Computing transformations

Prof. Noah Snavely

CS1114

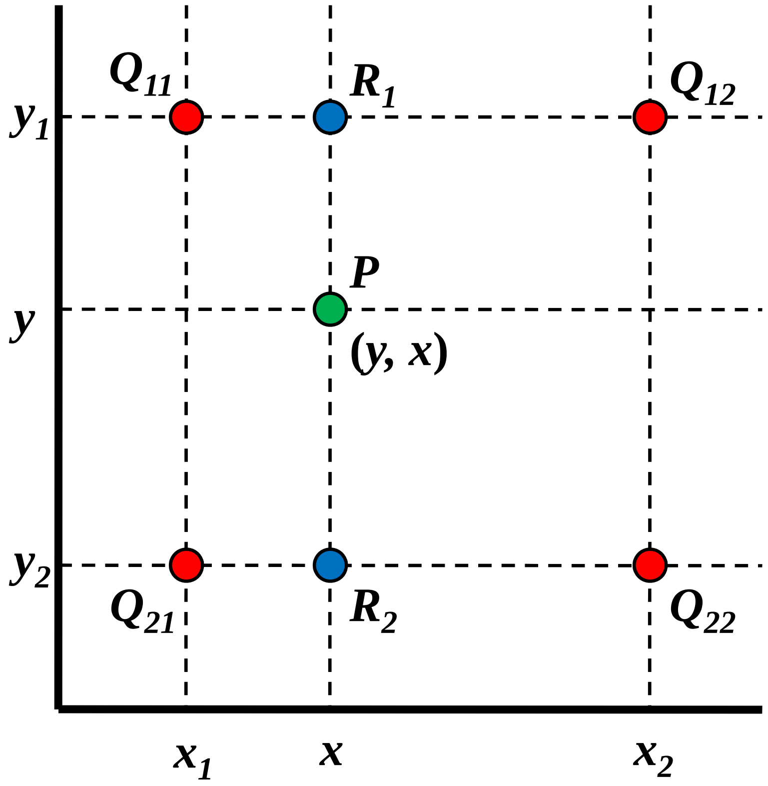
http://cs1114.cs.cornell.edu

Cornell University
Computer Science

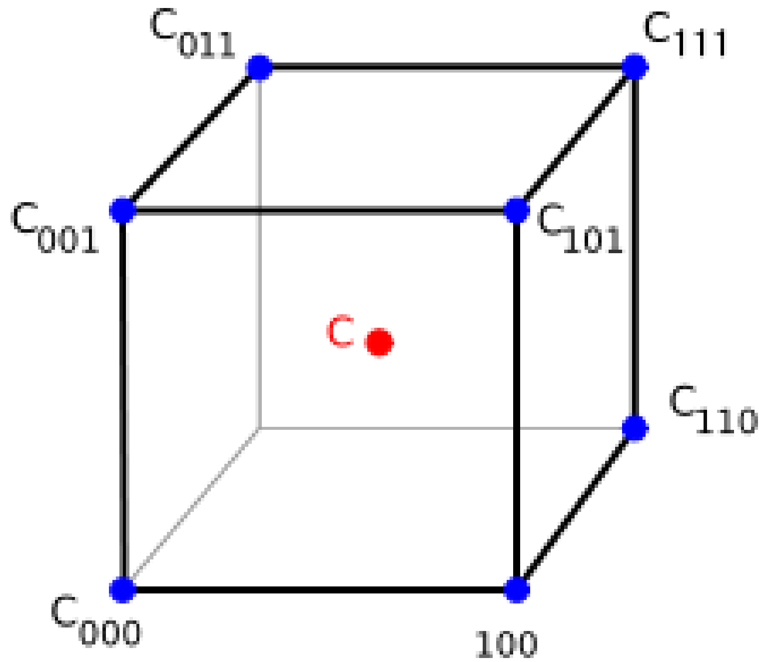
Administrivia

- A5 Part 1 due on Friday, A5 Part 2 out soon
- Prelim 2 next week, 4/7 (in class)
  - Covers everything since Prelim 1
  - Review session next Monday (time TBA)

