go to parallelograms

[Source: N. Snavely]

2D Image Tranformations

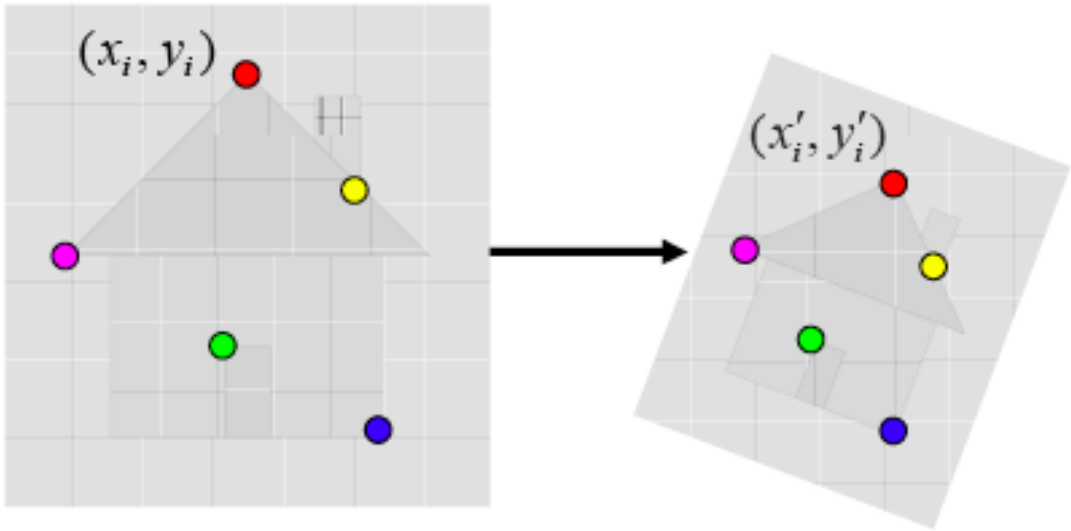


Transformation	Matrix	# DoF	Preserves	Icon
translation	$\begin{bmatrix} \boldsymbol{I} \mid \boldsymbol{t} \end{bmatrix}_{2\times 3}$	2	orientation	
rigid (Euclidean)	$\begin{bmatrix} \boldsymbol{R} \mid \boldsymbol{t} \end{bmatrix}_{2\times 3}$	3	lengths	
similarity	$\begin{bmatrix} s\boldsymbol{R} \mid \boldsymbol{t} \end{bmatrix}_{2\times 3}$	4	angles	
affine	$\begin{bmatrix} \boldsymbol{A} \end{bmatrix}_{2\times 3}$	6	parallelism	
projective	$\begin{bmatrix} \tilde{\boldsymbol{H}} \end{bmatrix}_{3\times 3}$	8	straight lines	

- These transformations are a nested set of groups
- Closed under composition and inverse is a member

What Transformation Happened to My DVD?

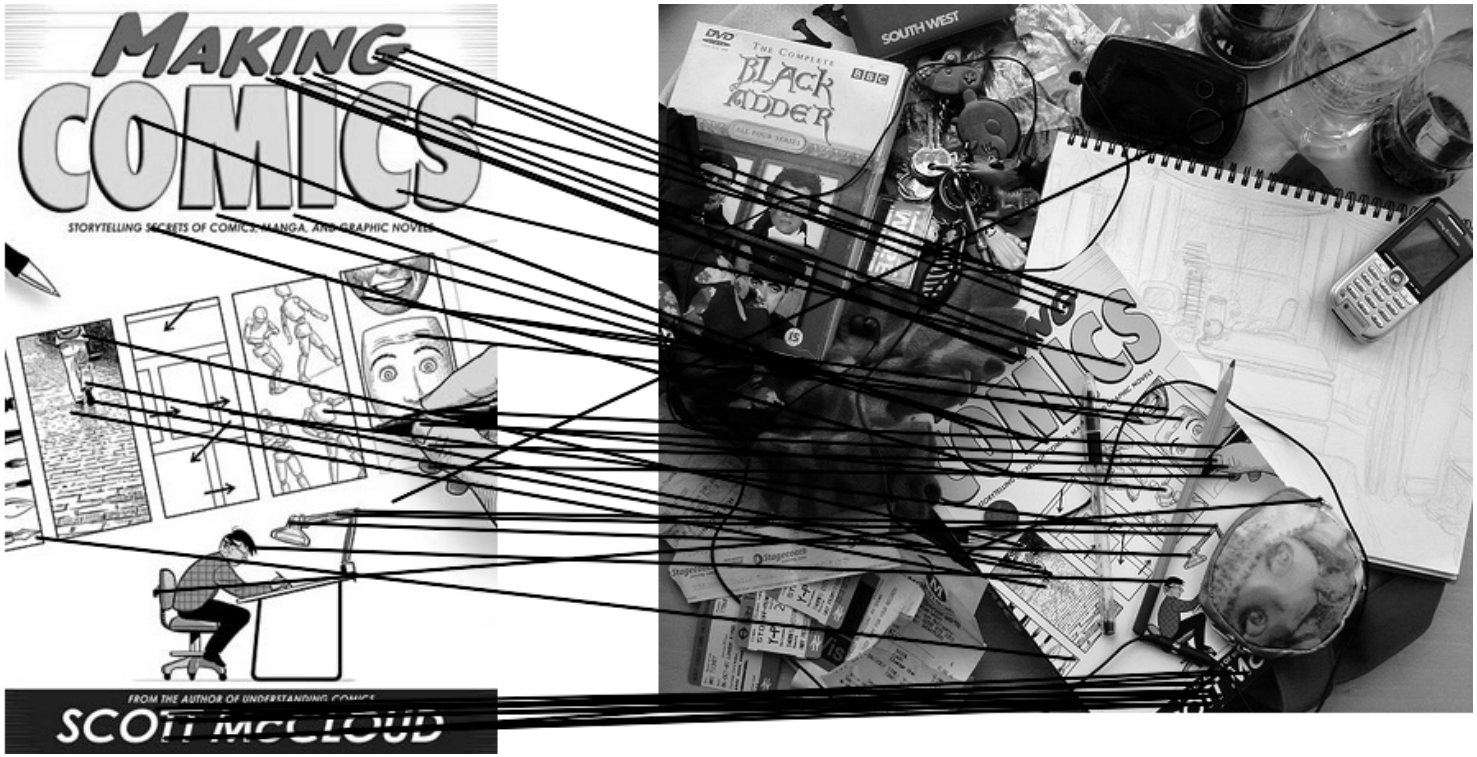
- Affine transformation approximates viewpoint changes for roughly **planar objects** and roughly **orthographic cameras** (more about these later in class)
- DVD went affine!



Computing the (Affine) Transformation

Given a set of matches between images I and J

- How can we compute the affine transformation A from I to J?
- Find transform A that best agrees with the matches

