Computer Vision: Image Alignment

Raquel Urtasun

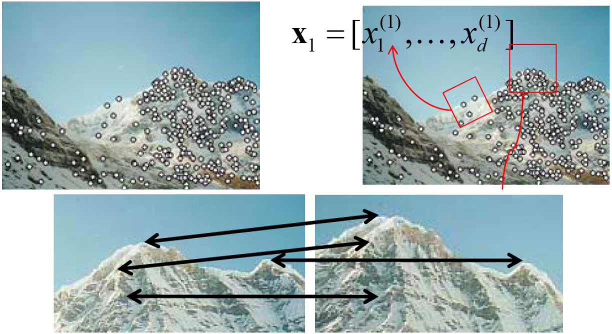
TTI Chicago

Jan 22, 2013

What did we see in class last week?

Local features

- **Detection**: Identify the interest points.
- **Description**: Extract vector feature descriptor around each interest point.
- **Matching**: Determine correspondence between descriptors in two views.



[Source: K. Grauman]

Detecting features

- **Harris corner detector**: looks at the singular values of the autocorrelation matrix
- **Laplacian of Gaussians**: Detects blobs
- **Difference of Gaussians**: fast approximation of the LOG

Properties of the ideal feature

- **Local**: features are local, so robust to occlusion and clutter (no prior segmentation).
- **Invariant**: to certain transformations, e.g, scale, rotation.

Properties of the ideal feature

- **Local**: features are local, so robust to occlusion and clutter (no prior segmentation).
- **Invariant**: to certain transformations, e.g, scale, rotation.
- **Robust**: noise, blur, discretization, compression, etc. do not have a big impact on the feature.

Properties of the ideal feature

- **Local**: features are local, so robust to occlusion and clutter (no prior segmentation).
- **Invariant**: to certain transformations, e.g, scale, rotation.
- **Robust**: noise, blur, discretization, compression, etc. do not have a big impact on the feature.
- **Distinctive**: individual features can be matched to a large database of objects.

Properties of the ideal feature

- **Local**: features are local, so robust to occlusion and clutter (no prior segmentation).
- **Invariant**: to certain transformations, e.g, scale, rotation.
- **Robust**: noise, blur, discretization, compression, etc. do not have a big impact on the feature.
- **Distinctive**: individual features can be matched to a large database of objects.
- **Quantity**: many features can be generated for even small objects.

Properties of the ideal feature

- **Local**: features are local, so robust to occlusion and clutter (no prior segmentation).
- **Invariant**: to certain transformations, e.g, scale, rotation.
- **Robust**: noise, blur, discretization, compression, etc. do not have a big impact on the feature.
- **Distinctive**: individual features can be matched to a large database of objects.
- **Quantity**: many features can be generated for even small objects.
- **Accurate**: precise localization.

Properties of the ideal feature

- **Local**: features are local, so robust to occlusion and clutter (no prior segmentation).
- **Invariant**: to certain transformations, e.g, scale, rotation.
- **Robust**: noise, blur, discretization, compression, etc. do not have a big impact on the feature.
- **Distinctive**: individual features can be matched to a large database of objects.
- **Quantity**: many features can be generated for even small objects.
- **Accurate**: precise localization.
- **Efficient**: close to real-time performance.

Properties of the ideal feature

- **Local**: features are local, so robust to occlusion and clutter (no prior segmentation).
- **Invariant**: to certain transformations, e.g, scale, rotation.
- **Robust**: noise, blur, discretization, compression, etc. do not have a big impact on the feature.
- **Distinctive**: individual features can be matched to a large database of objects.
- **Quantity**: many features can be generated for even small objects.
- **Accurate**: precise localization.
- **Efficient**: close to real-time performance.

Describing features

- Normalized gray-scale
- SIFT
- PCA-SIFT
- GLOH

The ideal feature descriptor

- Repeatable (invariant/robust)
- Distinctive
- Compact
- Efficient

Matching local features

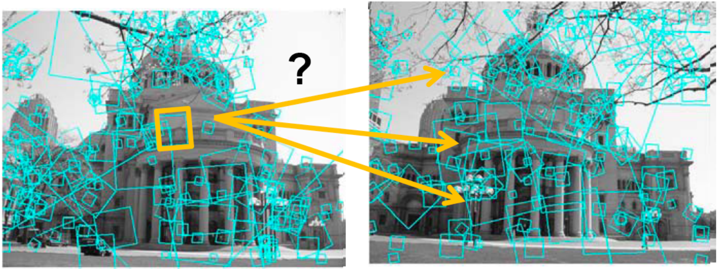
Once we have extracted features and their descriptors, we need to match the features between these images.

- **Matching strategy:** which correspondences are passed on to the next stage
- Devise **efficient data structures and algorithms** to perform this matching

Figure: Images from K. Grauman

Matching local features

- To **generate candidate matches**, find patches that have the most similar appearance (e.g., lowest SSD)
- Simplest approach: **compare them all**, take the closest (or closest k, or within a thresholded distance)

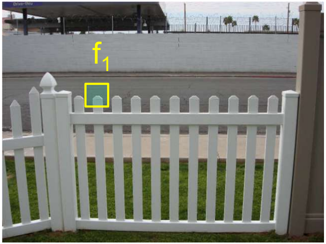


[Source: K. Grauman]

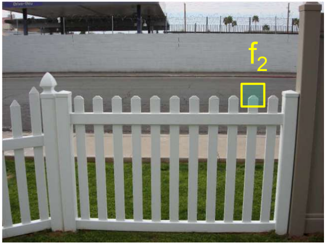
Feature Distance

How to define the difference between two features f_1, f_2?

- Simple approach: L2 distance, $||f_1 - f_2||_2$
- can give good scores to ambiguous (incorrect) matches



I_1



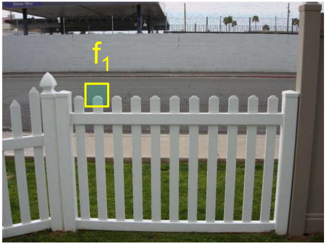
I_2

[Source: N. Snavely]

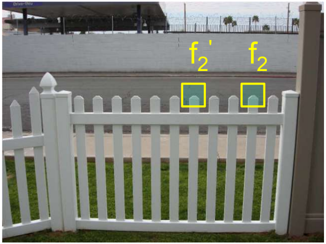
Feature Distance

Better approach: ratio distance $\frac{||f_1 - f_2||_2}{||f_1 - f_2'||_2}$

- f_2 is best SSD match to f_1 in I_2
- f_2' is 2nd best SSD match to f_1 in I_2
- gives large values for ambiguous matches