Bilinear interpolation



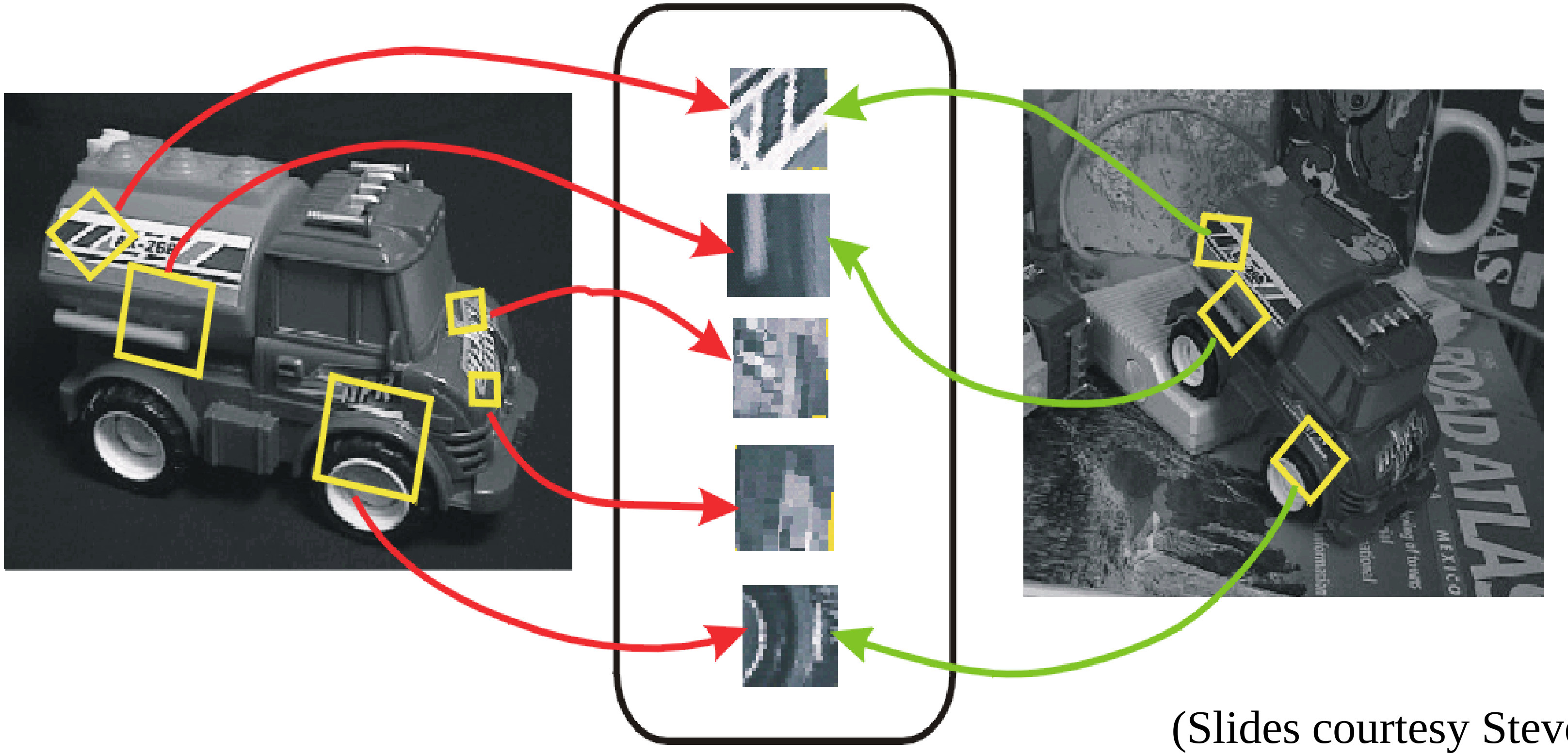
*Tri*linear interpolation



- How do we find the value of the function at C?

Invariant local features

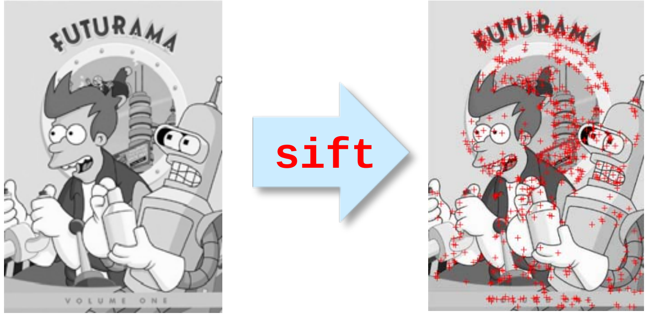
- Find features that are invariant to transformations
  - geometric invariance: translation, rotation, scale
  - photometric invariance: brightness, exposure, ...