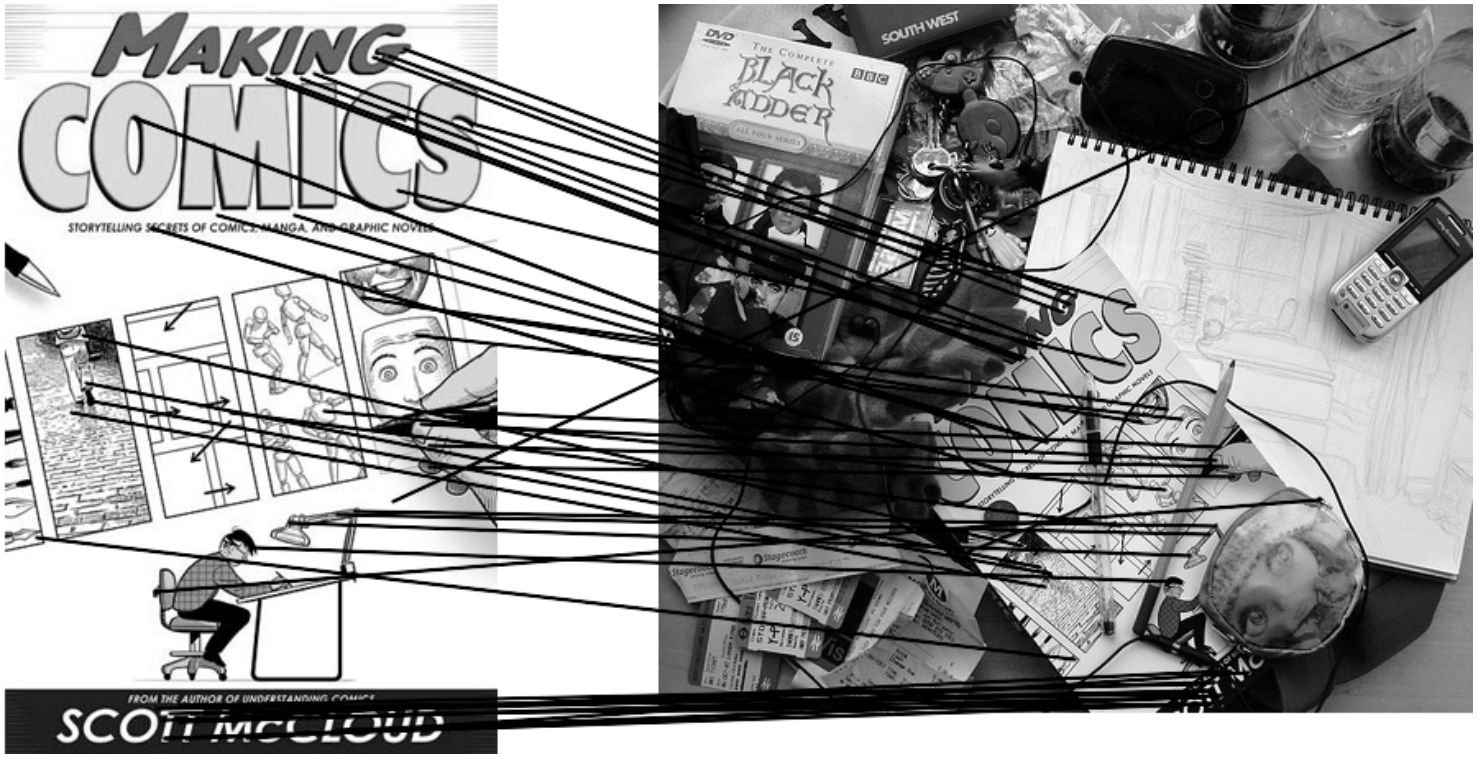


[Source: N. Snavely]

Computing the (Affine) Transformation

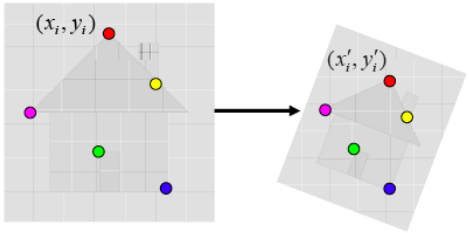
Given a set of matches between images I and J

- How can we compute the affine transformation A from I to J?
- Find transform A that best agrees with the matches



[Source: N. Snavely]

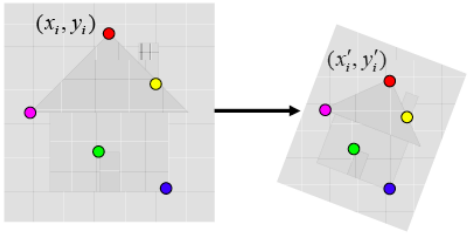
Computing the Affine Transformation



- Let (x_i, y_i) be a point on the reference (model) image, and (x_i', y_i') its match in the test image
- An affine transformation A maps (x_i, y_i) to (x_i', y_i'):

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

Computing the Affine Transformation



- Let (x_i, y_i) be a point on the reference (model) image, and (x_i', y_i') its match in the test image
- An affine transformation A maps (x_i, y_i) to (x_i', y_i'):

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

- We can rewrite this into a simple linear system:

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

Computing the Affine Transformation

- But we have many matches:

$$\underbrace{\begin{bmatrix} & & \vdots & & & \\ x_i & y_i & 0 & 0 & 1 & 0 \\ 0 & 0 & x_i & y_i & 0 & 1 \\ & & \vdots & & & \end{bmatrix}}_{\mathrm{P}} \underbrace{\begin{bmatrix} a \\ b \\ c \\ d \\ e \\ f \end{bmatrix}}_{\mathrm{a}} = \underbrace{\begin{bmatrix} \vdots \\ x_i' \\ y_i' \\ \vdots \end{bmatrix}}_{\mathrm{P}'}$$

- For each match we have two more equations

Computing the Affine Transformation

- But we have many matches:

$$\underbrace{\begin{bmatrix} & & \vdots & & & \\ x_i & y_i & 0 & 0 & 1 & 0 \\ 0 & 0 & x_i & y_i & 0 & 1 \\ & & \vdots & & & \end{bmatrix}}_{\mathrm{P}} \underbrace{\begin{bmatrix} a \\ b \\ c \\ d \\ e \\ f \end{bmatrix}}_{\mathrm{a}} = \underbrace{\begin{bmatrix} \vdots \\ x_i' \\ y_i' \\ \vdots \end{bmatrix}}_{\mathrm{P}'}$$

- For each match we have two more equations
- How many matches do we need to compute A?

Computing the Affine Transformation

- But we have many matches:

$$\underbrace{\begin{bmatrix} & & \vdots & & & \\ x_i & y_i & 0 & 0 & 1 & 0 \\ 0 & 0 & x_i & y_i & 0 & 1 \\ & & \vdots & & & \end{bmatrix}}_{\mathrm{P}} \underbrace{\begin{bmatrix} a \\ b \\ c \\ d \\ e \\ f \end{bmatrix}}_{\mathrm{a}} = \underbrace{\begin{bmatrix} \vdots \\ x_i' \\ y_i' \\ \vdots \end{bmatrix}}_{\mathrm{P}'}$$

- For each match we have two more equations
- How many matches do we need to compute A?

Computing the Affine Transformation

- But we have many matches:

$$\underbrace{\begin{bmatrix} & & \vdots & & & \\ x_i & y_i & 0 & 0 & 1 & 0 \\ 0 & 0 & x_i & y_i & 0 & 1 \\ & & \vdots & & & \end{bmatrix}}_{\mathrm{P}} \underbrace{\begin{bmatrix} a \\ b \\ c \\ d \\ e \\ f \end{bmatrix}}_{\mathrm{a}} = \underbrace{\begin{bmatrix} \vdots \\ x_i' \\ y_i' \\ \vdots \end{bmatrix}}_{\mathrm{P}'}$$

- For each match we have two more equations
- How many matches do we need to compute A?
- 6 parameters $\rightarrow$ 3 matches
- But the more, the better (more reliable)
- How do we compute A?

Computing the Affine Transformation

$$\underbrace{\begin{bmatrix} & & \vdots & & & \\ x_i & y_i & 0 & 0 & 1 & 0 \\ 0 & 0 & x_i & y_i & 0 & 1 \\ & & \vdots & & & \end{bmatrix}}_{\mathrm{P}} \underbrace{\begin{bmatrix} a \\ b \\ c \\ d \\ e \\ f \end{bmatrix}}_{\mathrm{a}} = \underbrace{\begin{bmatrix} \vdots \\ x_i' \\ y_i' \\ \vdots \end{bmatrix}}_{\mathrm{P}'}$$

- If we have 3 matches, then computing A is really easy:

$$\mathrm{a} = \mathrm{P}^{-1}\mathrm{P}'$$

Computing the Affine Transformation

$$\underbrace{\begin{bmatrix} & & \vdots & & & \\ x_i & y_i & 0 & 0 & 1 & 0 \\ 0 & 0 & x_i & y_i & 0 & 1 \\ & & \vdots & & & \end{bmatrix}}_{\mathrm{P}} \underbrace{\begin{bmatrix} a \\ b \\ c \\ d \\ e \\ f \end{bmatrix}}_{\mathrm{a}} = \underbrace{\begin{bmatrix} \vdots \\ x_i' \\ y_i' \\ \vdots \end{bmatrix}}_{\mathrm{P}'}$$

- If we have 3 matches, then computing A is really easy:

$$\mathrm{a} = \mathrm{P}^{-1}\mathrm{P}'$$

- If we have more than 3, then we do **least-squares estimation**:

$$\min_{a,b,\cdots,f} ||\mathrm{Pa} - \mathrm{P}'||_2^2$$

Computing the Affine Transformation

$$\underbrace{\begin{bmatrix} & & \vdots & & & \\ x_i & y_i & 0 & 0 & 1 & 0 \\ 0 & 0 & x_i & y_i & 0 & 1 \\ & & \vdots & & & \end{bmatrix}}_{\mathrm{P}} \underbrace{\begin{bmatrix} a \\ b \\ c \\ d \\ e \\ f \end{bmatrix}}_{\mathrm{a}} = \underbrace{\begin{bmatrix} \vdots \\ x_i' \\ y_i' \\ \vdots \end{bmatrix}}_{\mathrm{P}'}$$

