Feature-based object recognition

Prof. Noah Snavely
CS1114
http://cs1114.cs.cornell.edu

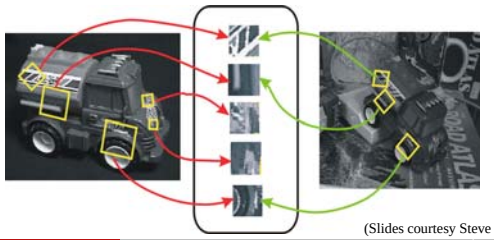
Cornell University Computer Science

Administrivia

- Assignment 4 due tomorrow, A5 will be out tomorrow, due in two parts
- Quiz 4 next Tuesday, 3/31
- Prelim 2 in two weeks, 4/7 (in class)
  - Covers everything since Prelim 1
  - There will be a review session next Thursday or the following Monday (TBA)

Invariant local features

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



(Slides courtesy Steve Seitz)

Feature Descriptors

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

More motivation…

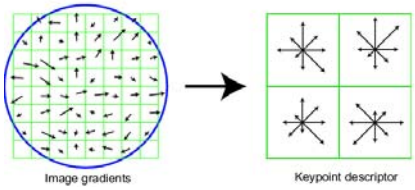
- Feature points are used for:
  - Image alignment (e.g., mosaics)
  - 3D reconstruction
  - Motion tracking
  - Object recognition
  - Robot navigation
  - …

SIFT Features

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

SIFT descriptor

- Very complicated, but very powerful
- (The details aren't all that important for this class.)
- 128 dimensional descriptor

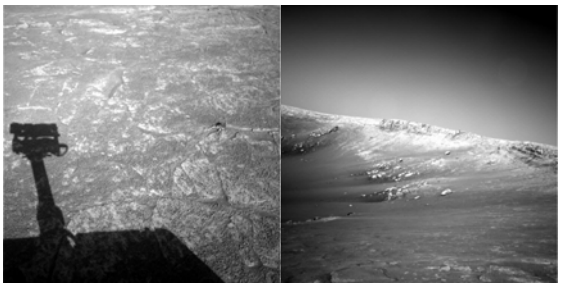


Adapted from a slide by David Lowe

Properties of SIFT

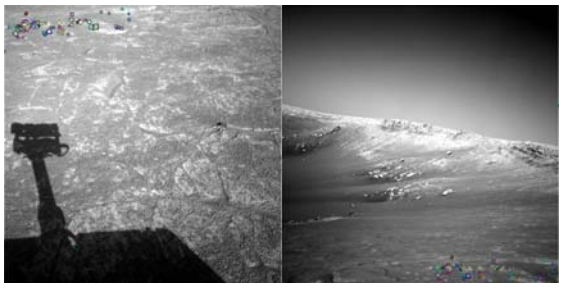
- Extraordinarily robust matching technique
  - Can handle significant changes in illumination
    - Sometimes even day vs. night (below)
  - Fast and efficient—can run in real time
  - Lots of code available
    - http://people.csail.mit.edu/albert/ladypack/wiki/index.php/Known_implementations_of_SIFT

Do these two images overlap?



NASA Mars Rover images

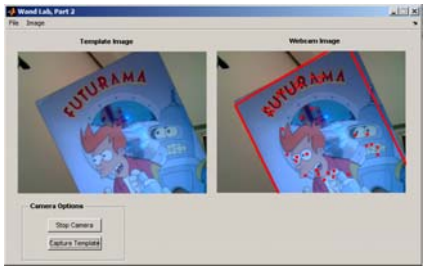
Answer below



NASA Mars Rover images

- Sony Aibo
- SIFT usage:
  - Recognize charging station
  - Communicate with visual cards
  - Teach object recognition

SIFT demo



How do we do this?

- Object matching in three steps:

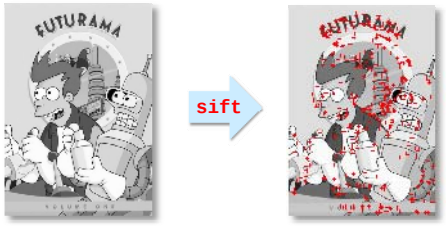
1. Detect features in the template and search images
2. Match features: find "similar-looking" features in the two images
3. Find a transformation T that explains the movement of the matched features

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);
```

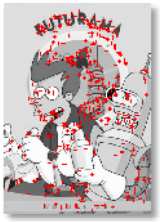

Step 1: Detecting SIFT features



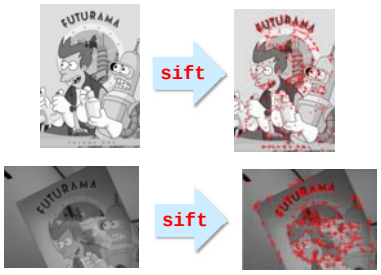
Step 1: Detecting SIFT features

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

% frames has a column for each
%    feature: [ x ; y ; scale ; orient ]
%
% descs also has a column for each
%    feature: 128-dimensional
%    vector describing the local
%    appearance of the feature
```



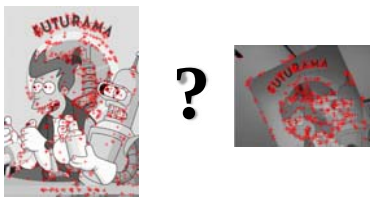
Step 1: Detecting SIFT features



(The number of features will very likely be different).

