Feature-based object recognition

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

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, ...

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

• Scale-Invariant Feature Transform
SIFT descriptor

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

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

Do these two images overlap?

Answer below

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

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

Step 1: Detecting SIFT features

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

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

Step 1: Detecting SIFT features

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

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

(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

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

Simple matching algorithm

```
[frame1, descs1] = sift(img1);
[frame2, descs2] = sift(img2);
N1 = length(frame1);  N2 = length(frame2);
for i = 1:N1
    minDist = Inf;  minIndex = -1;
    for j = 1:N2
        diff = descs1(i,:) - descs2(j,:);
        dist = diff' * diff;
        if dist < minDist
            minDist = dist; minIndex = j;
        end
    end
    fprintf('closest feature to %d is %d
', 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

```
N1 = length(frame1);  N2 = length(frame2);
for i = 1:N1
    minDist = inf;  minIndex = -1;
    for j = 1:N2
        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
', i, minIndex);
    end
end
```

Matching SIFT features

- Output of the matching step:
  Pairs of matching points
  - [ x1 y1 ] → [ x1' y1' ]
  - [ x2 y2 ] → [ x2' y2' ]
  - [ x3 y3 ] → [ x3' y3' ]
  ...
  - [ xk yk ] → [ xk' yk' ]
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 \( \rightarrow \) search image!

Image transformations

- Examples:

\[
S = \begin{bmatrix}
s & 0 \\
0 & s \\
\end{bmatrix}
\quad R = \begin{bmatrix}
\cos \theta & -\sin \theta \\
\sin \theta & \cos \theta \\
\end{bmatrix}
\]

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

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

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

\[
(x, y) \rightarrow (x, y, 1)
\]

- "Homogeneous" 2D points

Image transformations

- Example:

\[
T = \begin{bmatrix}
1 & 0 & s \\
0 & 1 & t \\
0 & 0 & 1 \\
\end{bmatrix}
\]

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

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?

How do we find $T$?

- We already have a bunch of point matches
  
  \[
  \begin{align*}
  [x_1, y_1] &\rightarrow [x_1', y_1'] \\
  [x_2, y_2] &\rightarrow [x_2', y_2'] \\
  [x_3, y_3] &\rightarrow [x_3', y_3'] \\
  \cdots \\
  [x_k, y_k] &\rightarrow [x_k', y_k']
  \end{align*}
  \]

- 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

- 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 unknowns
- 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
  \[ [x_1 \ y_1] \rightarrow [x'_1 \ y'_1] \]
  \[ [x_2 \ y_2] \rightarrow [x'_2 \ y'_2] \]
  \[ [x_3 \ y_3] \rightarrow [x'_3 \ y'_3] \]
  ...
  \[ [x_n \ y_n] \rightarrow [x'_n \ y'_n] \]

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 \( \rightarrow \) 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 \( d,e,f \))