- If we have 3 matches, then computing A is really easy:

$$\mathrm{a} = \mathrm{P}^{-1}\mathrm{P}'$$

- If we have more than 3, then we do **least-squares estimation**:

$$\min_{a,b,\cdots,f} ||\mathrm{Pa} - \mathrm{P}'||_2^2$$

- Which has a closed form solution:

$$\mathrm{a} = (\mathrm{P}^T\mathrm{P})^{-1}\mathrm{P}^T\mathrm{P}'$$

Image Alignment Algorithm: Affine Case

Given images I and J

1. Compute image features for I and J
2. Match features between I and J
3. Compute affine transformation A between I and J using least squares on the set of matches

Image Alignment Algorithm: Affine Case

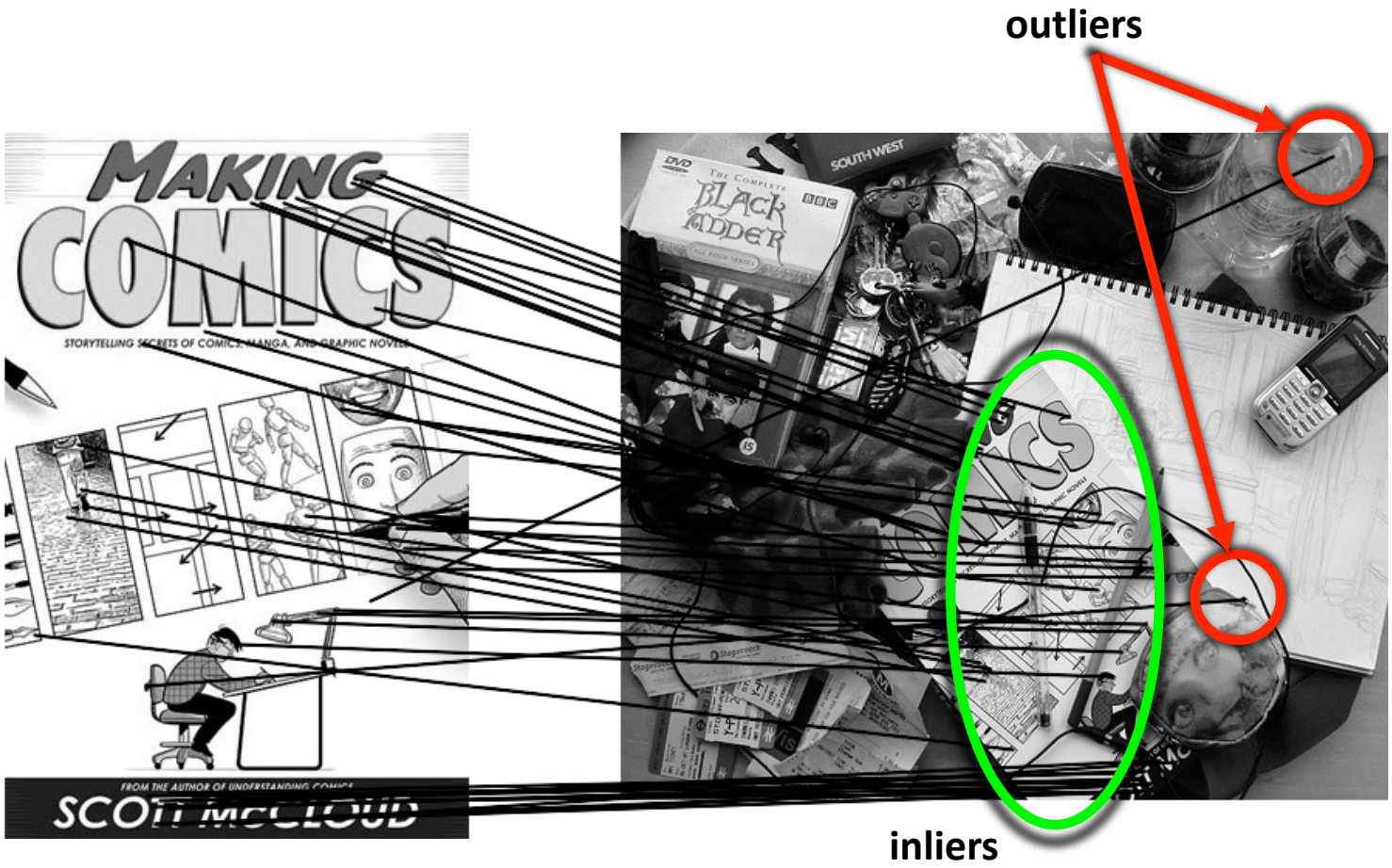
Given images I and J

1. Compute image features for I and J
2. Match features between I and J
3. Compute affine transformation A between I and J using least squares on the set of matches

Is there a problem with this?

[Source: N. Snavely]

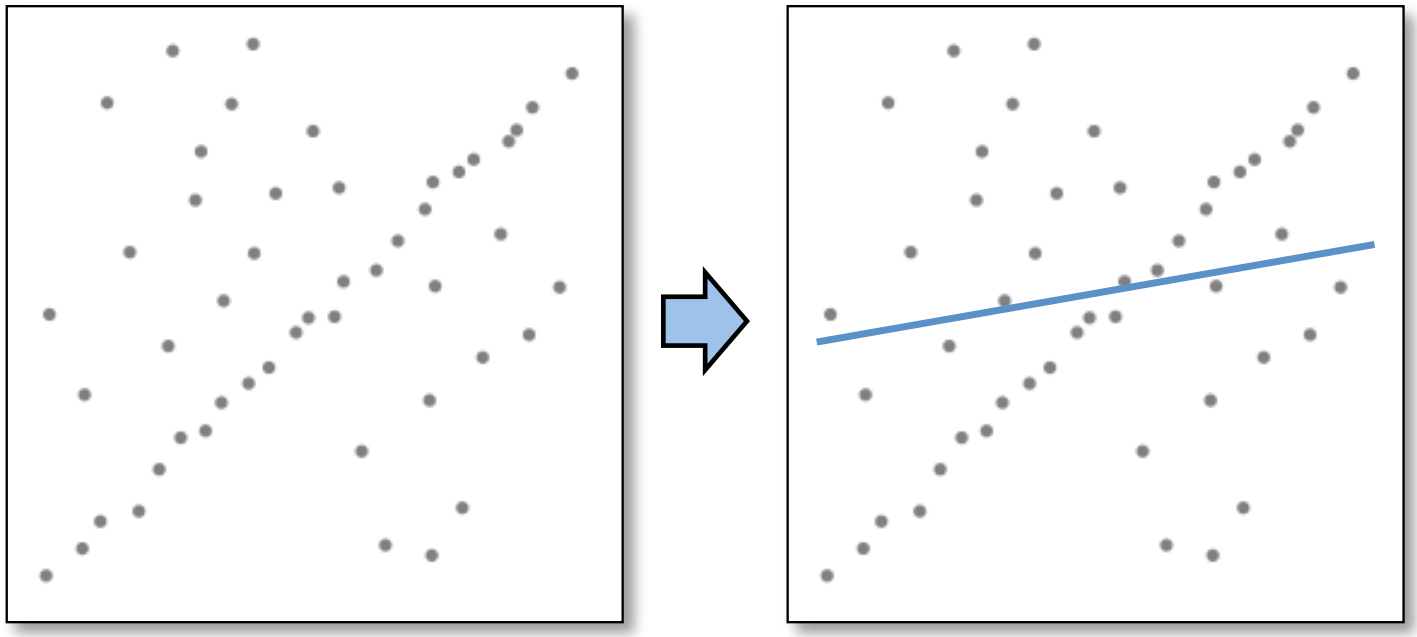
Robustness



[Source: N. Snavely]

Simple Case

- Lets consider a simpler example ... Fit a line to the points below!