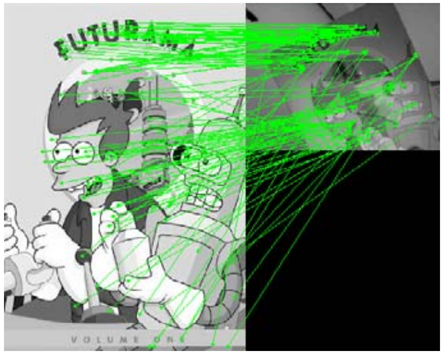
(Slides courtesy Steve Seitz)

Object matching in three steps

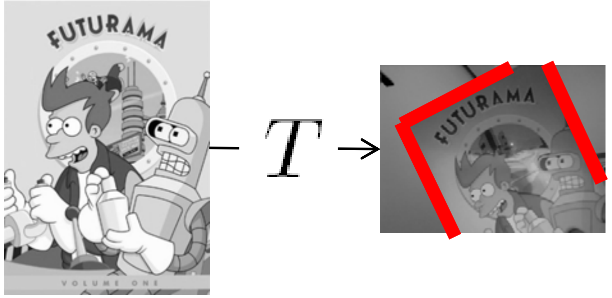
1. Detect features in the template and search images



2. Match features: find “similar-looking” features in the two images



3. Find a transformation T that explains the movement of the matched features



Step 1: Detecting SIFT features

- SIFT gives us a set of feature **frames** and **descriptors** for an image


```
img = imread('futurama.png');
[frames, descs] = sift(img);
```

Step 2: Matching SIFT features

- Answer: for each feature in image 1, find the feature with the *closest descriptor* in image 2

- Called *nearest neighbor* matching

Matching SIFT features

- Output of the matching step:

  Pairs of matching points

$$[\ x_1\ y_1\ ] \rightarrow [\ x_1'\ y_1'\ ]$$
$$[\ x_2\ y_2\ ] \rightarrow [\ x_2'\ y_2'\ ]$$
$$[\ x_3\ y_3\ ] \rightarrow [\ x_3'\ y_3'\ ]$$
$$\ldots$$
$$[\ x_k\ y_k\ ] \rightarrow [\ x_k'\ y_k'\ ]$$

Step 3: Find the transformation

- How do we draw a box around the template image in the search image?

- Key idea: there is a transformation that maps template → search image!

Image transformations

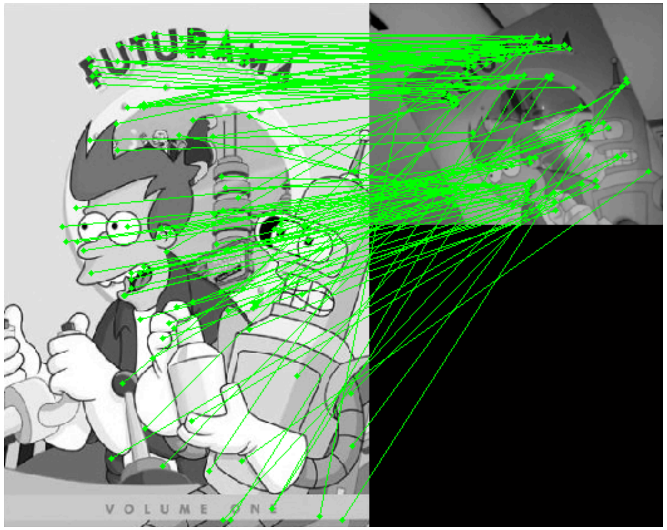
- 2D affine transformation

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

$$\begin{bmatrix} a & b & c \\ d & e & f \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} ax + by + c \\ dx + ey + f \\ 1 \end{bmatrix}$$

