Object recognition

Prof. Graeme Bailey

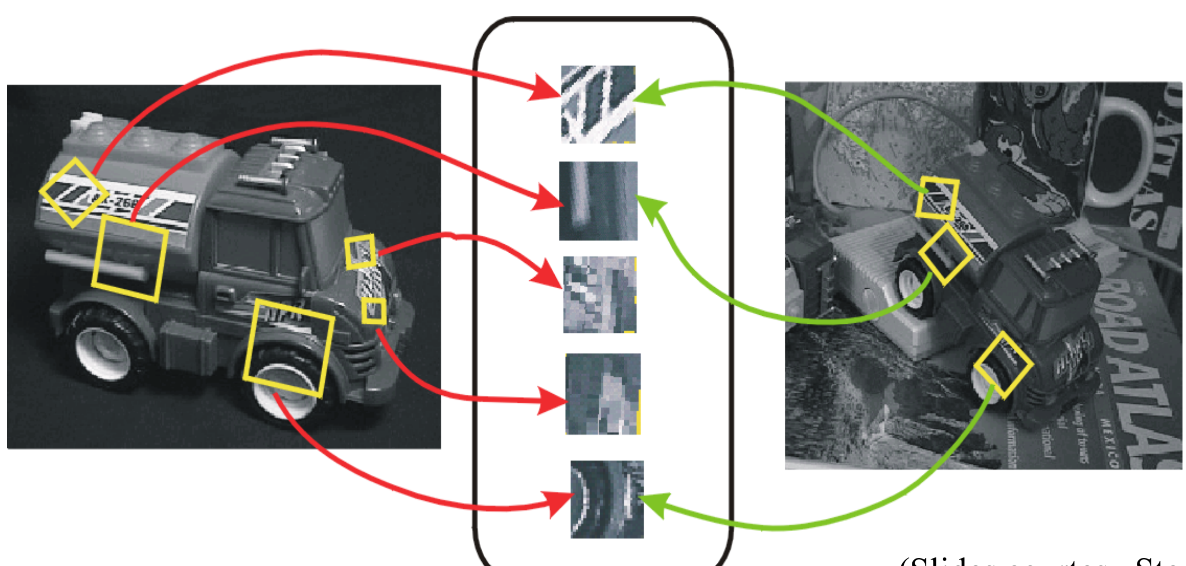
http://cs1114.cs.cornell.edu

(notes modified from Noah Snavely, Spring 2009)

Cornell University
Computer Science

Invariant local features

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



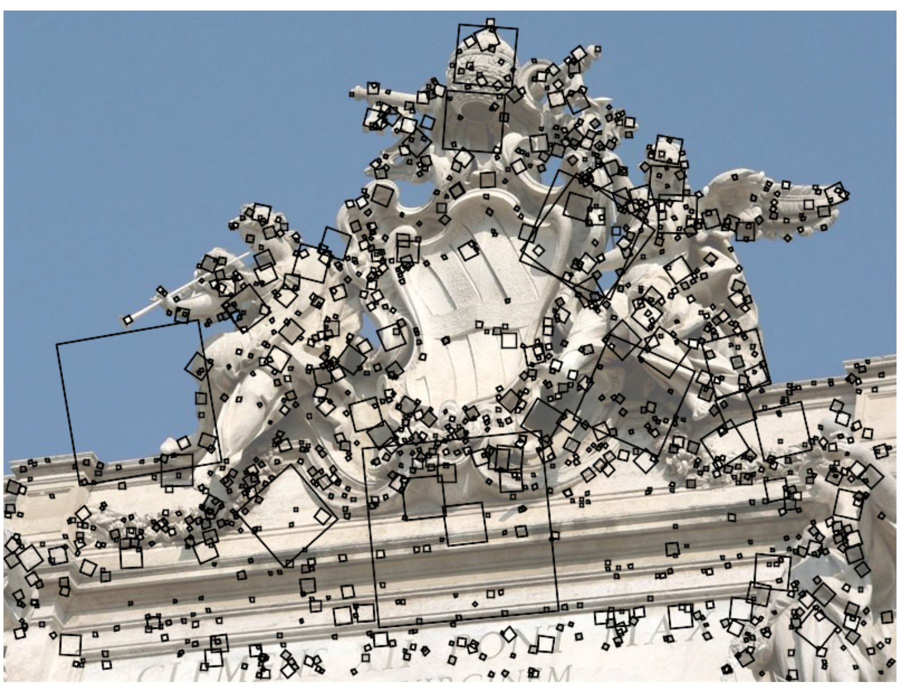
(Slides courtesy Steve Seitz)

Why local features?