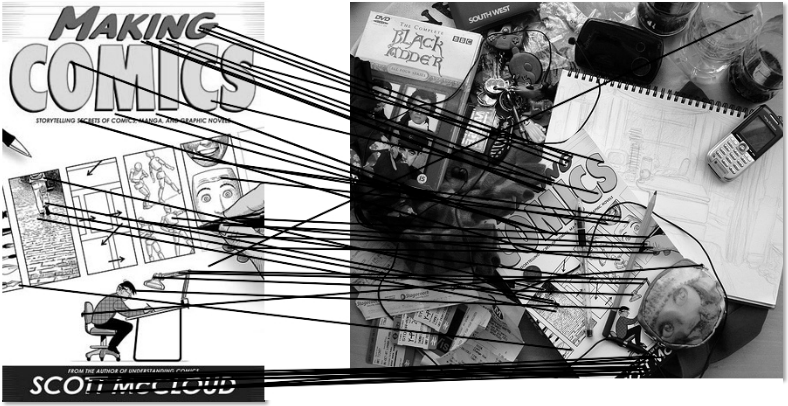
I_1



I_2

[Source: N. Snavely]

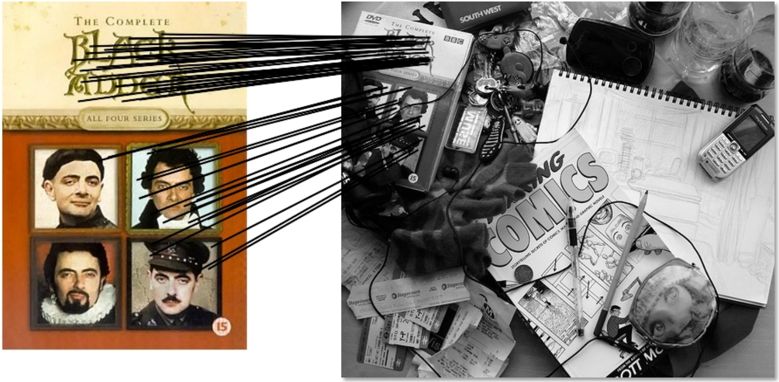
Matching Example



51 matches

[Source: N. Snavely]

Matching Example

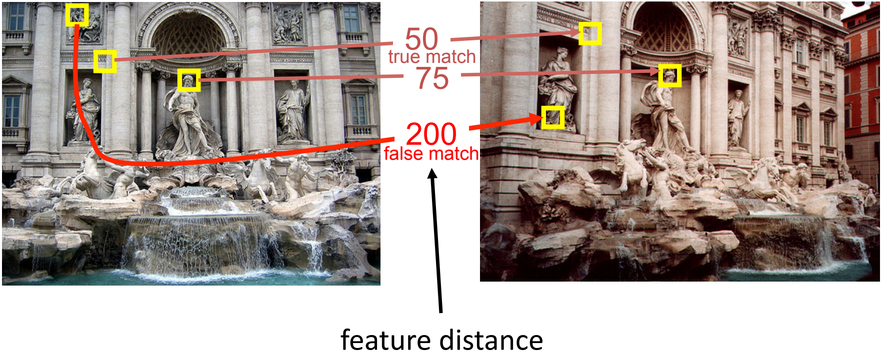


58 matches

[Source: N. Snavely]

How to measure performance

- How can we measure the performance of a feature matcher?



[Source: N. Snavely]

Measuring performance

- **Area under the curve (AUC)** is a way to summarize ROC with 1 number.
- **Mean average precision**, which is the average precision (PPV) as you vary the threshold, i.e., area under the curve in the precision-recall curve.
- The **equal error rate** is sometimes used as well.

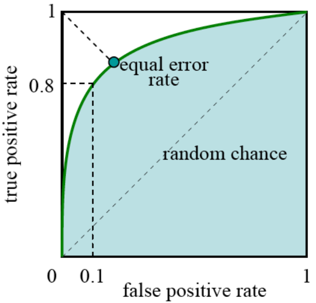


Figure: Images from R. Szeliski

Let's look at image alignment

Readings

- Chapter 3.6, 4.3 and 6.1 of Szeliski's book

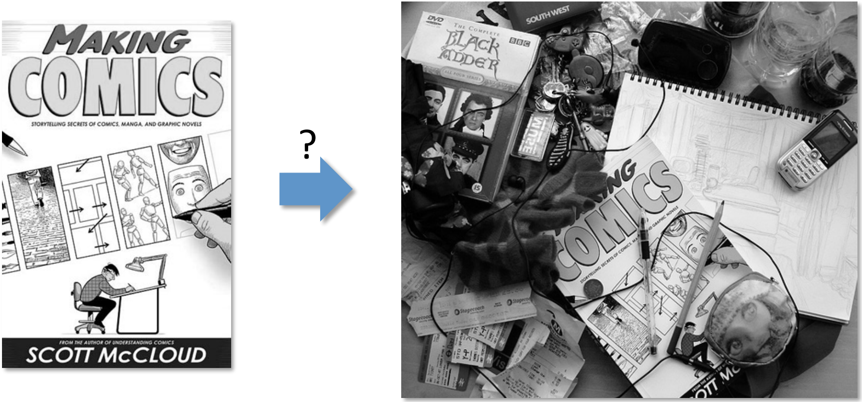
Image Alignment

Why don't this images line up exactly?

[Source: N. Snavely]

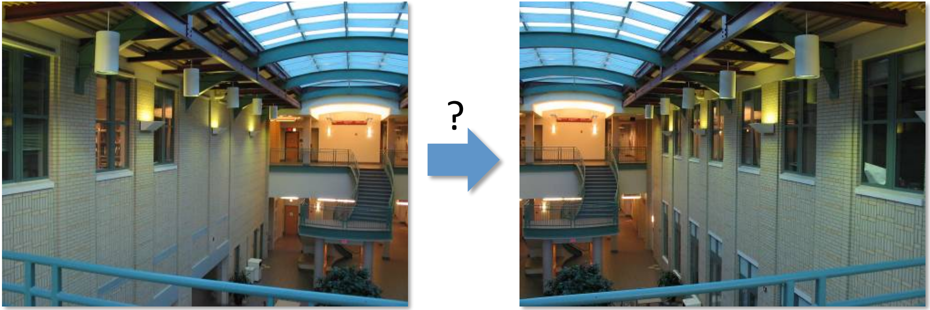
What is the geometric relationship between these images?

- Answer: Similarity transformation (translation, rotation, uniform scale)



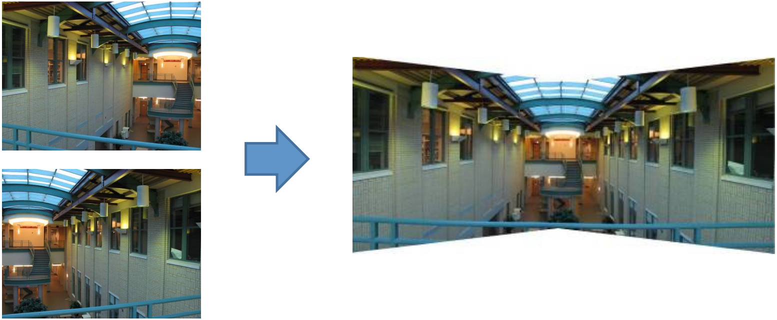
[Source: N. Snavely]

What is the geometric relationship between these images?