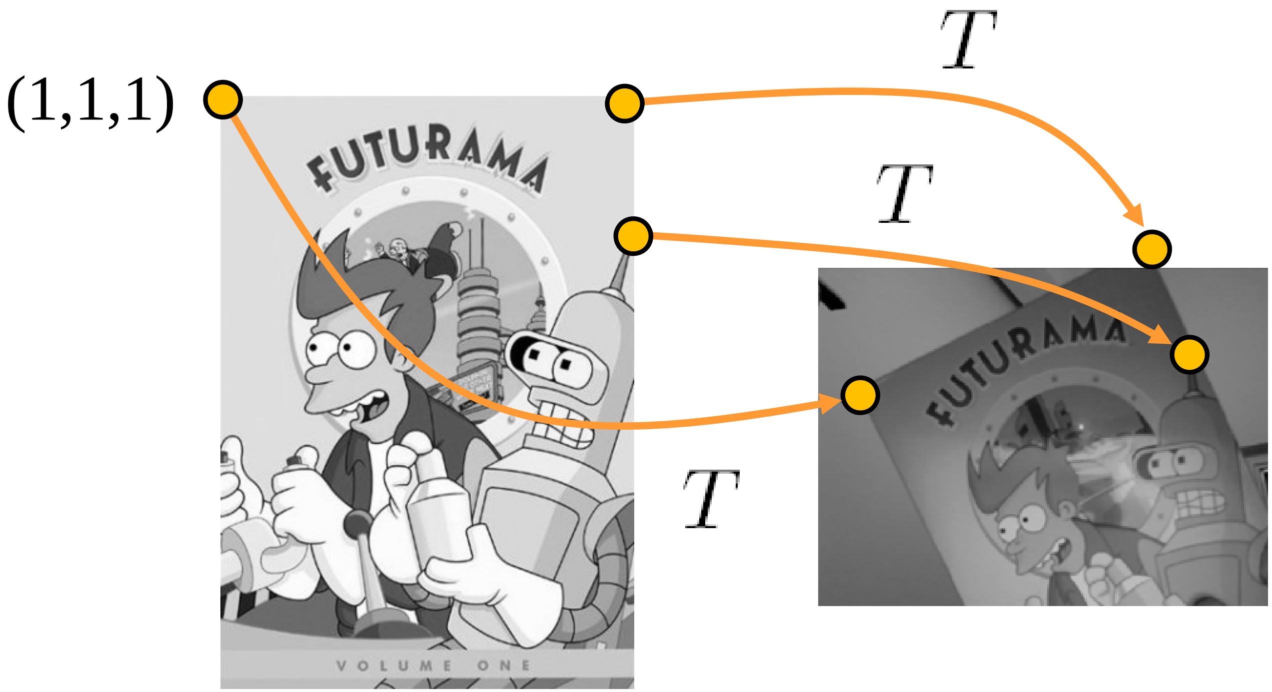
Solving for image transformations

- Given a set of matching points between image 1 and image 2...



... can we solve for an affine transformation *T* mapping 1 to 2?

Solving for image transformations



- *T* maps points in image 1 to the corresponding point in image 2

How do we find *T* ?

- We already have a bunch of point matches

$$[\ x_1\ y_1\ ] \rightarrow [\ x_1'\ y_1'\ ]$$
$$[\ x_2\ y_2\ ] \rightarrow [\ x_2'\ y_2'\ ]$$
$$[\ x_3\ y_3\ ] \rightarrow [\ x_3'\ y_3'\ ]$$
$$\ldots$$
$$[\ x_k\ y_k\ ] \rightarrow [\ x_k'\ y_k'\ ]$$

- Solution: Find the *T* that best agrees with these known matches
- This problem is a form of (linear) regression

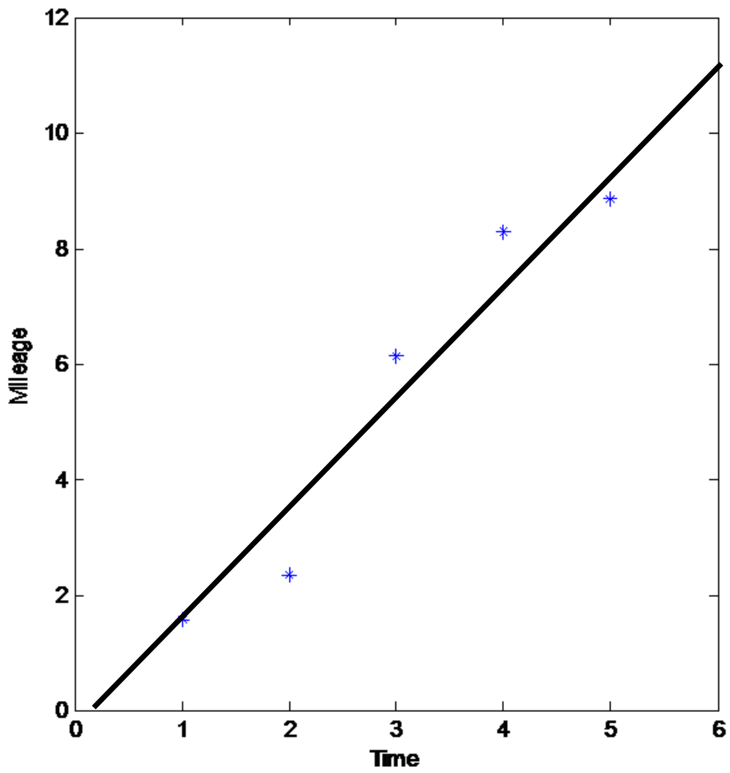
An Algorithm: Take 1

1. To find *T*, randomly guess a, b, c, d, e, f, check how well *T* matches the data
2. If it matches well, return *T*
3. Otherwise, go to step 1