- Locality
  - features are local, so robust to occlusion and clutter
- Distinctiveness:
  - can differentiate a large database of objects
- Quantity
  - hundreds or thousands in a single image
- Efficiency
  - real-time performance achievable

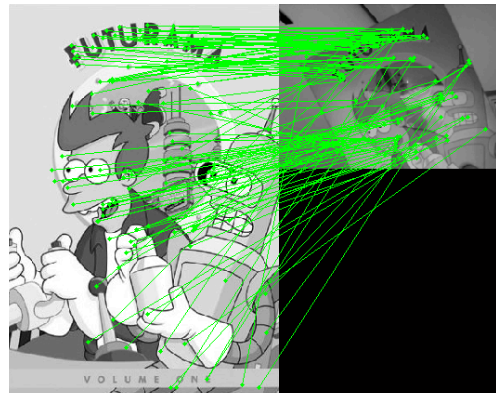
SIFT Features

- **S**cale-**I**nvariant **F**eature **T**ransform



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?

Image transformations

- What about a general homogeneous transformation?

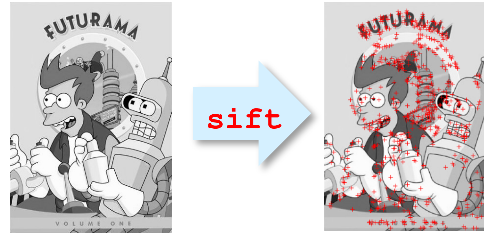
$$T = \begin{bmatrix} a & b & c \\ d & e & f \\ 0 & 0 & 1 \end{bmatrix}$$

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

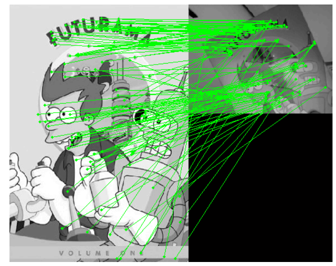
- Called a 2D *affine* transformation

Object matching in three steps

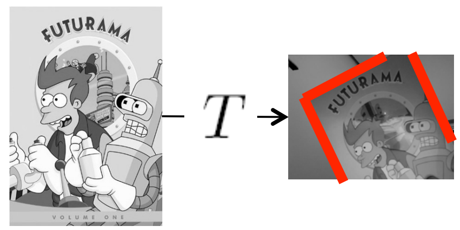
1. Detect features in the template and search images



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

Step 2: Matching SIFT features

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

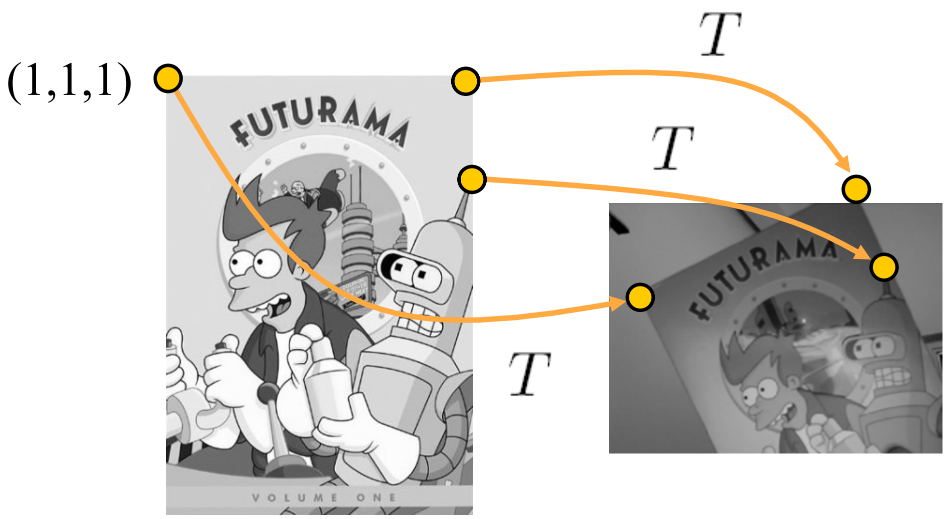
- Called *nearest neighbor* matching

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!

Solving for image transformations



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

How do we find *T* ?