[Source: N. Snavely]

What is the geometric relationship between these images?



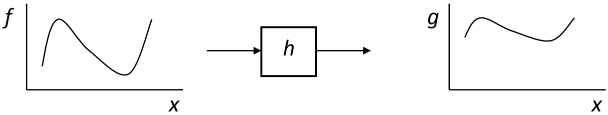
Very important for creating mosaics!

[Source: N. Snavely]

Image Warping

- **Image filtering**: change *range* of image

$$g(x) = h(f(x))$$

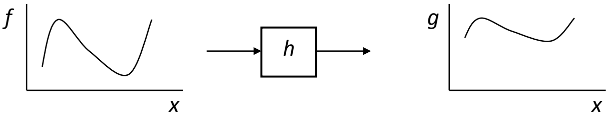


[Source: R. Szeliski]

Image Warping

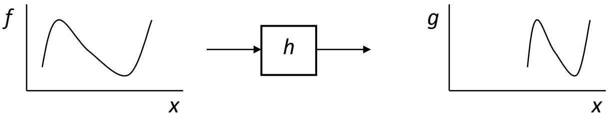
- **Image filtering**: change *range* of image

$$g(x) = h(f(x))$$



- **Image warping**: change *domain* of image

$$g(x) = f(h(x))$$

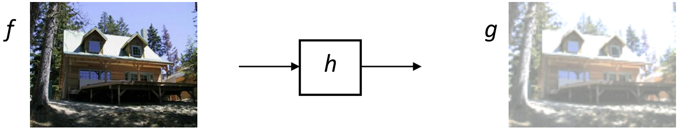


[Source: R. Szeliski]

Image Warping

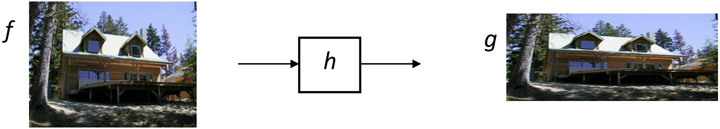
- **Image filtering**: change range of image

$$g(x) = h(f(x))$$



- **Image warping**: change domain of image

$$g(x) = f(h(x))$$



[Source: R. Szeliski]

Parametric (global) warping

- Examples of parametric warps:

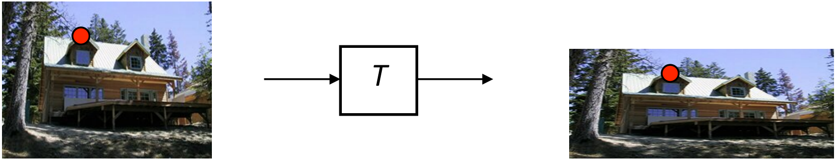
translation

rotation

aspect

- Why is it call parametric?

Parametric (global) warping



p = (x,y) **p’** = (x’,y’)

- Transformation T is a coordinate-changing machine:

$$p' = T(p)$$

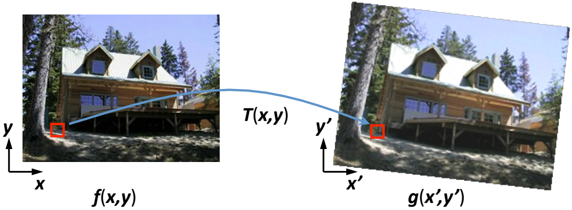
- What does it mean that T is global?
  - Is the same for any point p
  - Can be described by just a few numbers (parameters)

[Source: N. Snavely]

Image Warping

- Given a transformation specified by $x' = h(x)$ and a source image $f(x)$, how do we compute the values of the pixels in the new image

$$g(x) = f(h(x))$$



Forward Warping

- Send each pixel $f(x)$ to its corresponding location $(x', y') = T(x, y)$ in $g(x', y')$

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

For every pixel x in $f(x)$

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

- What are the problems with this?

Problems of Forward-Warp

1. What it the value of h(x) is non-integer? What do we do?
   - Round the value of x' to the nearest integer coordinate and copy the pixel there, but severe aliasing and pixels that jump around

Problems of Forward-Warp

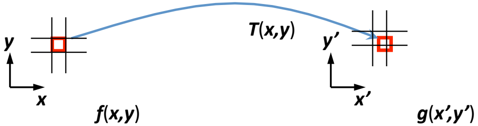
1. What it the value of h(x) is non-integer? What do we do?
   - Round the value of x' to the nearest integer coordinate and copy the pixel there, but severe aliasing and pixels that jump around
   - Distribute the value among its nearest neighbors in a weighted (bilinear) fashion, keeping track of the per-pixel weights and normalizing at the end.

Problems of Forward-Warp

1. What it the value of h(x) is non-integer? What do we do?
   - Round the value of x' to the nearest integer coordinate and copy the pixel there, but severe aliasing and pixels that jump around
   - Distribute the value among its nearest neighbors in a weighted (bilinear) fashion, keeping track of the per-pixel weights and normalizing at the end.
   - This is call **splatting**, it suffers from both moderate amounts of aliasing and a fair amount of blur

Problems of Forward-Warp

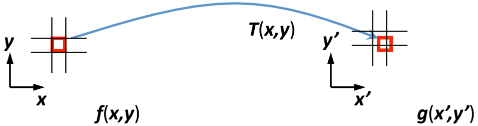
1. What it the value of h(x) is non-integer? What do we do?
   - Round the value of x' to the nearest integer coordinate and copy the pixel there, but severe aliasing and pixels that jump around
   - Distribute the value among its nearest neighbors in a weighted (bilinear) fashion, keeping track of the per-pixel weights and normalizing at the end.
   - This is call **splatting**, it suffers from both moderate amounts of aliasing and a fair amount of blur
2. Appearance of cracks and holes, especially when magnifying an image
   - Filling such holes with their nearby neighbors can lead to further aliasing and blurring



What should we do?

Problems of Forward-Warp

1. What it the value of h(x) is non-integer? What do we do?
   - Round the value of x' to the nearest integer coordinate and copy the pixel there, but severe aliasing and pixels that jump around
   - Distribute the value among its nearest neighbors in a weighted (bilinear) fashion, keeping track of the per-pixel weights and normalizing at the end.
   - This is call **splatting**, it suffers from both moderate amounts of aliasing and a fair amount of blur
2. Appearance of cracks and holes, especially when magnifying an image
   - Filling such holes with their nearby neighbors can lead to further aliasing and blurring



What should we do?

Inverse-Warping

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 $g(\boldsymbol{x}')$

- Each pixel at the destination is sampled from the original image
- How does this differ from forward mapping?

