Fitting image transformations

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

Cornell University
Computer Science

Administrivia

- A4 due tomorrow, A5 up next

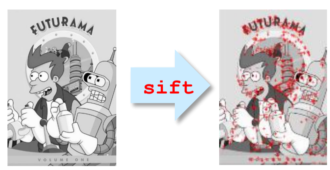
Next couple weeks

- How do we detect an object in an image?

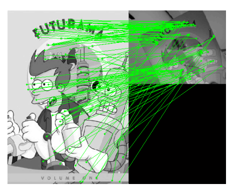
- Combines ideas from **image transformations**, **least squares**, and **robustness**

Object matching in three steps

1. Detect features in the template and search images

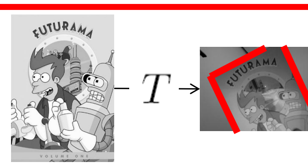


2. Match features: find "similar-looking" features in the two images



We started talking about this part last time

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



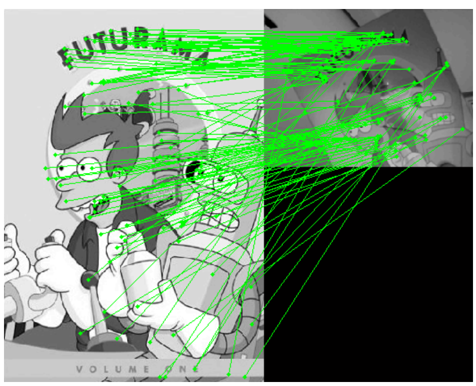
Affine transformations

- A *2D affine transformation* has the form:

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

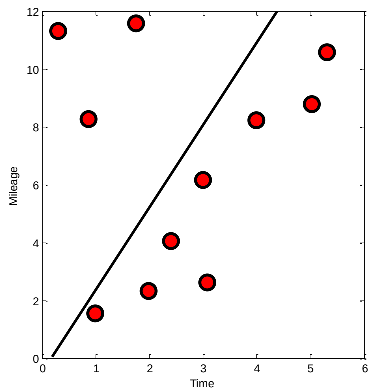
$$\begin{bmatrix} a & b & c \\ d & e & f \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} ax + by + c \\ dx + ey + f \\ 1 \end{bmatrix}$$

Fitting affine transformations



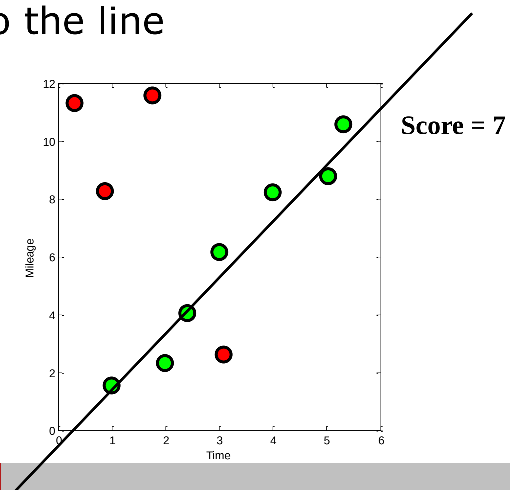
- We will fit an affine transformation to a set of feature matches
  - Problem: there are many incorrect matches

Linear regression



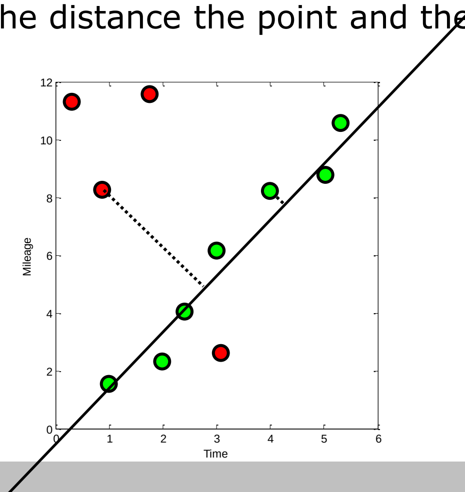
Testing goodness

- Idea: *count* the number of points that are "close" to the line



Testing goodness

- How can we tell if a point agrees with a line?
- Compute the distance the point and the line, and threshold