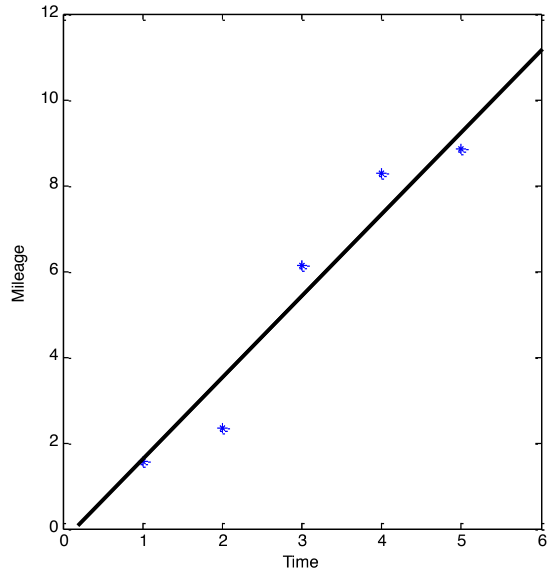
- We already have a bunch of point matches

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

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

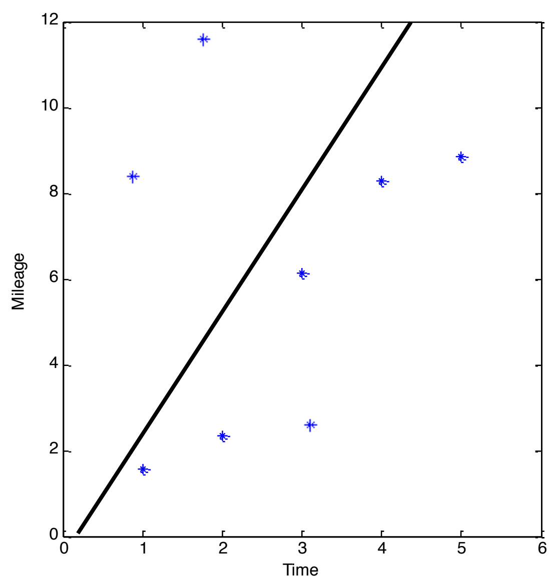
Linear regression

- Simplest case: fitting a line



Linear regression

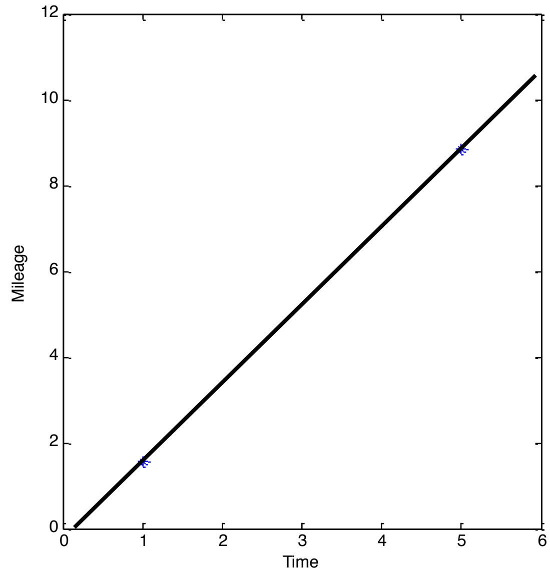
- But what happens here?



What does this remind you of ?

Linear regression

- Simplest case: just 2 points



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

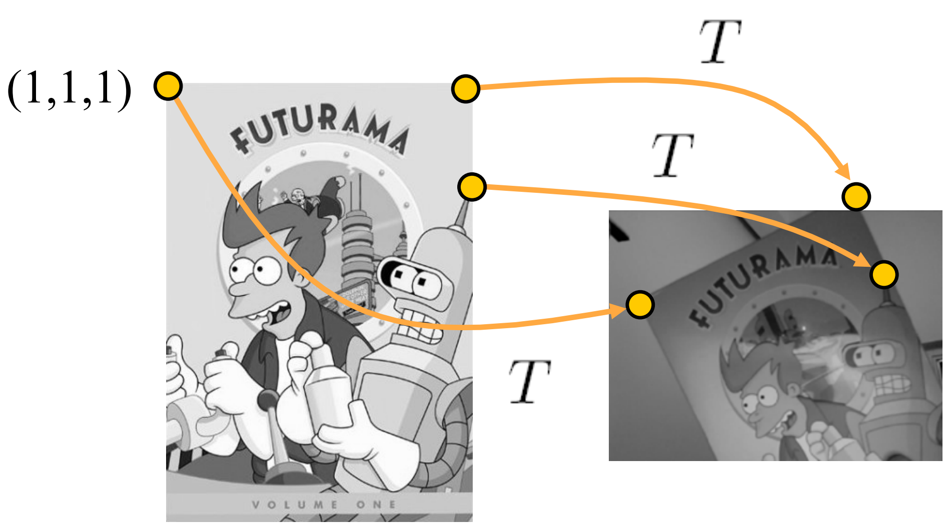
$$[\ x_1\ y_1\ ] \rightarrow [\ x_1'\ y_1'\ ]$$