Inverse-Warping

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

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 $g(\boldsymbol{x}')$

- Each pixel at the destination is sampled from the original image
- How does this differ from forward mapping?
- Since $\hat{h}(x')$ is defined for all pixels in $g(x')$, we no longer have holes

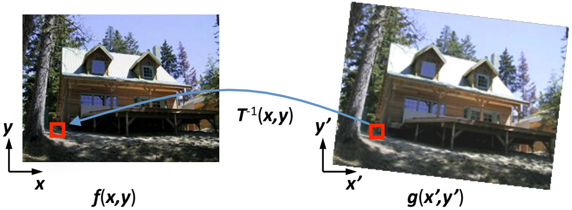
Inverse-Warping

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

For every pixel x' in $g(x')$

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

- Each pixel at the destination is sampled from the original image
- How does this differ from forward mapping?
- Since $\hat{h}(x')$ is defined for all pixels in $g(x')$, we no longer have holes
- What if pixel comes from between two pixels?



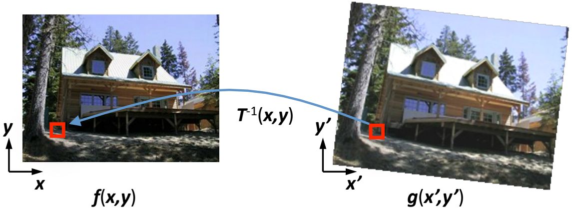
Inverse-Warping

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

For every pixel x' in $g(x')$

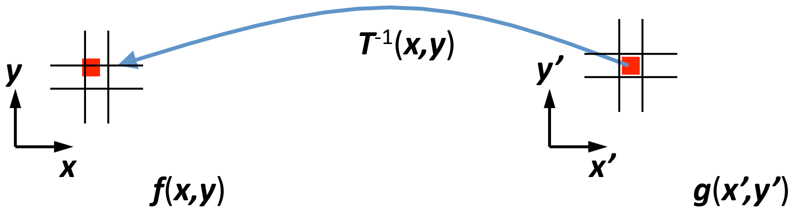
1. Compute the source location $x = \hat{h}(x')$
2. Resample $f(x)$ at location x and copy to $g(x')$

- Each pixel at the destination is sampled from the original image
- How does this differ from forward mapping?
- Since $\hat{h}(x')$ is defined for all pixels in $g(x')$, we no longer have holes
- What if pixel comes from between two pixels?



Inverse-Warping

- What if pixel comes from between two pixels?
- Resampling an image at non-integer locations is a well-studied problem (i.e., image interpolation) high-quality filters that control aliasing can be used



How to computer the inverse-warping?

- Often $\hat{h}(x')$ can simply be computed as the inverse of $h(x)$.
- In other cases, it is preferable to formulate as resampling a source image $f(x)$ given a mapping $x = \hat{h}(x')$ from destination pixels x' to source pixels x.

How to computer the inverse-warping?

- Often $\hat{h}(x')$ can simply be computed as the inverse of $h(x)$.
- In other cases, it is preferable to formulate as resampling a source image $f(x)$ given a mapping $x = \hat{h}(x')$ from destination pixels x' to source pixels x.
- Let's see some examples of the former

How to computer the inverse-warping?

- Often $\hat{h}(x')$ can simply be computed as the inverse of $h(x)$.
- In other cases, it is preferable to formulate as resampling a source image $f(x)$ given a mapping $x = \hat{h}(x')$ from destination pixels x' to source pixels x.
- Let's see some examples of the former
- Lets consider linear transformations (can be represented by a 2D matrix):

$$p' = \mathbf{T}p \qquad \begin{bmatrix} x' \\ y' \end{bmatrix} = \mathbf{T} \begin{bmatrix} x \\ y \end{bmatrix}$$

How to computer the inverse-warping?

- Often $\hat{h}(x')$ can simply be computed as the inverse of $h(x)$.
- In other cases, it is preferable to formulate as resampling a source image $f(x)$ given a mapping $x = \hat{h}(x')$ from destination pixels x' to source pixels x.
- Let's see some examples of the former
- Lets consider linear transformations (can be represented by a 2D matrix):

$$p' = \mathbf{T}p \qquad \begin{bmatrix} x' \\ y' \end{bmatrix} = \mathbf{T} \begin{bmatrix} x \\ y \end{bmatrix}$$

Common linear transformations

- Uniform scaling by s

$$\mathbf{S} = \begin{bmatrix} s & 0 \\ 0 & s \end{bmatrix}$$

(0,0)  (0,0) 

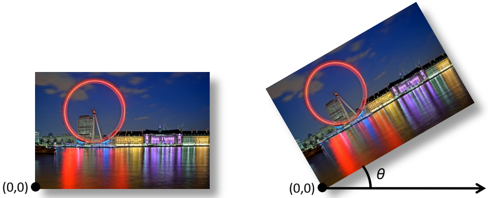
- What's the inverse?

[Source: N. Snavely]

Common linear transformations

- Rotation by an angle θ (about the origin)

$$\mathbf{R} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}$$



- What's the inverse?

$$\mathbf{R}^{-1} = \mathbf{R}^T$$

[Source: N. Snavely]

2×2 Matrices

What types of transformations can be represented with a 2×2 matrix?

- 2D mirror about Y axis?

$$\begin{aligned} x' &= -x \\ y' &= y \end{aligned} \qquad \mathbf{T} = \begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix}$$

- 2D mirror across line y = x?

$$\begin{aligned} x' &= y \\ y' &= x \end{aligned} \qquad \mathbf{T} = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$$

[Source: N. Snavely]

2×2 Matrices

What types of transformations can be represented with a 2×2 matrix?

- 2D mirror about Y axis?

$$\begin{aligned} x' &= -x \\ y' &= y \end{aligned} \qquad \mathbf{T} = \begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix}$$

- 2D mirror across line y = x?

$$\begin{aligned} x' &= y \\ y' &= x \end{aligned} \qquad \mathbf{T} = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$$

[Source: N. Snavely]

2×2 Matrices

What types of transformations can be represented with a 2×2 matrix?

- 2D Translation?

$$\begin{aligned} x' &= x + t_x \\ y' &= y + t_y \end{aligned}$$

- Translation is NOT a linear operation on 2D coordinates

2×2 Matrices

What types of transformations can be represented with a 2×2 matrix?

- 2D Translation?

$$\begin{aligned} x' &= x + t_x \\ y' &= y + t_y \end{aligned}$$

- Translation is NOT a linear operation on 2D coordinates
- What can we do?

[Source: N. Snavely]

2×2 Matrices

What types of transformations can be represented with a 2×2 matrix?

- 2D Translation?

$$\begin{aligned} x' &= x + t_x \\ y' &= y + t_y \end{aligned}$$

- Translation is NOT a linear operation on 2D coordinates
- What can we do?

[Source: N. Snavely]

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

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

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?