Problem: Fit a line to these datapoints

Least squares fit

Simple Idea: RANSAC

- Take the minimal number of points to compute what we want. In the line example, two points (in our affine example, three matches)

Simple Idea: RANSAC

- Take the minimal number of points to compute what we want. In the line example, two points (in our affine example, three matches)
- By "take" we mean choose at random from all points

Simple Idea: RANSAC

- Take the minimal number of points to compute what we want. In the line example, two points (in our affine example, three matches)
- By "take" we mean choose at random from all points
- Fit a line to the selected pair of points

Simple Idea: RANSAC

- Take the minimal number of points to compute what we want. In the line example, two points (in our affine example, three matches)
- By "take" we mean choose at random from all points
- Fit a line to the selected pair of points
- Count the number of all points that "agree" with the line: We call the agreeing points **inliers**

Simple Idea: RANSAC

- Take the minimal number of points to compute what we want. In the line example, two points (in our affine example, three matches)
- By "take" we mean choose at random from all points
- Fit a line to the selected pair of points
- Count the number of all points that "agree" with the line: We call the agreeing points **inliers**
- "Agree" = within a small distance of the line

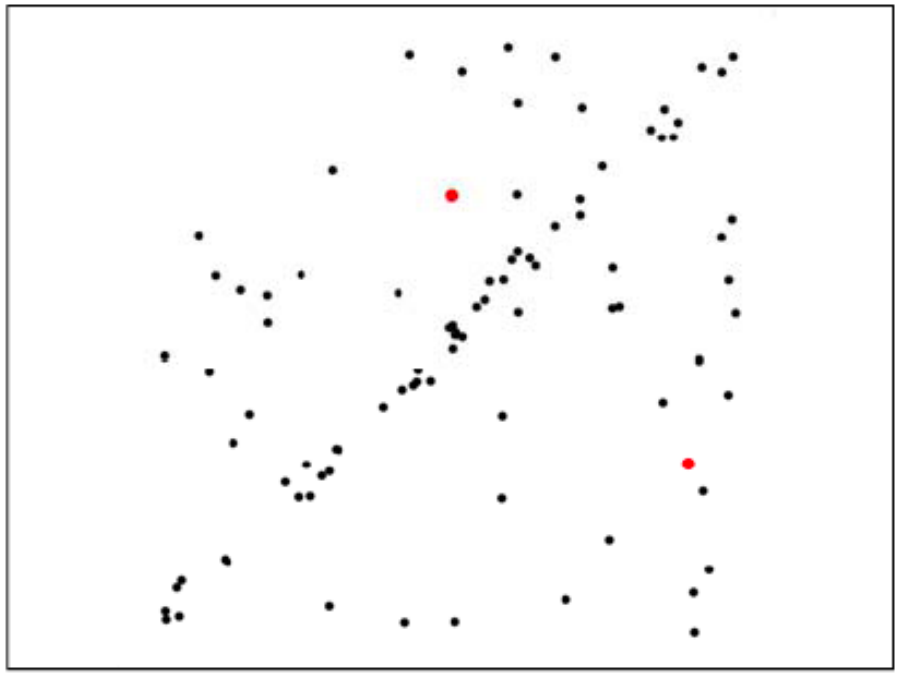
Simple Idea: RANSAC

- Take the minimal number of points to compute what we want. In the line example, two points (in our affine example, three matches)
- By "take" we mean choose at random from all points
- Fit a line to the selected pair of points
- Count the number of all points that "agree" with the line: We call the agreeing points **inliers**
- "Agree" = within a small distance of the line
- Repeat this many times, remember the number of inliers for each trial
- Among several trials, select the one with the largest number of inliers

This procedure is called **RA**ndom **SA**mple **C**onsensus

RANSAC for Line Fitting Example

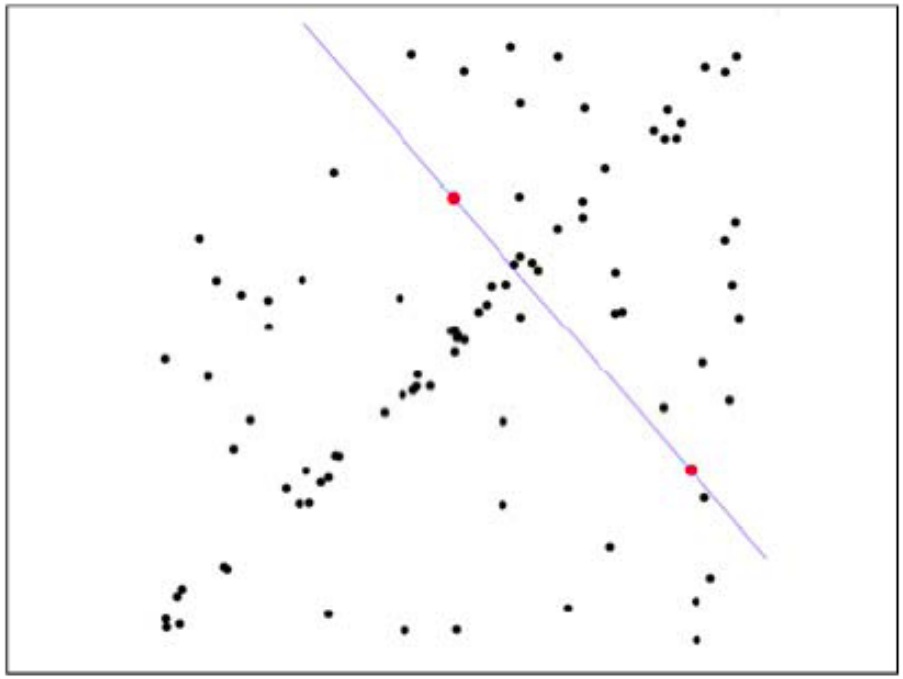
1. Randomly select minimal subset of points



[Source: R. Raguram]

RANSAC for Line Fitting Example

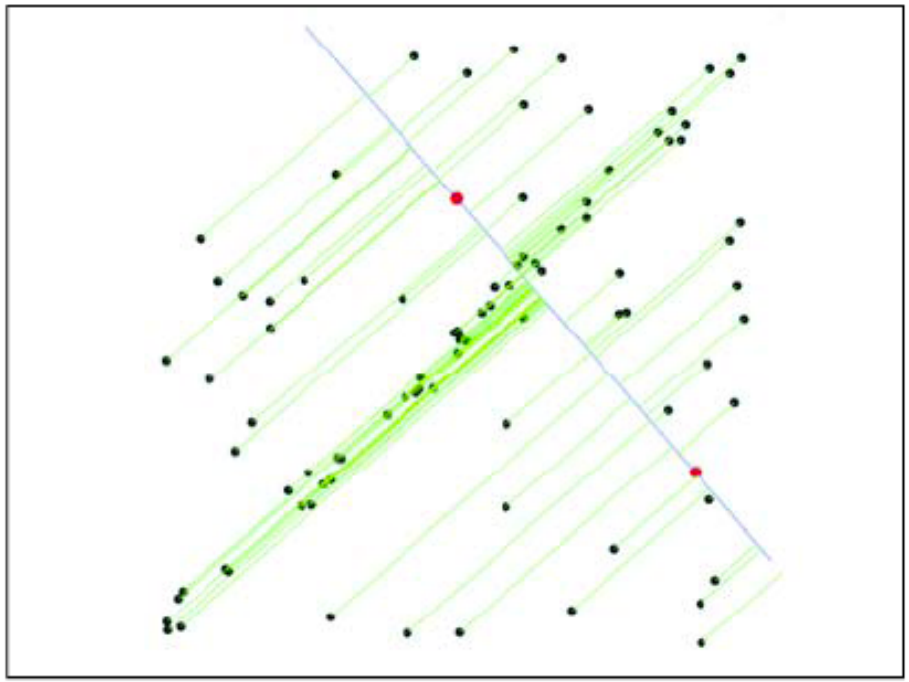
1. Randomly select minimal subset of points
2. Hypothesize a model



[Source: R. Raguram]