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

$$\mathbf{a}x_1 + \mathbf{b}y_1 + \mathbf{c} = x_1'$$
$$\mathbf{d}x_1 + \mathbf{e}y_1 + \mathbf{f} = y_1'$$

- 2 equations, 6 unknowns → we need 3 matches

Solving for image transformations



- *T* is then determined by the maps of 3 points in image 1 to the corresponding points in image 2

How do we solve for T?

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

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

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

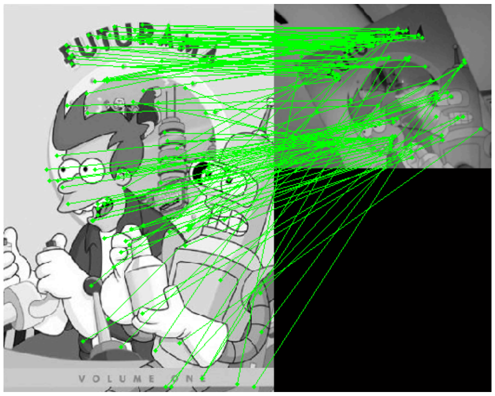
$[\ x_2\ y_2\ ] \rightarrow [\ x_2'\ \ y_2'\ ]$

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

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

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

An Algorithm: Take 1



- We have many more than three matches
- Some are correct, many are wrong
- Idea: select three matches at random, compute *T*
- How can we select a good *T* from amongst all the potential *T*'s?

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?

Testing goodness

- A good transformation will agree with most of the matches
- A bad transformation will disagree with many 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

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

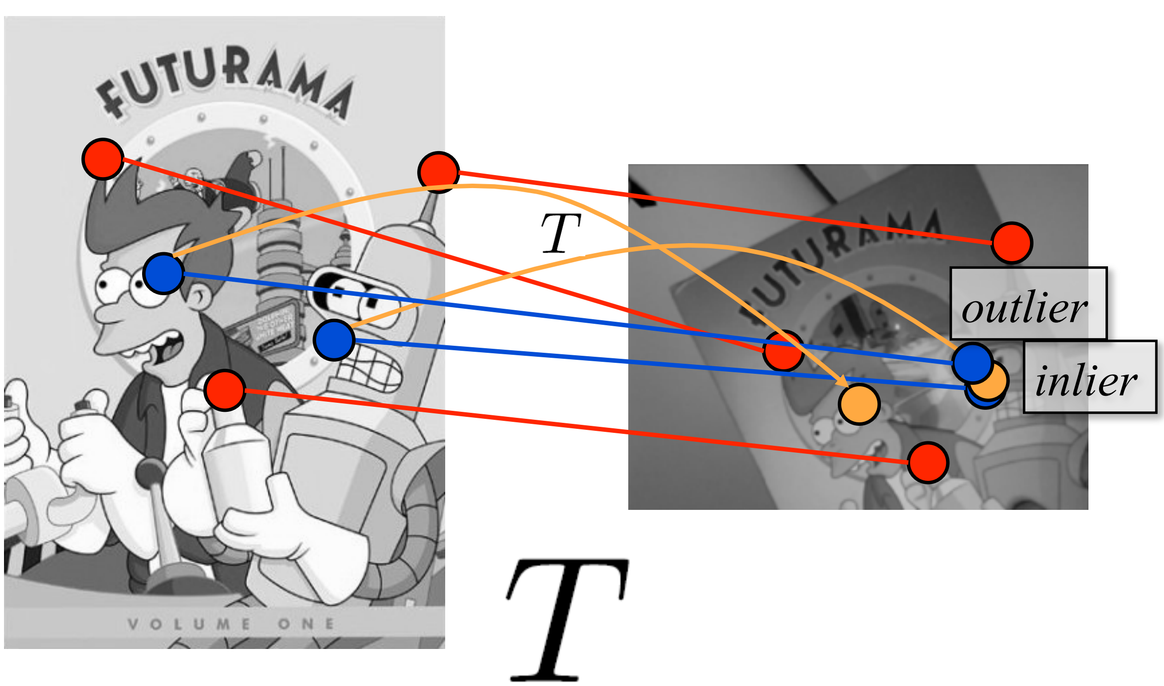
Testing goodness

- Find the distance between

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

- 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



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*, take 2

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?)