- The “snailsort” method
- We can do much better

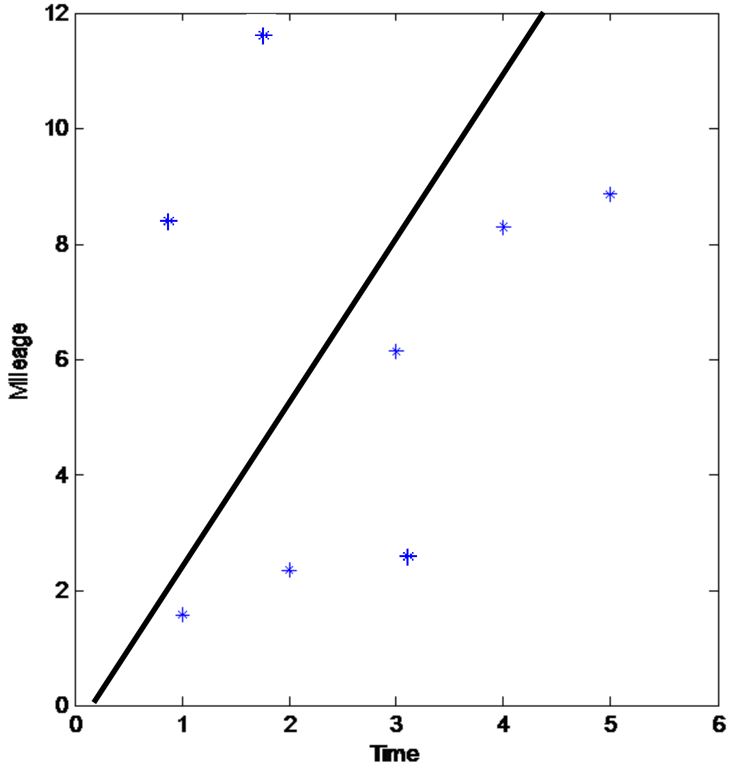
Linear regression

- Simplest case: fitting a line



Linear regression

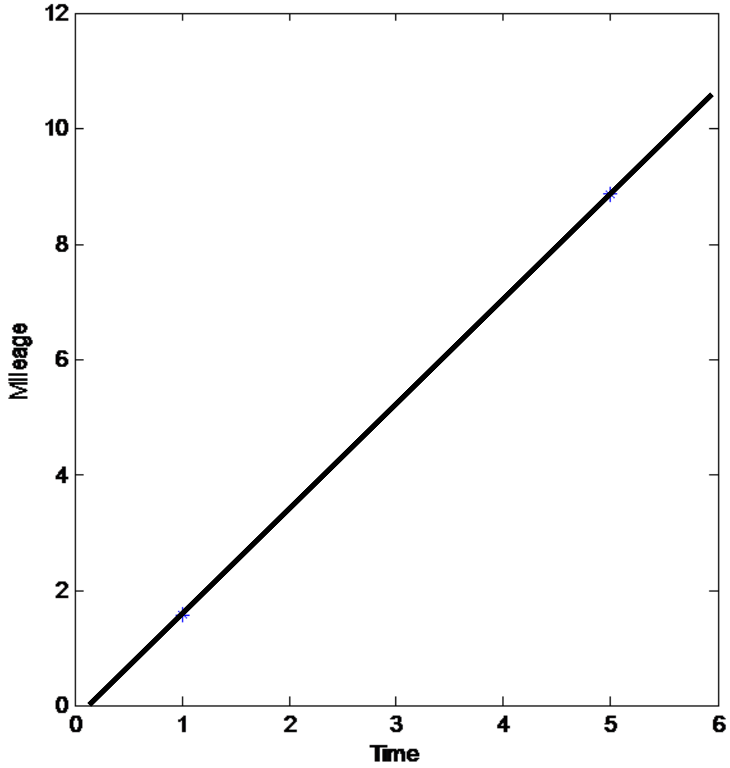
- But what happens here?



What does this remind you of ?