RANSAC for Line Fitting Example

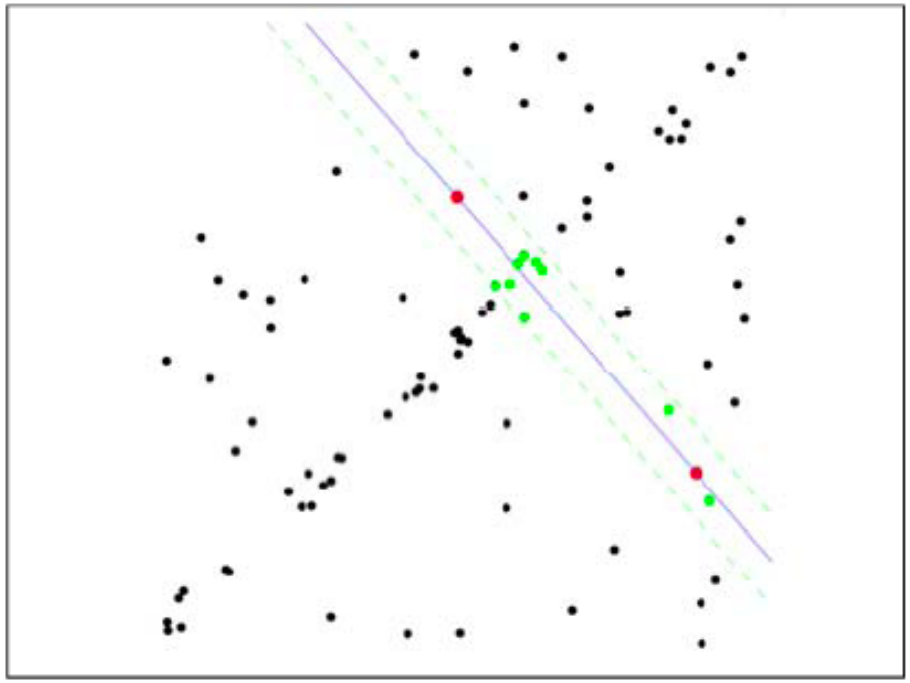
1. Randomly select minimal subset of points
2. Hypothesize a model
3. Compute error function



[Source: R. Raguram]

RANSAC for Line Fitting Example

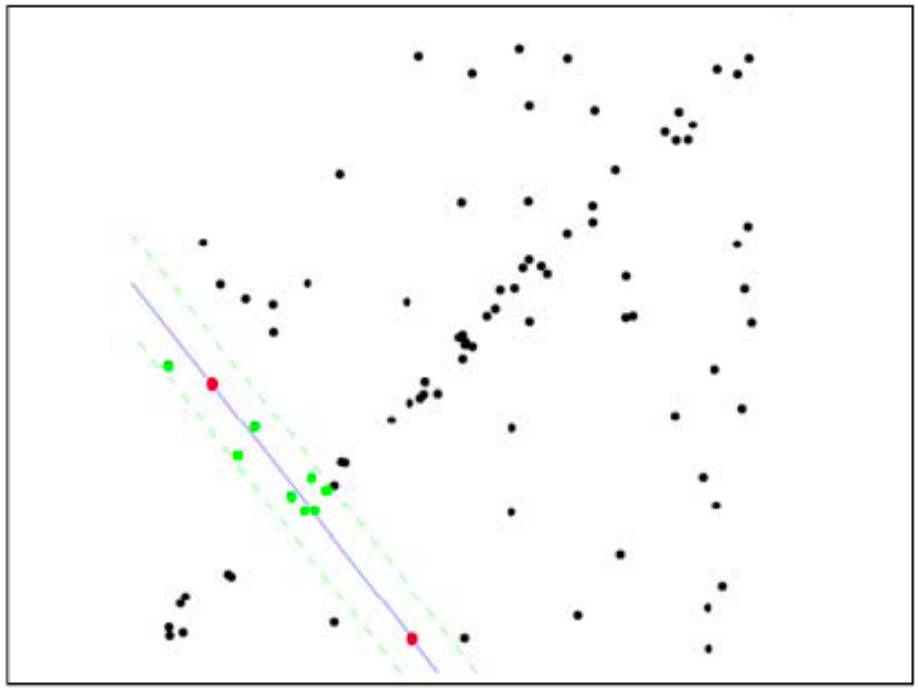
1. Randomly select minimal subset of points
2. Hypothesize a model
3. Compute error function
4. Select points consistent with model



[Source: R. Raguram]

RANSAC for Line Fitting Example

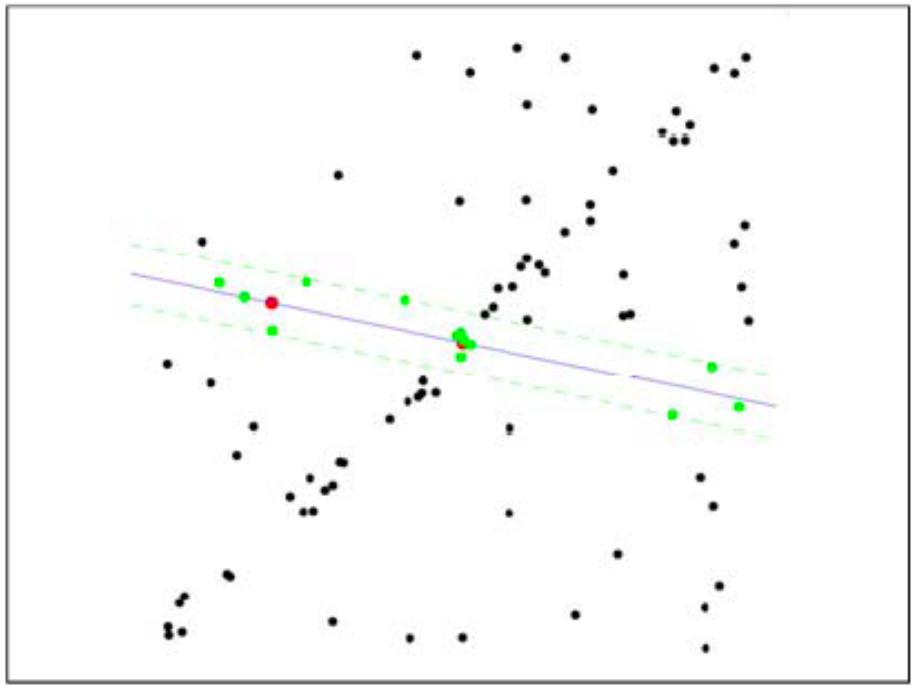
1. Randomly select minimal subset of points
2. Hypothesize a model
3. Compute error function
4. Select points consistent with model
5. Repeat hypothesize and verify loop



[Source: R. Raguram]

RANSAC for Line Fitting Example

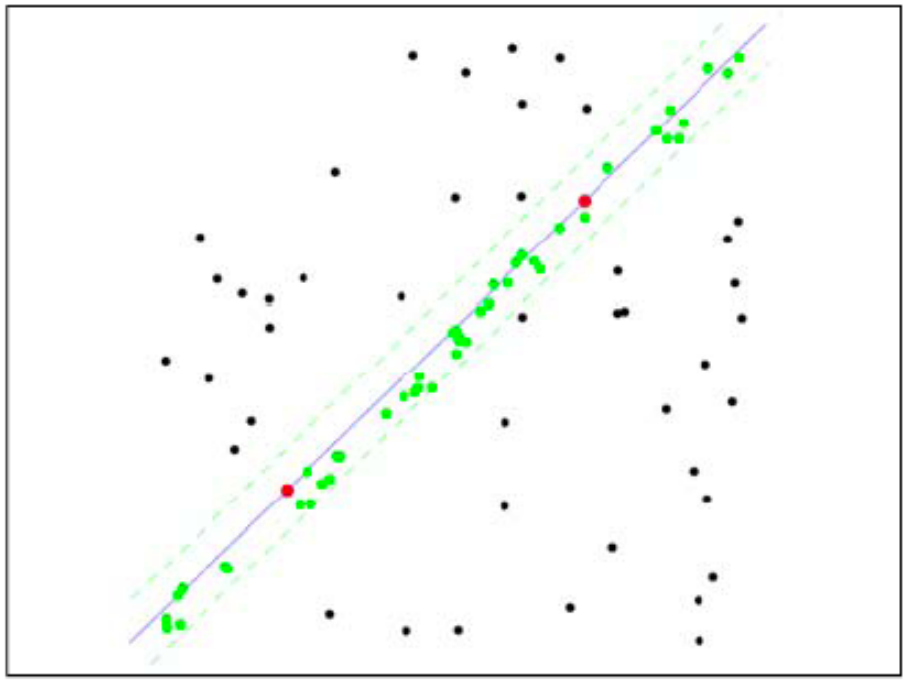
1. Randomly select minimal subset of points
2. Hypothesize a model
3. Compute error function
4. Select points consistent with model
5. Repeat hypothesize and verify loop