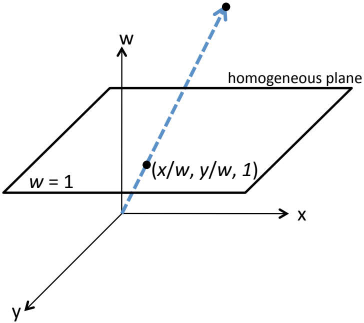
[Source: N. Snavely]

Homogeneous coordinates

Trick: add one more coordinate:

$$(x, y) \Rightarrow \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

homogeneous image coordinates



Converting *from* homogeneous coordinates

$$\begin{bmatrix} x \\ y \\ w \end{bmatrix} \Rightarrow (x/w, y/w)$$

[Source: N. Snavely]

Translation

- Solution: homogeneous coordinates to the rescue

$$\mathbf{T} = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix}$$

Thus we can write

$$\begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} x + t_x \\ y + t_y \\ 1 \end{bmatrix}$$

[Source: N. Snavely]

Affine Transformations

$$\mathbf{T} = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix}$$

any transformation with last row [0 0 1] we call an *affine* transformation

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

[Source: N. Snavely]

Basic Affine Transformations

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Translation

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Scale

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

2D in-plane rotation

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & sh_x & 0 \\ sh_y & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Shear

[Source: N. Snavely]

Affine Transformations

Affine transformations are combinations of

- Linear transformations, and
- Translations

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

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

[Source: N. Snavely]

Is this an affine Tranformation?

[Source: N. Snavely]

What's next?

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

what happens when we mess with this row?

affine transformation

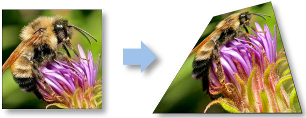
[Source: N. Snavely]

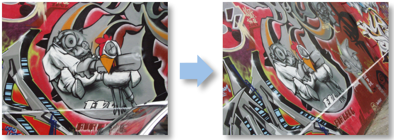
Homography

- Also called **Projective Transformation** or **Planar Perspective Map**

$$\mathbf{H} = \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & 1 \end{bmatrix}$$

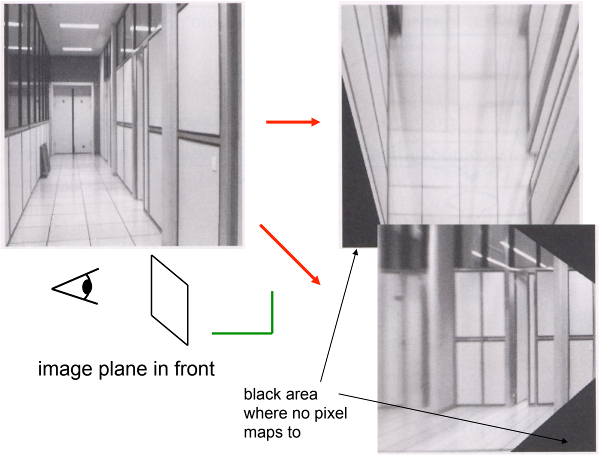
Called a *homography*
(or *planar perspective map*)





[Source: N. Snavely]

Image warping with homographies



[Source: N. Snavely]

Homographies

[Source: N. Snavely]

Projective Transformations

- Affine transformations and Projective warps

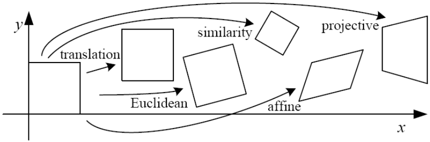
$$\begin{bmatrix} x' \\ y' \\ w' \end{bmatrix} = \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix} \begin{bmatrix} x \\ y \\ w \end{bmatrix}$$

Properties of affine transformations:

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

[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

Homographies

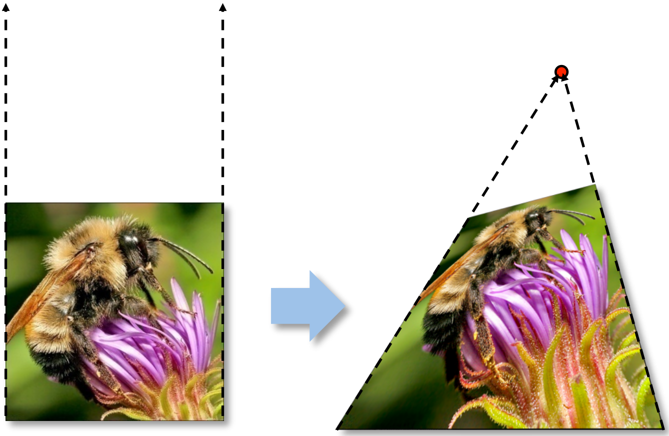
$$\begin{bmatrix} x' \\ y' \\ w' \end{bmatrix} = \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

What happens when the denominator is 0?

$$\sim \begin{bmatrix} \frac{ax+by+c}{gx+hy+1} \\ \frac{dx+ey+f}{gx+hy+1} \\ 1 \end{bmatrix}$$

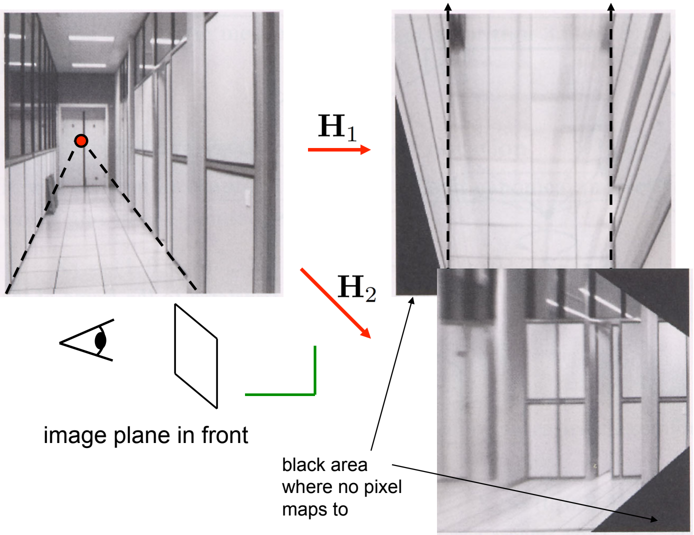
Points at infinity

- Points at infinity become finite i.e., **vanishing points**



[Source: N. Snavely]