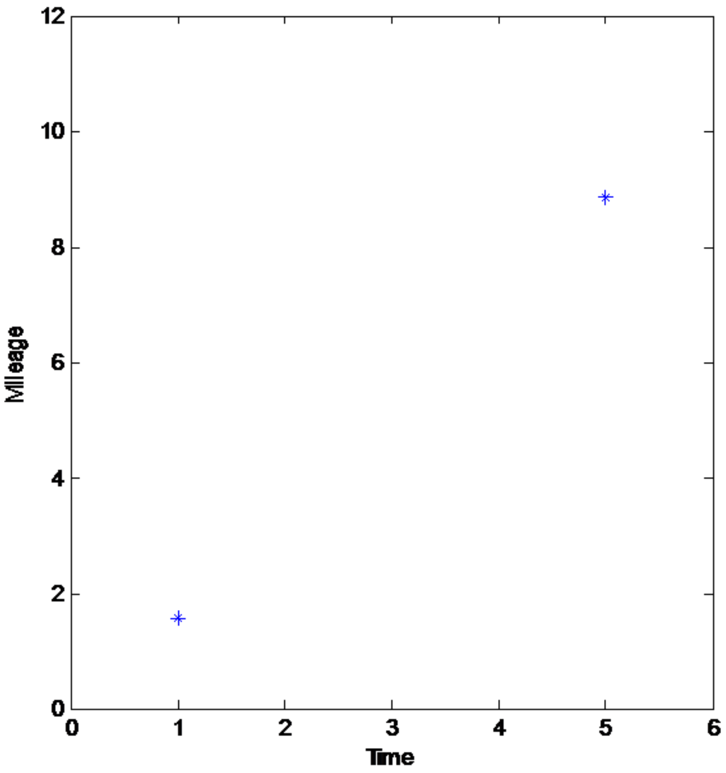
Linear regression

- Simplest case: just 2 points



Linear regression

- Simplest: just 2 points



- Want to find a line

  $y = mx + b$
- $x_1 \rightarrow y_1$, $x_2 \rightarrow y_2$
- This forms a linear system:

  $y_1 = mx_1 + b$

  $y_2 = mx_2 + b$
- x's, y's are knowns
- m, b are unknown
- Very easy to solve

Multi-variable linear regression

- What about 2D affine transformations?
  - maps a 2D point to another 2D point

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

- We have a set of matches

$[x_1 \; y_1] \rightarrow [x_1' \; y_1']$

$[x_2 \; y_2] \rightarrow [x_2' \; y_2']$

$[x_3 \; y_3] \rightarrow [x_3' \; y_3']$

...

$[x_4 \; y_4] \rightarrow [x_4' \; y_4']$

Multi-variable linear regression

- Consider just one match

$[\ x_1\ y_1\ ] \rightarrow [\ x_1'\ y_1'\ ]$

$$\begin{bmatrix} a & b & c \\ d & e & f \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ y_1 \\ 1 \end{bmatrix} = \begin{bmatrix} x_1' \\ y_1' \\ 1 \end{bmatrix}$$

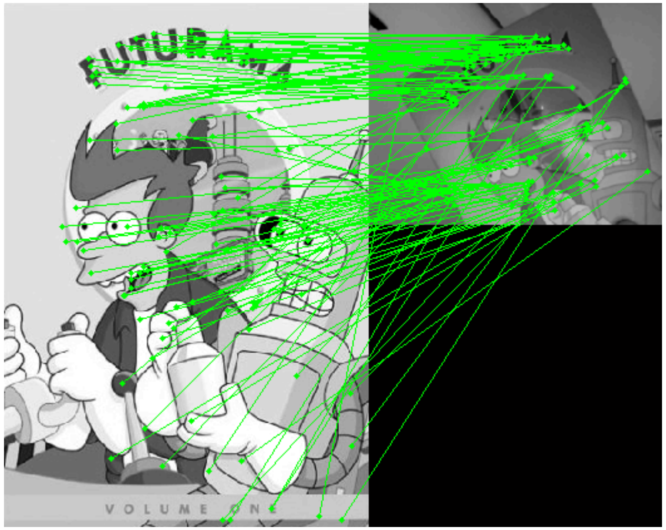
$$ax_1 + by_1 + c = x_1'$$
$$dx_1 + ey_1 + f = y_1'$$