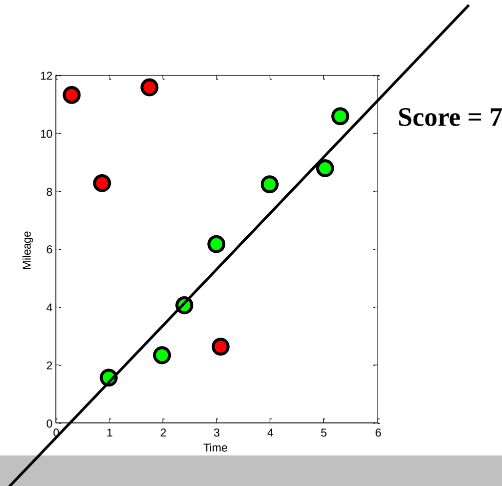
Testing goodness

- If the distance is small, we call this point an *inlier* to the line
- If the distance is large, it's an *outlier* to the line
- For an inlier point and a good line, this distance will be close to (but not exactly) zero
- For an outlier point or bad line, this distance will probably be large

- Objective function: find the line with the most inliers (or the fewest outliers)

Optimizing for inlier count

- How do we find the best possible line?



Algorithm (RANSAC)

1. Select two points at random
2. Solve for the line between those point
3. Count the number of inliers to the line *L*
4. If *L* has the highest number of inliers so far, save it
5. Repeat for N rounds, return the best *L*

Testing goodness

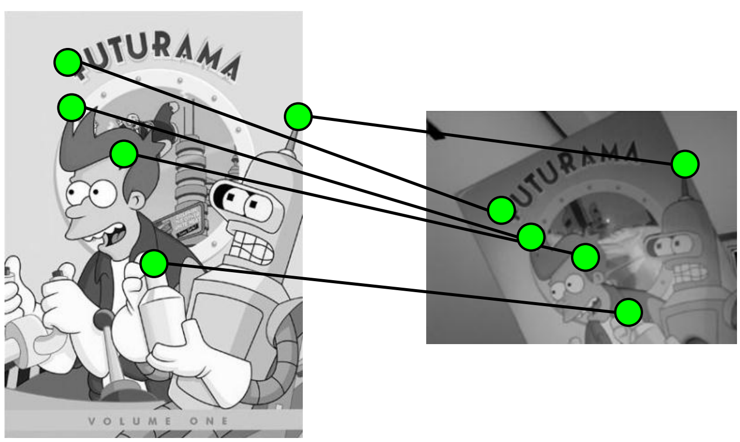
- This algorithm is called RANSAC (RANdom SAmple Consensus) – example of a randomized algorithm
- Used in an amazing number of computer vision algorithms
- Requires two parameters:
  - The agreement threshold (how close does an inlier have to be?)
  - The number of rounds (how many do we need?)

Randomized algorithms

- Very common in computer science
  - In this case, we avoid testing an infinite set of possible lines, or all $O(n^2)$ lines generated by pairs of points
- These algorithms find the right answer with some probability
- Often work very well in practice

Questions?

Very similar idea



- Given two images with a set of feature matches, how do we compute an affine transform between the two images?

Multi-variable fitting

- Let's consider 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 n matches

$[x_1 \; y_1] \rightarrow [x_1' \; y_1']$
$[x_2 \; y_2] \rightarrow [x_2' \; y_2']$
$[x_3 \; y_3] \rightarrow [x_3' \; y_3']$
…
$[x_n \; y_n] \rightarrow [x_n' \; y_n']$