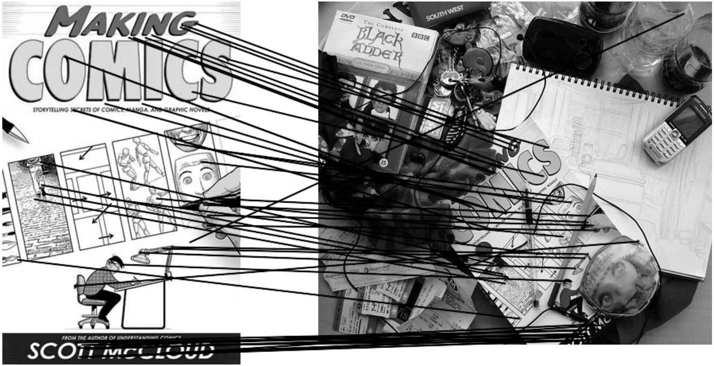
Image warping with homographies



Computing transformations

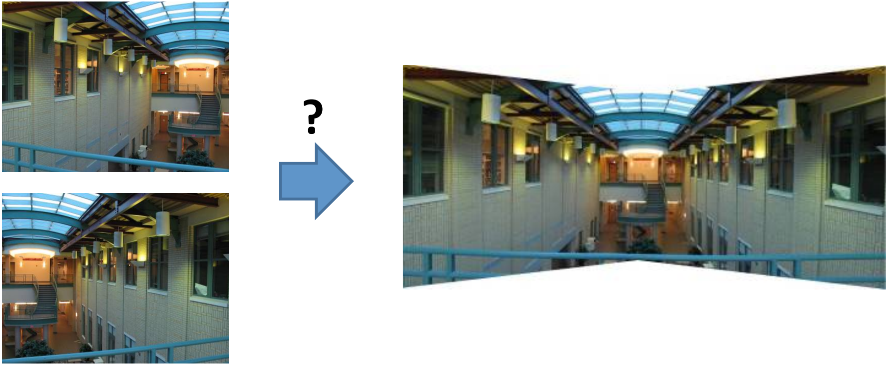
Given a set of matches between images A and B

- How can we compute the transform T from A to B?
- Find transform T that best agrees with the matches



[Source: N. Snavely]

Computing Transformations



[Source: N. Snavely]

Computing Transformations

Can also think of as fitting a "model" to our data

- The model is the transformation of a given type, e.g. a translation, affine xform, homography etc

Computing Transformations

Can also think of as fitting a "model" to our data

- The model is the transformation of a given type, e.g. a translation, affine xform, homography etc
- Fitting the model means solving for the parameters that best explain the observed data

Computing Transformations

Can also think of as fitting a "model" to our data

- The model is the transformation of a given type, e.g. a translation, affine xform, homography etc
- Fitting the model means solving for the parameters that best explain the observed data
- Usually involves minimizing some objective / cost function

[Source: N. Snavely]

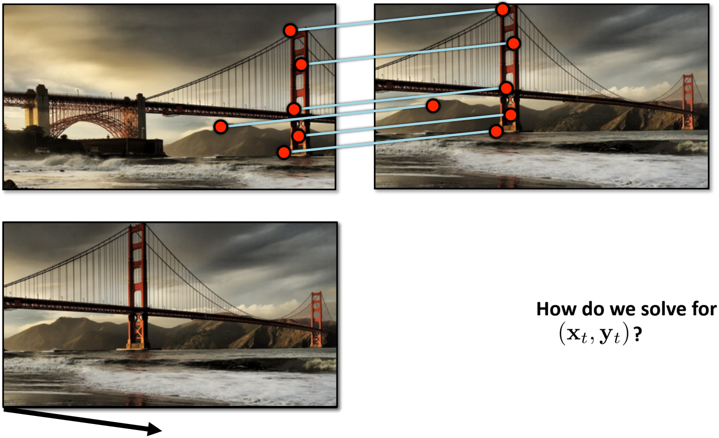
Computing Transformations

Can also think of as fitting a "model" to our data

- The model is the transformation of a given type, e.g. a translation, affine xform, homography etc
- Fitting the model means solving for the parameters that best explain the observed data
- Usually involves minimizing some objective / cost function

[Source: N. Snavely]

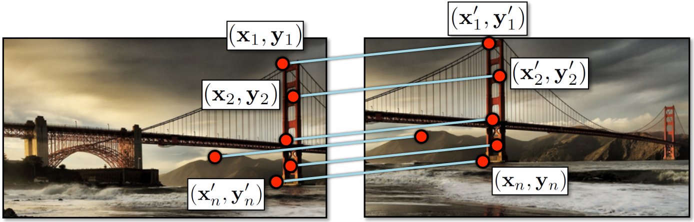
Simple Case: Translations



How do we solve for $(\mathbf{x}_t, \mathbf{y}_t)$ **?**

[Source: N. Snavely]

Simple Case: Translations

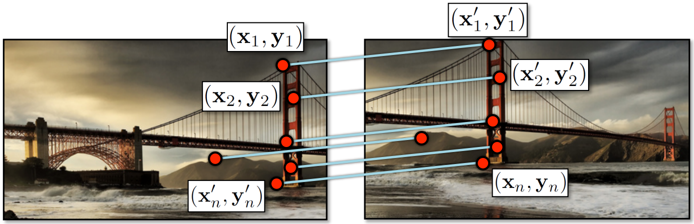


The displacement of match i is $(x'_i - x_i, y'_i - y_i)$. We can thus solve for

$$(x_t, y_t) = (\frac{1}{n}\sum_{i=1}^{n} x'_i - x_i, \frac{1}{n}\sum_{i=1}^{n} y'_i - y_i)$$

[Source: N. Snavely]

Another View

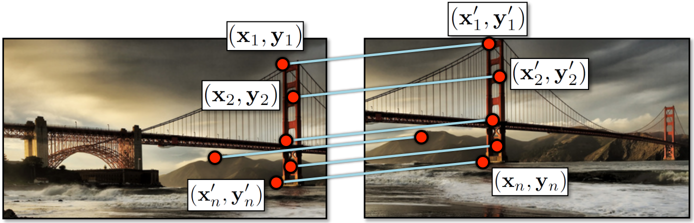


$$
\begin{aligned}
x_i + x_t &= x_i' \\
y_i + y_t &= y_i'
\end{aligned}
$$

System of linear equations

- What are the knowns?

Another View

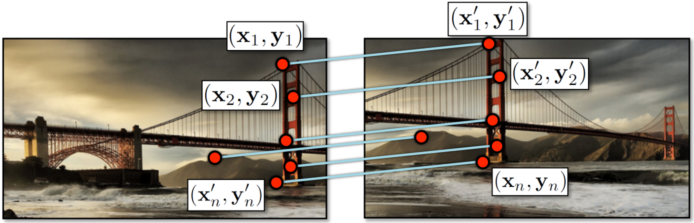


$$\begin{aligned} x_i + x_t &= x'_i \\ y_i + y_t &= y'_i \end{aligned}$$

System of linear equations

- What are the knowns?
- How many unknowns?

Another View



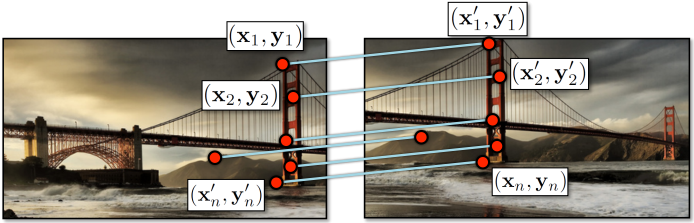
$$\begin{aligned} x_i + x_t &= x'_i \\ y_i + y_t &= y'_i \end{aligned}$$

System of linear equations

- What are the knowns?
- How many unknowns?
- How many equations (per match)?