[Source: R. Raguram]

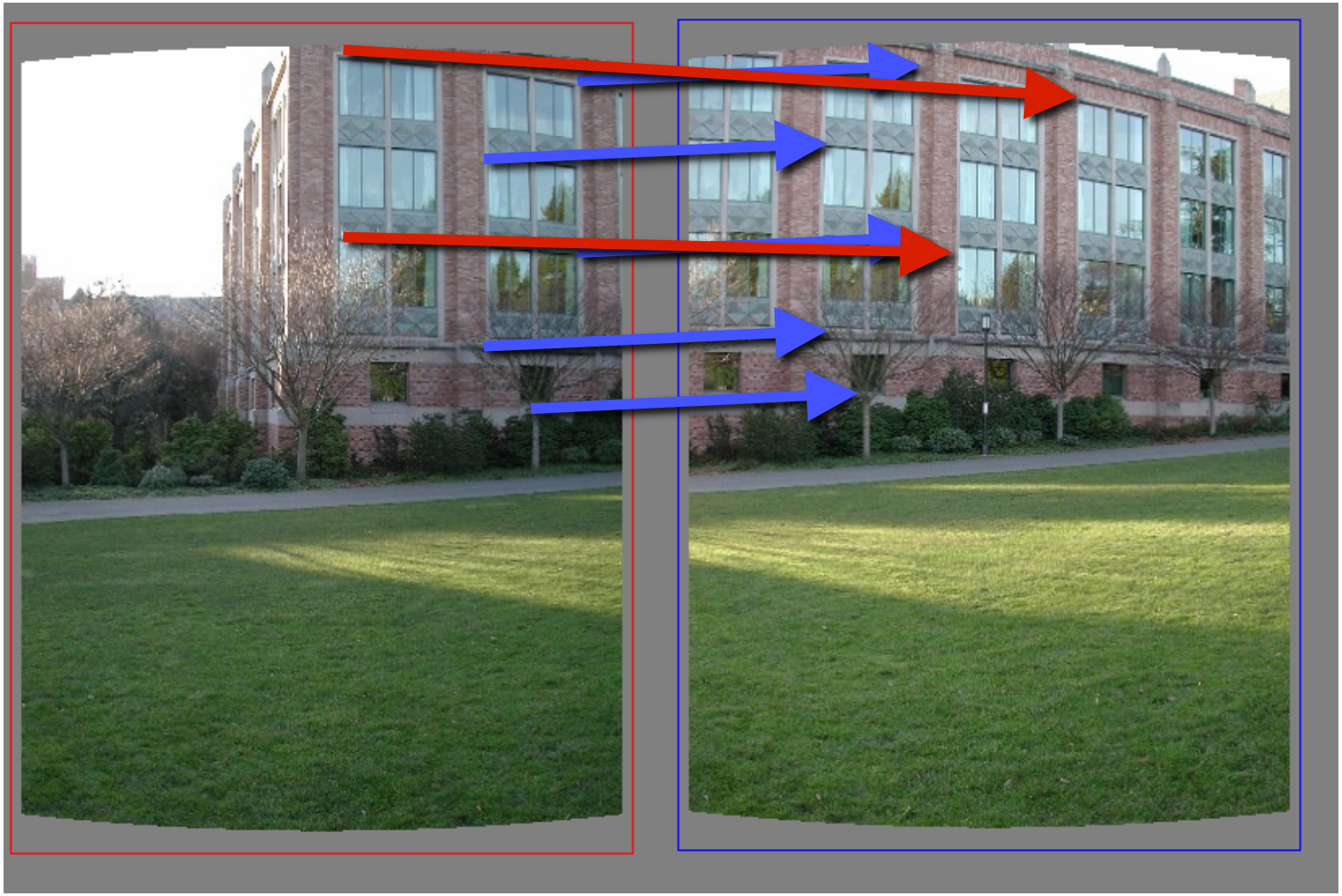
RANSAC for Line Fitting Example

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
6. Choose model with largest set of inliers



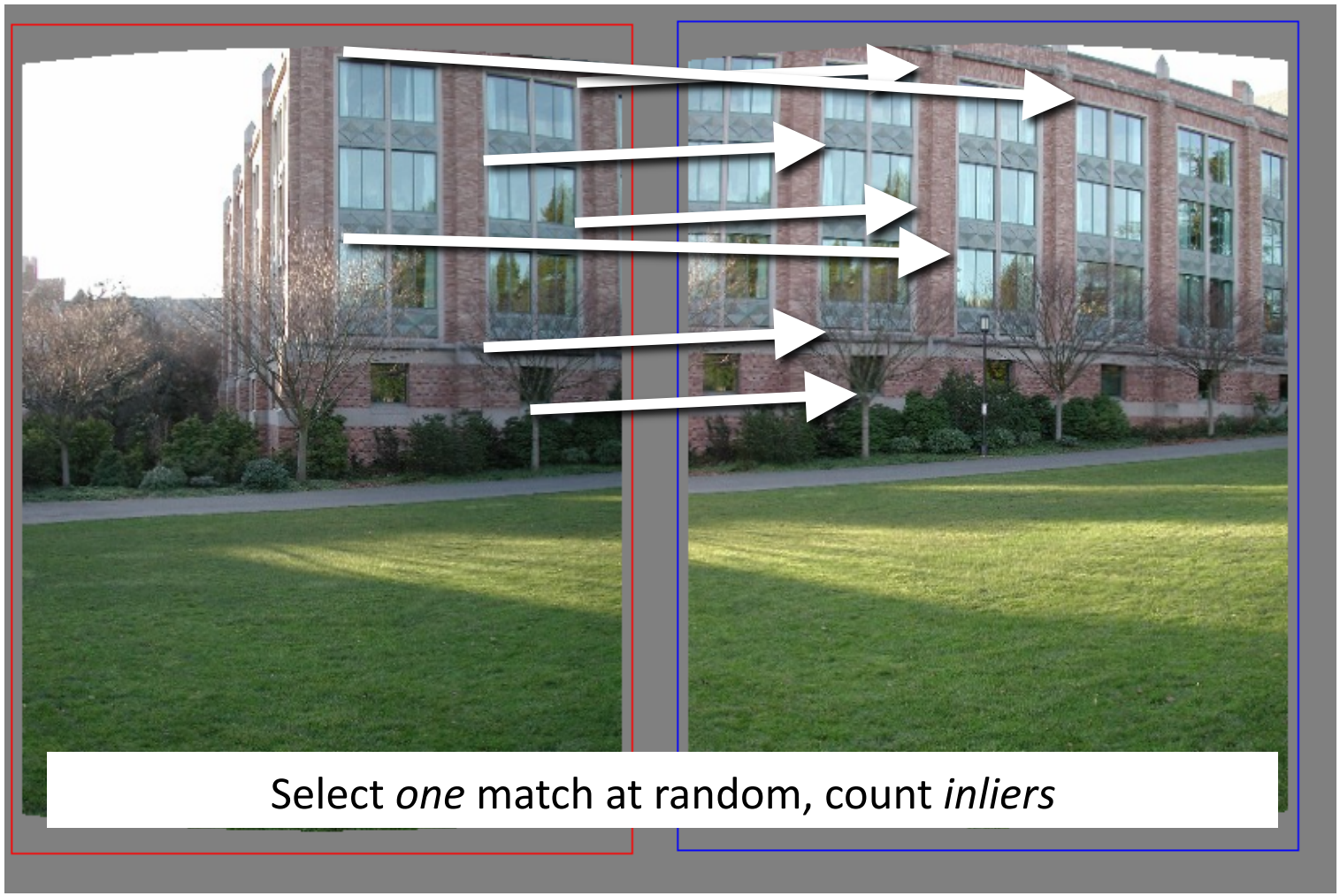
[Source: R. Raguram]