Fitting an affine transformation

- Consider just one match

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

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

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

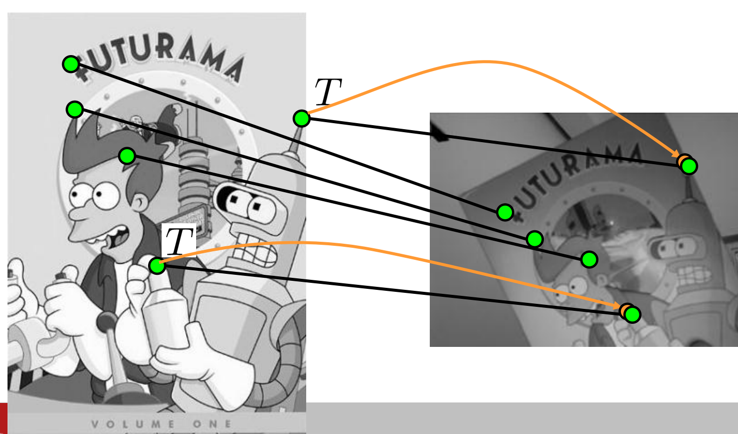
- 2 equations, 6 unknowns → we need at least 3 matches, but can fit n using least squares

Fitting an affine transformation

- 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

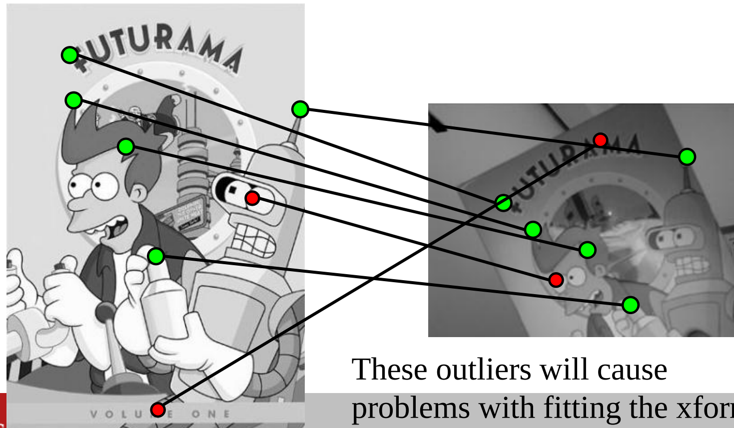
Fitting an affine transformation

- In other words:
  - Find 2D affine xform *T* that maps points in image 1 as close as possible to their matches in image 2



Back to fitting

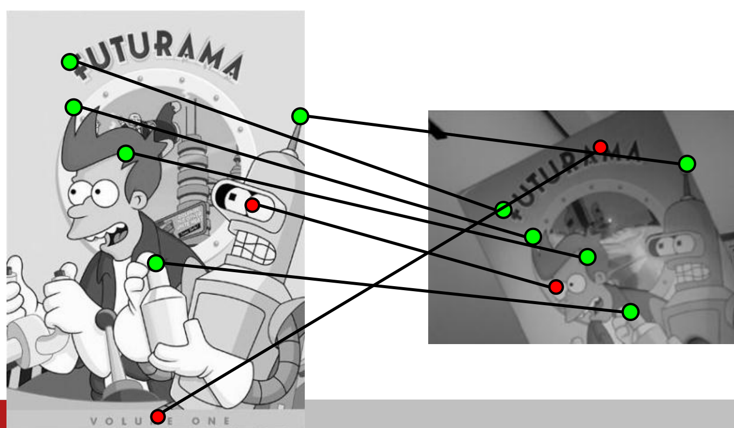
- Just like in the case of fitting a line or computing a median, we have some bad data (incorrect matches)



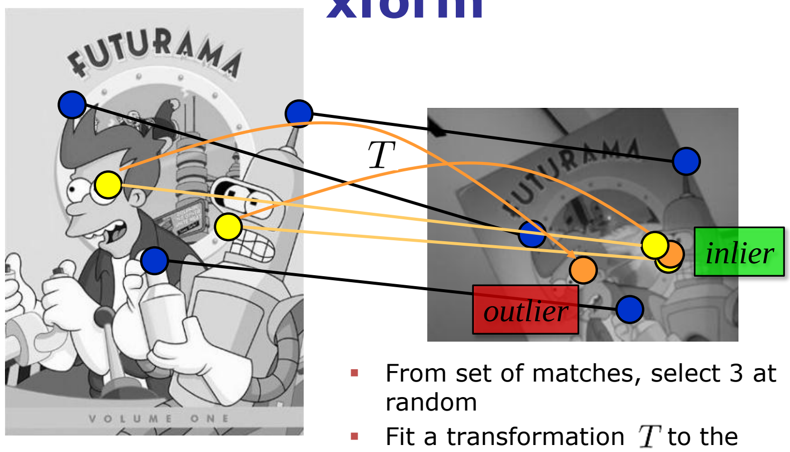
These outliers will cause problems with fitting the xform

How do we fix this?