- 2 equations, 6 unknowns → we need 3 matches

Finding an affine transform

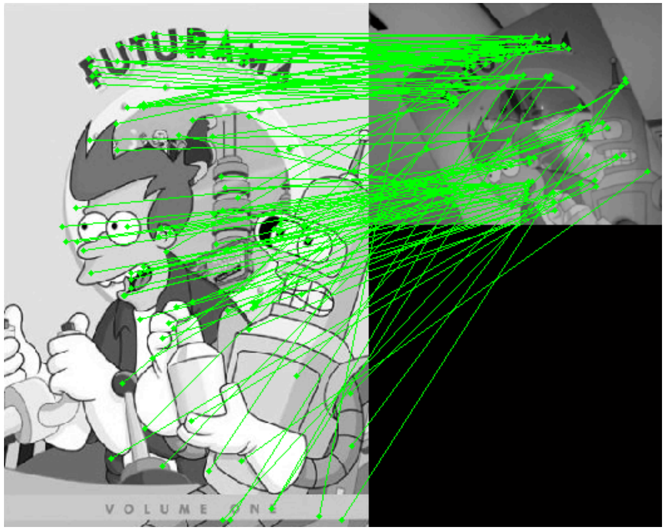
- This is just a bigger linear system, still (relatively) easy to solve
- Really just two linear systems with 3 equations each (one for a,b,c, the other for d,e,f)
- We'll figure out how to solve this in a minute

An Algorithm: Take 2



- We have many more than three matches
- Some are correct, many are wrong
- Idea 2: select three matches at random, compute *T*

An Algorithm: Take 2



- Better then randomly guessing a,b,c,d,e,f
- What could go wrong?

Robustness

- Suppose 1/3 of the matches are wrong
- We select three at random
- The probability of *at least one* selected match being wrong is ?
- If we get just one match wrong, the transformation could be wildly off
- (The Arnold Schwarzenegger problem)

- How do we fix this?

Fixing the problem

- First observation:
- There is a way to test how good the transformation we get is (how?)

Testing goodness

- A good transformation will agree with most of the matches
- A bad transformation will disagree with most of the matches
- How can we tell if a match agrees with the transformation *T*?

$$[\ x_1\ y_1\ ] \rightarrow [\ x_1'\ y_1'\ ]$$

- Compute the distance between

$$T \begin{bmatrix} x_1 \\ y_1 \\ 1 \end{bmatrix} \quad \text{and} \quad \begin{bmatrix} x_1' \\ y_1' \\ 1 \end{bmatrix}$$