Step 2: Matching SIFT features

- How do we find matching features?



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

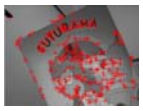
Simple matching algorithm

```
[frames1, descs1] = sift(img1);
[frames2, descs2] = sift(img2);
nF1 = length(frames1);  nF2 = length(frames2);

for i = 1:nF1
   minDist = Inf;  minIndex = -1;

   for j = 1:nF2
      diff = descs1(i,:) – descs2(j,:);
      dist = diff * diff';
      if dist < minDist
         minDist = dist; minIndex = j;
      end
   end
   fprintf('closest feature to %d is %d\n', i, minIndex);
end
```


What problems can come up?



- Not all features in image 1 are present in image 2
  - Some features aren't visible
  - Some features weren't detected
- → We might get lots of incorrect matches
- Slightly better version:
  - If the closest match is still too far away, throw the match away

Matching algorithm, Take 2

```
nF1 = length(frames1);  nF2 = length(frames2);

for i = 1:nF1
   minDist = inf;  minIndex = -1;

   for j = 1:nF2
      diff = descs1(i,:) – descs2(j,:);
      dist = diff * diff';
      if dist < minDist
         minDist = dist; minIndex = j;
      end
   end
   if minDist < threshold
     fprintf('closest feature to %d is %d\n', i, minIndex);
   end
end
```


Matching SIFT features

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

- Refresher: earlier, we learned about 2D linear transformations

$$T = \begin{bmatrix} a & b \\ c & d \end{bmatrix}$$

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} ax + by \\ cx + dy \end{bmatrix}$$

Image transformations

- Examples:

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

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

$$\begin{bmatrix} s & 0 \\ 0 & s \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} sx \\ sy \end{bmatrix}$$

Image transformations

- To handle translations, we added a third coordinate (always 1)

(x, y) → (x, y, 1)

- "Homogeneous" 2D points

Image transformations

- Example:

$$T = \begin{bmatrix} 1 & 0 & s \\ 0 & 1 & t \\ 0 & 0 & 1 \end{bmatrix}$$ *translation*

$$\begin{bmatrix} 1 & 0 & s \\ 0 & 1 & t \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} x+s \\ y+t \\ 1 \end{bmatrix}$$

Image transformations

- What about a general homogeneous transformation?

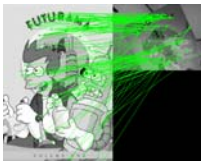
$$T = \begin{bmatrix} a & b & c \\ d & e & f \\ 0 & 0 & 1 \end{bmatrix}$$

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

- Called a 2D *affine* transformation

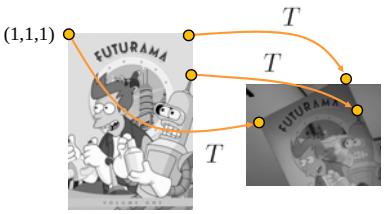
Solving for image transformations

- 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

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

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

An Algorithm: Take 1

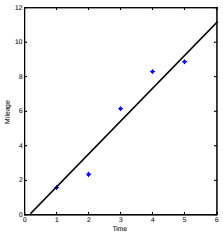
1. To find *T*, randomly guess a, b, c, d, e, f, check how well *T* matches the data
2. If it matches well, return *T*
3. Otherwise, go to step 1

Q: What does this remind you of?

There are much better ways to solve linear regression problems

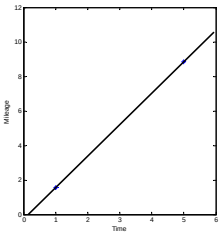
Linear regression

- Simplest case: fitting a line



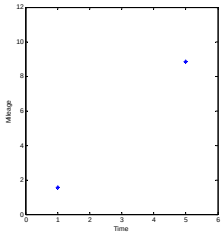
Linear regression

- Even simpler case: just 2 points



Linear regression

- Even simpler case: 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
  $[x_1\ y_1] \rightarrow [x_1'\ y_1']$

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

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

- How many equations, how many unknowns?

Finding an affine transform

- Need 3 matches → 6 equations
- 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)