[Source: N. Snavely]

Another View



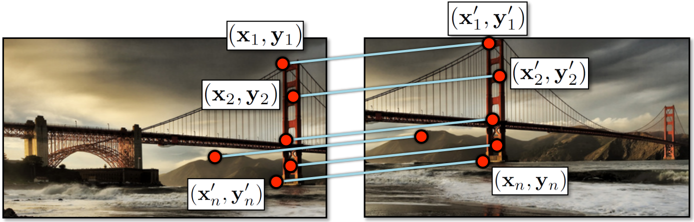
$$\begin{aligned} x_i + x_t &= x'_i \\ y_i + y_t &= y'_i \end{aligned}$$

System of linear equations

- What are the knowns?
- How many unknowns?
- How many equations (per match)?

[Source: N. Snavely]

Another View



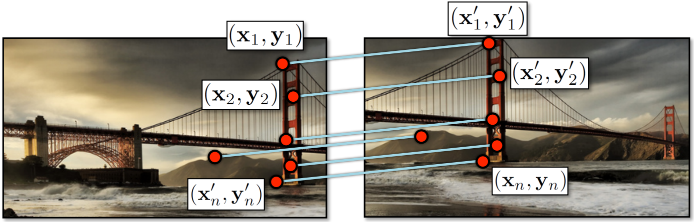
$$
\begin{aligned}
x_i + x_t &= x'_i \\
y_i + y_t &= y'_i
\end{aligned}
$$

Problem: more equations than unknowns

- **Overdetermined** system of equations
- We will find the least squares solution

[Source: N. Snavely]

Another View



$$\begin{aligned} x_i + x_t &= x'_i \\ y_i + y_t &= y'_i \end{aligned}$$

Problem: more equations than unknowns

- **Overdetermined** system of equations
- We will find the least squares solution

[Source: N. Snavely]

Least squares formulation

- For each point (x_i, y_i) we have

$$\begin{aligned} x_i + x_t &= x_i' \\ y_i + y_t &= y_i' \end{aligned}$$

- We define the residuals as

$$\begin{aligned} r_{x_i}(x_t) &= x_i + x_t - x_i' \\ r_{y_i}(y_t) &= y_i + y_t - y_i' \end{aligned}$$

Least squares formulation

- For each point (x_i, y_i) we have

$$\begin{aligned} x_i + x_t &= x_i' \\ y_i + y_t &= y_i' \end{aligned}$$

- We define the residuals as

$$\begin{aligned} r_{x_i}(x_t) &= x_i + x_t - x_i' \\ r_{y_i}(y_t) &= y_i + y_t - y_i' \end{aligned}$$

- **Goal:** minimize sum of squared residuals

$$C(x_t, y_t) = \sum_{i=1}^{n} (r_{x_i}(x_t)^2 + r_{y_i}(y_t)^2)$$

Least squares formulation

- For each point (x_i, y_i) we have

$$\begin{aligned} x_i + x_t &= x'_i \\ y_i + y_t &= y'_i \end{aligned}$$

- We define the residuals as

$$\begin{aligned} r_{x_i}(x_t) &= x_i + x_t - x'_i \\ r_{y_i}(y_t) &= y_i + y_t - y'_i \end{aligned}$$

- **Goal:** minimize sum of squared residuals

$$C(x_t, y_t) = \sum_{i=1}^{n} (r_{x_i}(x_t)^2 + r_{y_i}(y_t)^2)$$

- The solution is called the **least squares** solution

Least squares formulation

- For each point (x_i, y_i) we have

$$\begin{aligned} x_i + x_t &= x'_i \\ y_i + y_t &= y'_i \end{aligned}$$

- We define the residuals as

$$\begin{aligned} r_{x_i}(x_t) &= x_i + x_t - x'_i \\ r_{y_i}(y_t) &= y_i + y_t - y'_i \end{aligned}$$

- **Goal:** minimize sum of squared residuals

$$C(x_t, y_t) = \sum_{i=1}^{n} (r_{x_i}(x_t)^2 + r_{y_i}(y_t)^2)$$

- The solution is called the **least squares** solution
- For translations, is equal to mean displacement

Least squares formulation

- For each point (x_i, y_i) we have

$$\begin{aligned} x_i + x_t &= x_i' \\ y_i + y_t &= y_i' \end{aligned}$$

- We define the residuals as

$$\begin{aligned} r_{x_i}(x_t) &= x_i + x_t - x_i' \\ r_{y_i}(y_t) &= y_i + y_t - y_i' \end{aligned}$$

- **Goal:** minimize sum of squared residuals

$$C(x_t, y_t) = \sum_{i=1}^{n} (r_{x_i}(x_t)^2 + r_{y_i}(y_t)^2)$$

- The solution is called the **least squares** solution
- For translations, is equal to mean displacement
- What do we do?

[Source: N. Snavely]

Least squares formulation

- For each point (x_i, y_i) we have

$$\begin{aligned} x_i + x_t &= x_i' \\ y_i + y_t &= y_i' \end{aligned}$$

- We define the residuals as

$$\begin{aligned} r_{x_i}(x_t) &= x_i + x_t - x_i' \\ r_{y_i}(y_t) &= y_i + y_t - y_i' \end{aligned}$$

- **Goal:** minimize sum of squared residuals

$$C(x_t, y_t) = \sum_{i=1}^{n} (r_{x_i}(x_t)^2 + r_{y_i}(y_t)^2)$$

- The solution is called the **least squares** solution
- For translations, is equal to mean displacement
- What do we do?

[Source: N. Snavely]

Matrix Formulation

- We can also write as a matrix equation

$$\begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 1 & 0 \\ 0 & 1 \\ & \vdots \\ 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x_t \\ y_t \end{bmatrix} = \begin{bmatrix} x'_1 - x_1 \\ y'_1 - y_1 \\ x'_2 - x_2 \\ y'_2 - y_2 \\ \vdots \\ x'_n - x_n \\ y'_n - y_n \end{bmatrix}$$

$$\underset{2n \times 2}{\mathbf{A}} \quad \underset{2 \times 1}{\mathbf{t}} = \underset{2n \times 1}{\mathbf{b}}$$

Least Squares

$$\mathbf{At} = \mathbf{b}$$

- We want to find the optimal $\mathbf{t}$ by

$$\min_{\mathbf{t}} ||\mathbf{At} - \mathbf{b}||_2^2$$

Least Squares

$$\mathbf{A}\mathbf{t} = \mathbf{b}$$

- We want to find the optimal $\mathbf{t}$ by

$$\min_{\mathbf{t}} ||\mathbf{A}\mathbf{t} - \mathbf{b}||_2^2$$

- We can write

$$||\mathbf{A}\mathbf{t} - \mathbf{b}||_2^2 = \mathbf{t}^T(\mathbf{A}^T\mathbf{A})\mathbf{t} - 2\mathbf{t}^T(\mathbf{A}^T\mathbf{b}) + ||\mathbf{b}||_2^2$$