Translations



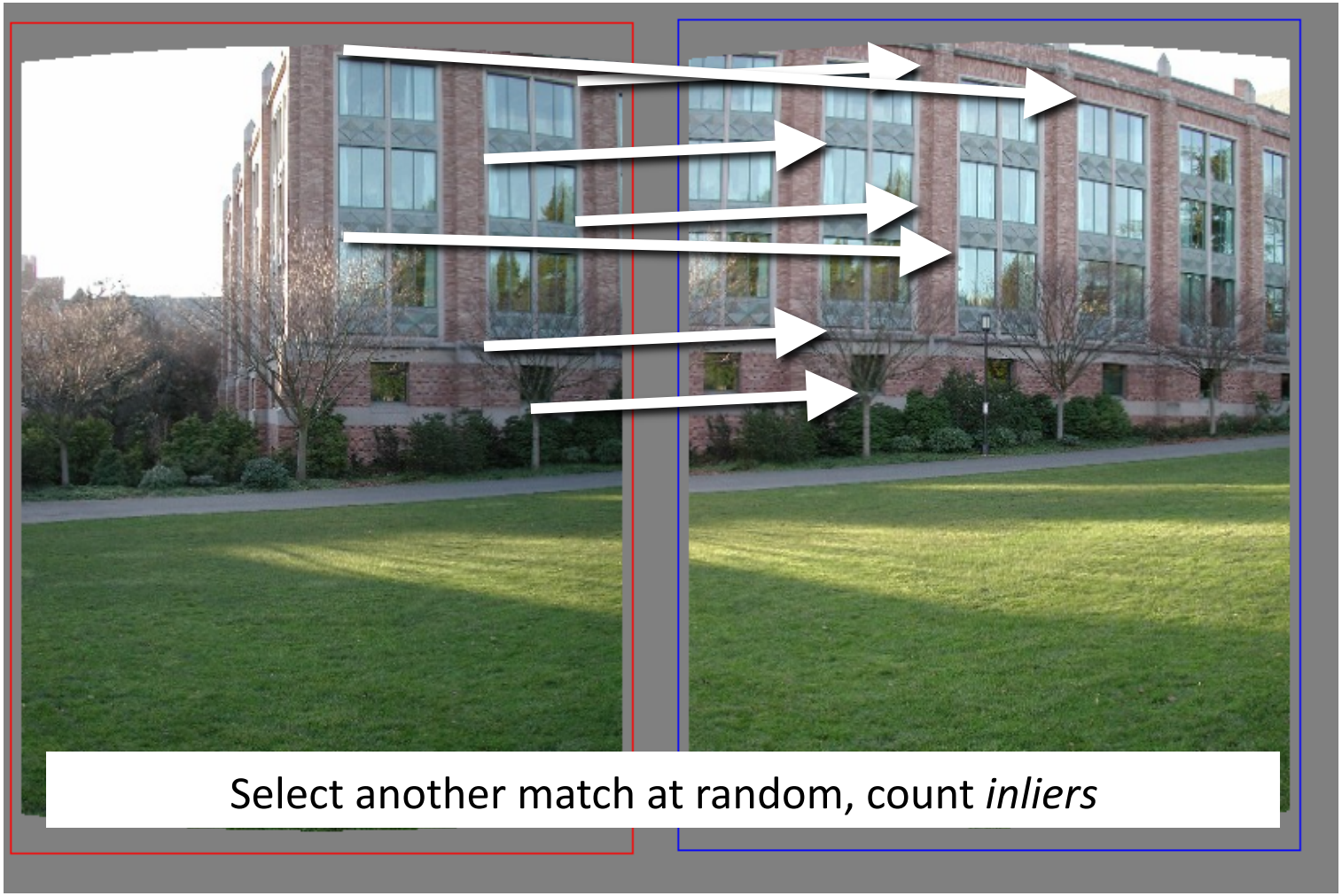
[Source: N. Snavely]

RAndom SAmple Consensus



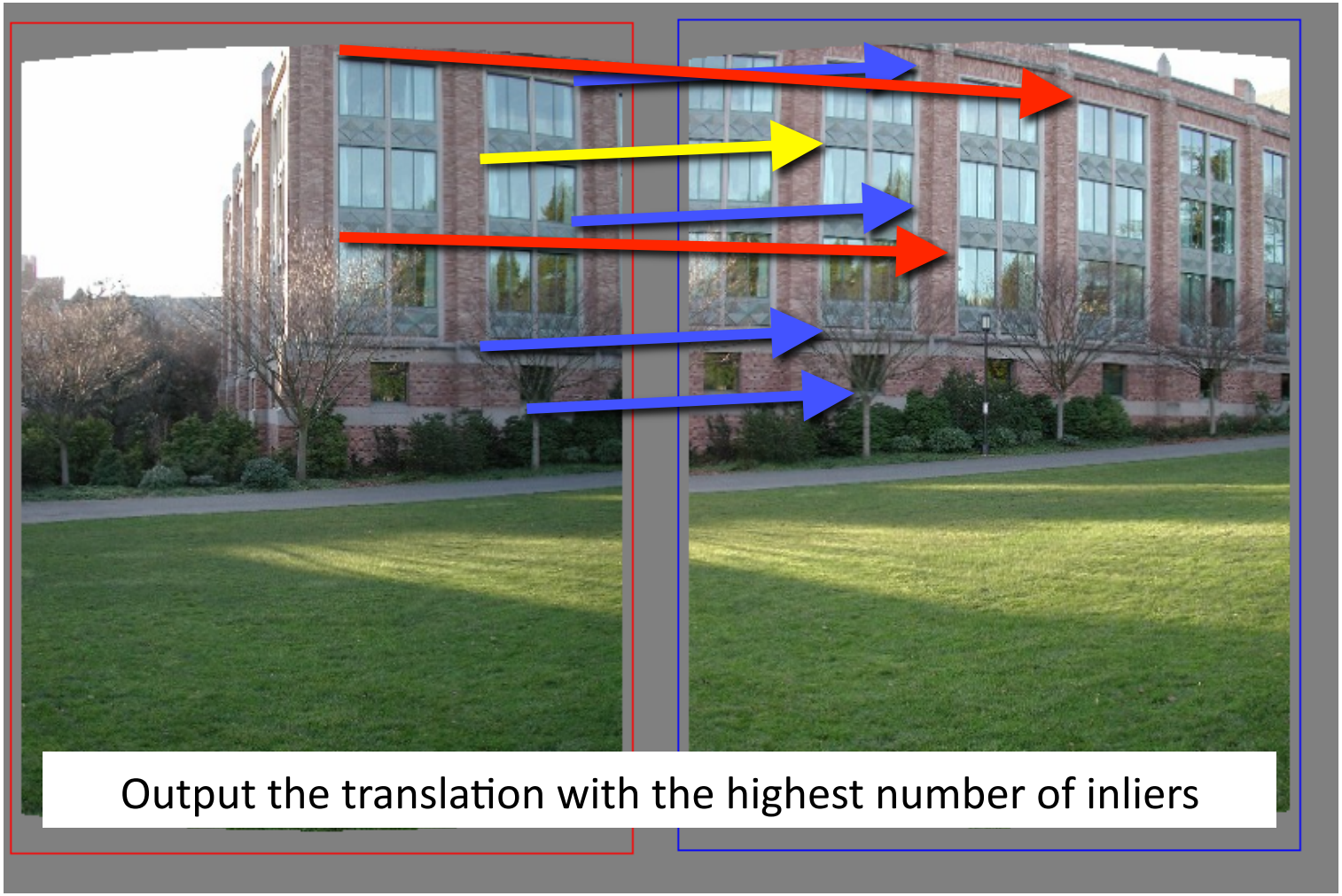
[Source: N. Snavely]

RAndom SAmple Consensus



[Source: N. Snavely]

RAndom SAmple Consensus



[Source: N. Snavely]

RANSAC

- All the inliers will agree with each other on the translation vector; the (hopefully small) number of outliers will (hopefully) disagree with each other

RANSAC

- All the inliers will agree with each other on the translation vector; the (hopefully small) number of outliers will (hopefully) disagree with each other
- RANSAC only has guarantees if there are $< 50\%$ outliers

RANSAC

- All the inliers will agree with each other on the translation vector; the (hopefully small) number of outliers will (hopefully) disagree with each other
- RANSAC only has guarantees if there are $< 50\%$ outliers
- "All good matches are alike; every bad match is bad in its own way." – [Tolstoy via Alyosha Efros]

[Source: N. Snavely]

RANSAC

- **Inlier threshold** related to the amount of noise we expect in inliers

RANSAC

- **Inlier threshold** related to the amount of noise we expect in inliers
- Often model noise as Gaussian with some standard deviation (e.g., 3 pixels)

RANSAC

- **Inlier threshold** related to the amount of noise we expect in inliers
- Often model noise as Gaussian with some standard deviation (e.g., 3 pixels)
- **Number of rounds** related to the percentage of outliers we expect, and the probability of success we'd like to guarantee

RANSAC

- **Inlier threshold** related to the amount of noise we expect in inliers
- Often model noise as Gaussian with some standard deviation (e.g., 3 pixels)
- **Number of rounds** related to the percentage of outliers we expect, and the probability of success we'd like to guarantee
- Suppose there are 20% outliers, and we want to find the correct answer with 99% probability

RANSAC

- **Inlier threshold** related to the amount of noise we expect in inliers
- Often model noise as Gaussian with some standard deviation (e.g., 3 pixels)
- **Number of rounds** related to the percentage of outliers we expect, and the probability of success we'd like to guarantee
- Suppose there are 20% outliers, and we want to find the correct answer with 99% probability
- How many rounds do we need?