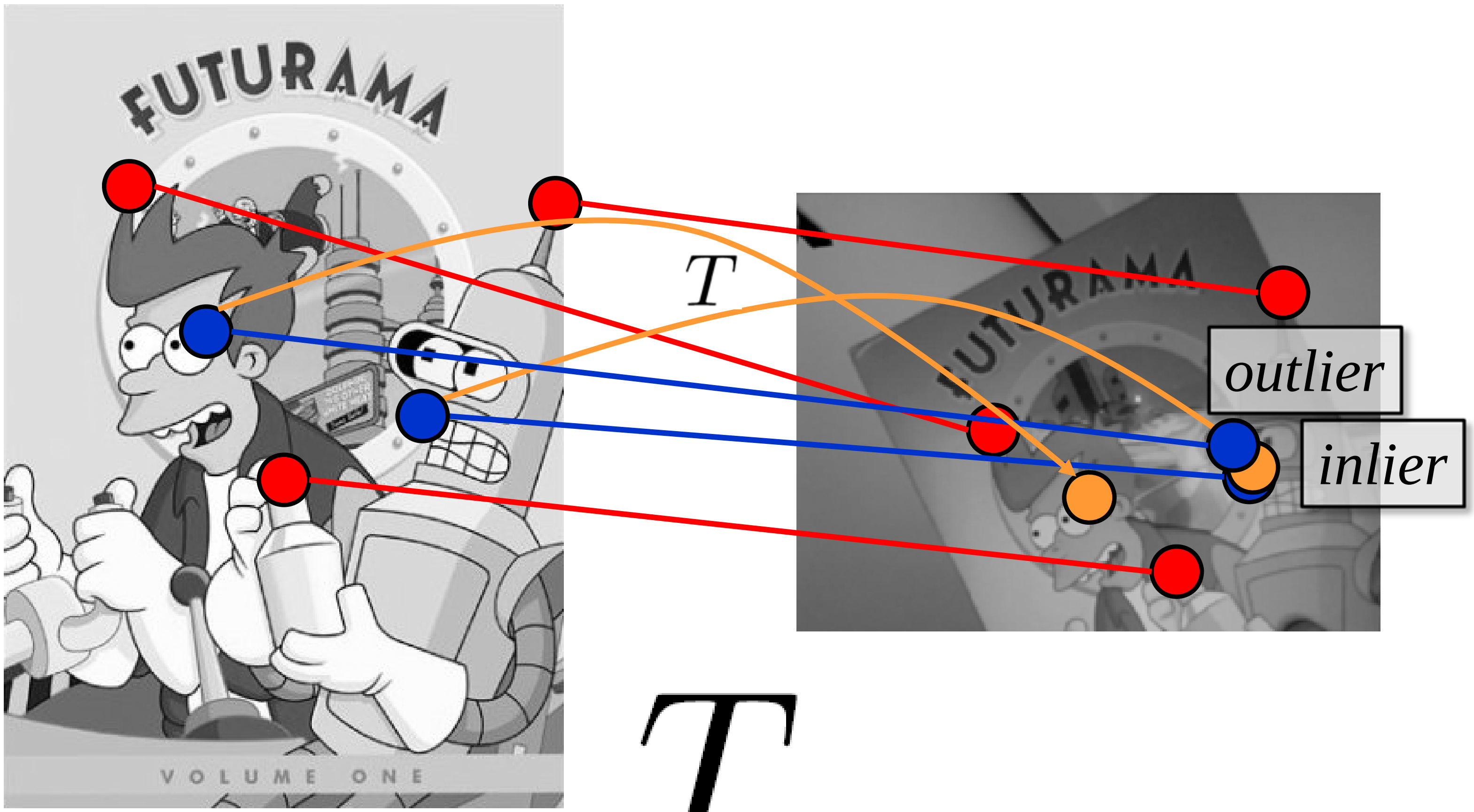
Testing goodness

- Find the distance between

$$T \begin{bmatrix} x_1 \\ y_1 \\ 1 \end{bmatrix} \quad \text{and} \quad \begin{bmatrix} x'_1 \\ y'_1 \\ 1 \end{bmatrix}$$

- If the distance is small, we call this match an *inlier* to *T*
- If the distance is large, it's an *outlier* to *T*
- For a correct match and transformation, this distance will be close to (but not exactly) zero
- For an incorrect match or transformation, this distance will probably be large

Testing goodness



$$T$$

Testing goodness

```
% define a threshold
thresh = 5.0;  % 5 pixels

num_agree = 0;
diff = T * [x1 y1 1]’ – [x1p y1p 1]’;
if norm(diff) < thresh
    num_agree = num_agree + 1;
```

Finding *T*, third attempt

1. Select three points at random
2. Solve for the affine transformation *T*
3. Count the number of inlier matches to *T*
4. If *T* is has the highest number of inliers so far, save it
5. Repeat for N rounds, return the best *T*

Testing goodness

- This algorithm is called RANSAC (RANdom SAmple Consensus)
- Used in an amazing number of computer vision algorithms
- Requires two parameters:
  - The agreement threshold
  - The number of rounds (how many do we need?)

How do we solve for T?

- Given three matches, we have a linear system with six equations:

$[\ x_1\ y_1\ ] \rightarrow [\ x_1'\ y_1'\ ]$

$$ax_1 + by_1 + c = x_1'$$
$$dx_1 + ey_1 + f = y_1'$$

$[\ x_2\ y_2\ ] \rightarrow [\ x_2'\ y_2'\ ]$

$$ax_2 + by_2 + c = x_2'$$
$$dx_2 + ey_2 + f = y_2'$$

$[\ x_3\ y_3\ ] \rightarrow [\ x_3'\ y_3'\ ]$

$$ax_3 + by_3 + c = x_3'$$
$$dx_3 + ey_3 + f = y_3'$$

Two 3x3 linear systems

$$ax_1 + by_1 + c = x_1'$$
$$ax_2 + by_2 + c = x_2'$$
$$ax_3 + by_3 + c = x_3'$$

$$dx_1 + ey_1 + f = y_1'$$
$$dx_2 + ey_2 + f = y_2'$$
$$dx_3 + ey_3 + f = y_3'$$

Solving a 3x3 system

$$ax_1 + by_1 + c = x_1'$$
$$ax_2 + by_2 + c = x_2'$$
$$ax_3 + by_3 + c = x_3'$$

- We can write this in matrix form:

$$\begin{bmatrix} x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \\ x_3 & y_3 & 1 \end{bmatrix} \begin{bmatrix} a \\ b \\ c \end{bmatrix} = \begin{bmatrix} x'_1 \\ x'_2 \\ x'_3 \end{bmatrix}$$

- Now what?

Putting it all together

1. Select three points at random
2. Solve for the affine transformation *T*
3. Count the number of inlier matches to *T*
4. If *T* is has the highest number of inliers so far, save it
5. Repeat for N rounds, return the best *T*

Finding the object boundary