Least Squares

$$\mathbf{A}\mathbf{t} = \mathbf{b}$$

- We want to find the optimal $\mathbf{t}$ by

$$\min_{\mathbf{t}} ||\mathbf{A}\mathbf{t} - \mathbf{b}||_2^2$$

- We can write

$$||\mathbf{A}\mathbf{t} - \mathbf{b}||_2^2 = \mathbf{t}^T(\mathbf{A}^T\mathbf{A})\mathbf{t} - 2\mathbf{t}^T(\mathbf{A}^T\mathbf{b}) + ||\mathbf{b}||_2^2$$

- To solve, form the **normal equations**

$$(\mathbf{A}^T\mathbf{A})\mathbf{t} = \mathbf{A}^T\mathbf{b}$$

Least Squares

$$\mathbf{At} = \mathbf{b}$$

- We want to find the optimal $\mathbf{t}$ by

$$\min_{\mathbf{t}} ||\mathbf{At} - \mathbf{b}||_2^2$$

- We can write

$$||\mathbf{At} - \mathbf{b}||_2^2 = \mathbf{t}^T(\mathbf{A}^T\mathbf{A})\mathbf{t} - 2\mathbf{t}^T(\mathbf{A}^T\mathbf{b}) + ||\mathbf{b}||_2^2$$

- To solve, form the **normal equations**

$$(\mathbf{A}^T\mathbf{A})\mathbf{t} = \mathbf{A}^T\mathbf{b}$$

- and compute

$$\mathbf{t} = (\mathbf{A}^T\mathbf{A})^{-1}\mathbf{Ab}$$

Least Squares

$$\mathbf{At} = \mathbf{b}$$

- We want to find the optimal $\mathbf{t}$ by

$$\min_{\mathbf{t}} ||\mathbf{At} - \mathbf{b}||_2^2$$

- We can write

$$||\mathbf{At} - \mathbf{b}||_2^2 = \mathbf{t}^T(\mathbf{A}^T\mathbf{A})\mathbf{t} - 2\mathbf{t}^T(\mathbf{A}^T\mathbf{b}) + ||\mathbf{b}||_2^2$$

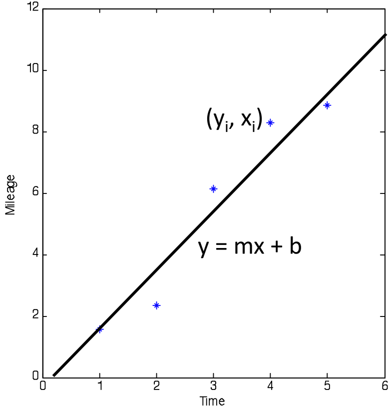
- To solve, form the **normal equations**

$$(\mathbf{A}^T\mathbf{A})\mathbf{t} = \mathbf{A}^T\mathbf{b}$$

- and compute

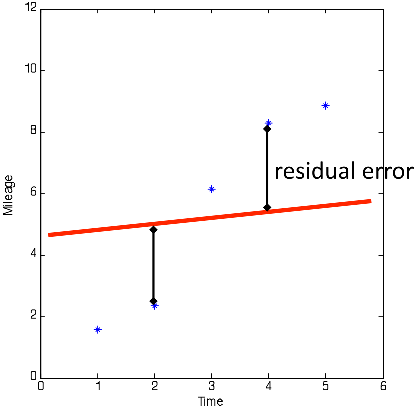
$$\mathbf{t} = (\mathbf{A}^T\mathbf{A})^{-1}\mathbf{Ab}$$

Least squares: generalized linear regression



[Source: N. Snavely]

Linear regression



$$Cost(m, b) = \sum_{i=1}^{n} |y_i - (mx_i + b)|^2$$

[Source: N. Snavely]

Linear regression

$$\begin{bmatrix} x_1 & 1 \\ x_2 & 1 \\ \vdots & \\ x_n & 1 \end{bmatrix} \begin{bmatrix} m \\ b \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}$$

[Source: N. Snavely]

Affine Transformations

When we are dealing with an affine transformation

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

- How many unknowns?

Affine Transformations

When we are dealing with an affine transformation

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

- How many unknowns?
- How many equations per match?

Affine Transformations

When we are dealing with an affine transformation

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

- How many unknowns?
- How many equations per match?
- How many matches do we need?

Affine Transformations

When we are dealing with an affine transformation

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

- How many unknowns?
- How many equations per match?
- How many matches do we need?
- Why to use more?

[Source: N. Snavely]

Affine Transformations

When we are dealing with an affine transformation

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

- How many unknowns?
- How many equations per match?
- How many matches do we need?
- Why to use more?

[Source: N. Snavely]

Affine Transformation Cost Function

- We can write the residuals as

$$\begin{aligned} r_{x_i}(a,b,c,d,e,f) &= (ax_i + by_i + c) - x'_i \\ r_{y_i}(a,b,c,d,e,f) &= (dx_i + ey_i + f) - y'_i \end{aligned}$$

- Cost function

$$C(a,b,c,d,e,f) = \sum_{i=1}^{N} \left(r_{x_i}(a,b,c,d,e,f)^2 + r_{y_i}(a,b,c,d,e,f)^2 \right)$$

Affine Transformation Cost Function

- We can write the residuals as

$$\begin{aligned} r_{x_i}(a,b,c,d,e,f) &= (ax_i + by_i + c) - x_i' \\ r_{y_i}(a,b,c,d,e,f) &= (dx_i + ey_i + f) - y_i' \end{aligned}$$

- Cost function

$$C(a,b,c,d,e,f) = \sum_{i=1}^{N} \left(r_{x_i}(a,b,c,d,e,f)^2 + r_{y_i}(a,b,c,d,e,f)^2 \right)$$

- And in matrix form ...

[Source: N. Snavely]

Affine Transformation Cost Function

- We can write the residuals as

$$\begin{aligned} r_{x_i}(a,b,c,d,e,f) &= (ax_i + by_i + c) - x_i' \\ r_{y_i}(a,b,c,d,e,f) &= (dx_i + ey_i + f) - y_i' \end{aligned}$$

- Cost function

$$C(a,b,c,d,e,f) = \sum_{i=1}^{N} \left(r_{x_i}(a,b,c,d,e,f)^2 + r_{y_i}(a,b,c,d,e,f)^2 \right)$$

- And in matrix form ...

[Source: N. Snavely]

Matrix form

$$
\begin{bmatrix}
x_1 & y_1 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & x_1 & y_1 & 1 \\
x_2 & y_2 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & x_2 & y_2 & 1 \\
 & & & \vdots & & \\
x_n & y_n & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & x_n & y_n & 1
\end{bmatrix}
\begin{bmatrix} a \\ b \\ c \\ d \\ e \\ f \end{bmatrix}
=
\begin{bmatrix} x'_1 \\ y'_1 \\ x'_2 \\ y'_2 \\ \vdots \\ x'_n \\ y'_n \end{bmatrix}
$$

$$
\underset{2n \times 6}{\mathbf{A}} \quad \underset{6 \times 1}{\mathbf{t}} = \underset{2n \times 1}{\mathbf{b}}
$$

[Source: N. Snavely]

Next class ... more sophisticated matching