[Source: R. Urtasun]

How many rounds?

- Sufficient number of trials S must be tried.

How many rounds?

- Sufficient number of trials S must be tried.
- Let p be the probability that any given correspondence is valid and P be the total probability of success after S trials.

How many rounds?

- Sufficient number of trials S must be tried.
- Let p be the probability that any given correspondence is valid and P be the total probability of success after S trials.
- The likelihood in one trial that all k random samples are inliers is p^k

How many rounds?

- Sufficient number of trials S must be tried.
- Let p be the probability that any given correspondence is valid and P be the total probability of success after S trials.
- The likelihood in one trial that all k random samples are inliers is p^k
- The likelihood that S such trials will all fail is

$$1 - P = (1 - p^k)^S$$

How many rounds?

- Sufficient number of trials S must be tried.
- Let p be the probability that any given correspondence is valid and P be the total probability of success after S trials.
- The likelihood in one trial that all k random samples are inliers is p^k
- The likelihood that S such trials will all fail is

$$1 - P = (1 - p^k)^S$$

- The required minimum number of trials is

$$S = \frac{\log(1 - P)}{\log(1 - p^k)}$$

How many rounds?

- Sufficient number of trials S must be tried.
- Let p be the probability that any given correspondence is valid and P be the total probability of success after S trials.
- The likelihood in one trial that all k random samples are inliers is p^k
- The likelihood that S such trials will all fail is

$$1 - P = (1 - p^k)^S$$

- The required minimum number of trials is

$$S = \frac{\log(1 - P)}{\log(1 - p^k)}$$

- The number of trials grows quickly with the number of sample points used.

How many rounds?

- Sufficient number of trials S must be tried.
- Let p be the probability that any given correspondence is valid and P be the total probability of success after S trials.
- The likelihood in one trial that all k random samples are inliers is p^k
- The likelihood that S such trials will all fail is

$$1 - P = (1 - p^k)^S$$

- The required minimum number of trials is

$$S = \frac{\log(1 - P)}{\log(1 - p^k)}$$

- The number of trials grows quickly with the number of sample points used.

[Source: R. Urtasun]

RANSAC pros and cons

Pros

- Simple and general

RANSAC pros and cons

Pros

- Simple and general
- Applicable to many different problems

RANSAC pros and cons

Pros

- Simple and general
- Applicable to many different problems
- Often works well in practice

RANSAC pros and cons

Pros

- Simple and general
- Applicable to many different problems
- Often works well in practice

Cons

- Parameters to tune

RANSAC pros and cons

Pros

- Simple and general
- Applicable to many different problems
- Often works well in practice

Cons

- Parameters to tune
- Sometimes too many iterations are required

RANSAC pros and cons

Pros

- Simple and general
- Applicable to many different problems
- Often works well in practice

Cons

- Parameters to tune
- Sometimes too many iterations are required
- Can fail for extremely low inlier ratios

RANSAC pros and cons

Pros

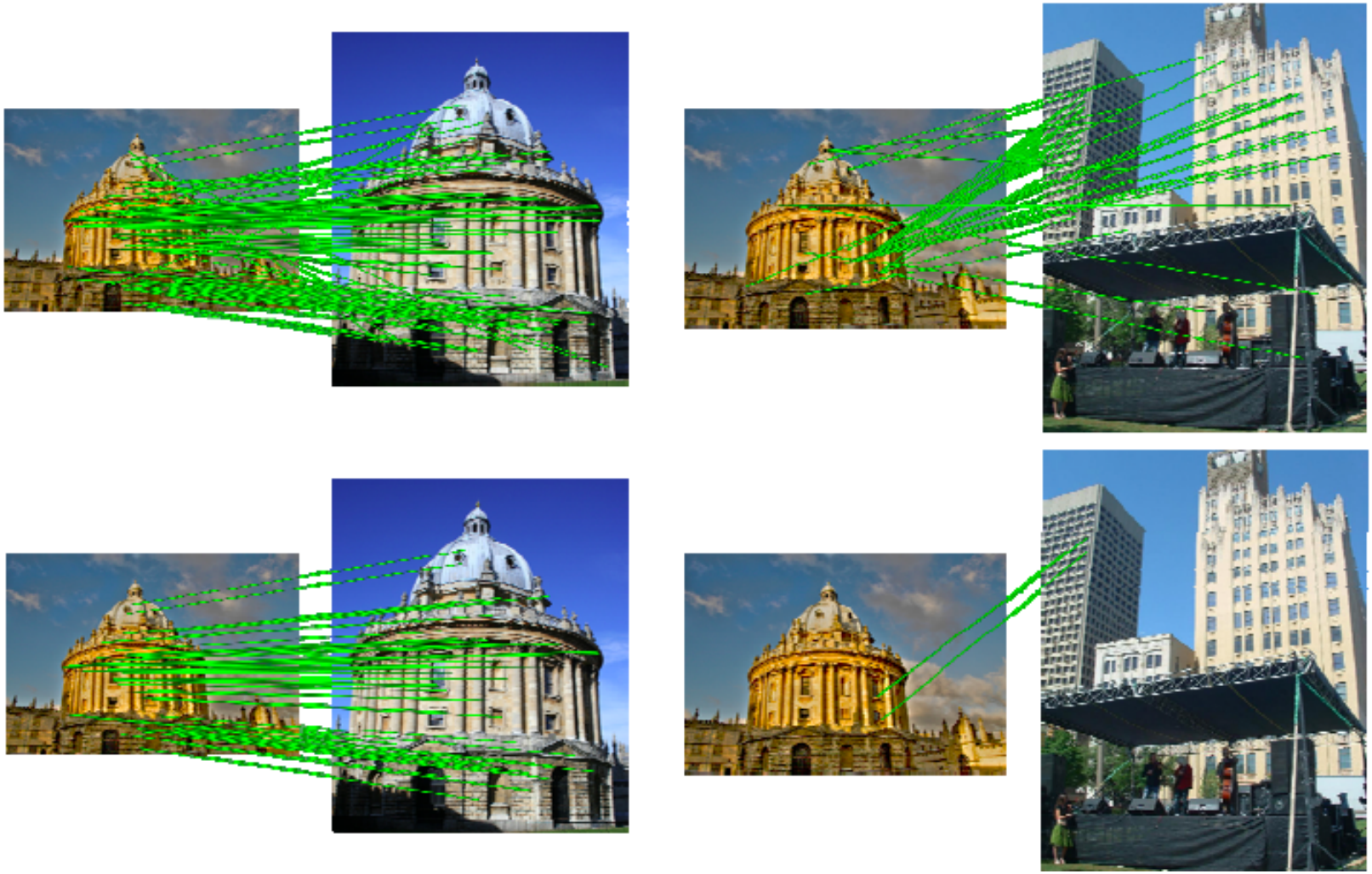
- Simple and general
- Applicable to many different problems
- Often works well in practice

Cons

- Parameters to tune
- Sometimes too many iterations are required
- Can fail for extremely low inlier ratios
- We can often do better than brute-force sampling

[Source: N. Snavely, slide credit: R. Urtasun]

Ransac Verification



[Source: K. Grauman, slide credit: R. Urtasun]

Summary – Stuff You Need To Know

To match image I and J under affine transformation:

- Compute scale and rotation invariant keypoints in both images
- Compute a (rotation invariant) feature vector in each keypoint (e.g., SIFT)
- Match all features in I to all features in J
- For each feature in reference image I find closest match in J
- If ratio between closest and second closest match is < 0.8, keep match
- Do RANSAC to compute affine transformation A:
  - Select 3 matches at random
  - Compute A
  - Compute the number of inliers
  - Repeat
  - Find A that gave the most inliers