- RANSAC to the rescue!



Generating and testing an xform



- From set of matches, select 3 at random
- Fit a transformation T to the selected matches
- Count inliers

Transform Fitting Algorithm (RANSAC)

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

How do we solve for T given 3 matches?

- Three matches give a linear system with six equations:

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

$$\begin{aligned} ax_1 + by_1 + c &= x_1' \\ dx_1 + ey_1 + f &= y_1' \end{aligned}$$

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

$$\begin{aligned} ax_2 + by_2 + c &= x_2' \\ dx_2 + ey_2 + f &= y_2' \end{aligned}$$

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

$$\begin{aligned} ax_3 + by_3 + c &= x_3' \\ dx_3 + ey_3 + f &= y_3' \end{aligned}$$

Two 3x3 linear systems

$$\begin{aligned} ax_1 + by_1 + c &= x_1' \\ ax_2 + by_2 + c &= x_2' \\ ax_3 + by_3 + c &= x_3' \end{aligned}$$

$$\begin{aligned} dx_1 + ey_1 + f &= y_1' \\ dx_2 + ey_2 + f &= y_2' \\ dx_3 + ey_3 + f &= y_3' \end{aligned}$$

Solving a 3x3 system

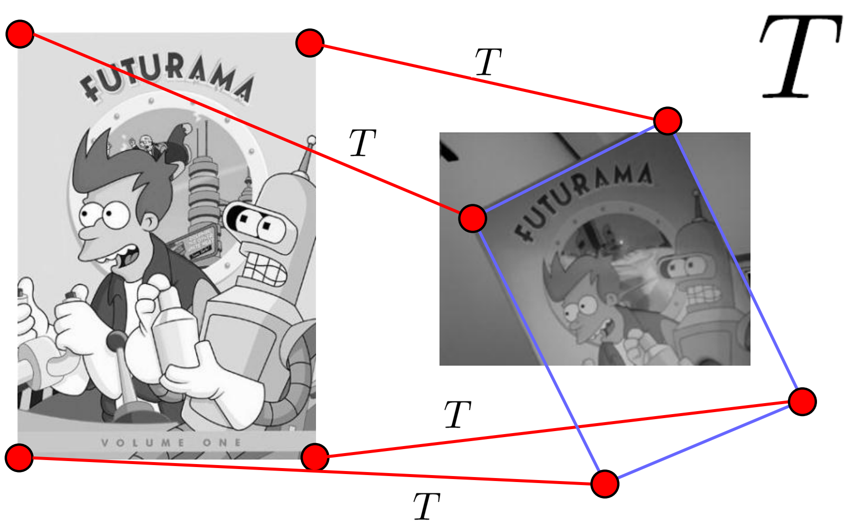
$$\mathbf{a}\mathbf{x}_1 + \mathbf{b}\mathbf{y}_1 + \mathbf{c} = \mathbf{x}_1'$$
$$\mathbf{a}\mathbf{x}_2 + \mathbf{b}\mathbf{y}_2 + \mathbf{c} = \mathbf{x}_2'$$
$$\mathbf{a}\mathbf{x}_3 + \mathbf{b}\mathbf{y}_3 + \mathbf{c} = \mathbf{x}_3'$$

- 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?

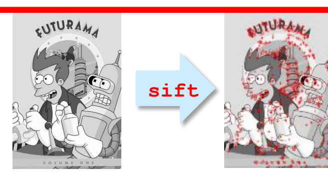
Finding the object boundary



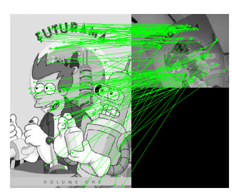
Questions?

Object matching in three steps

1. Detect features in the template and search images

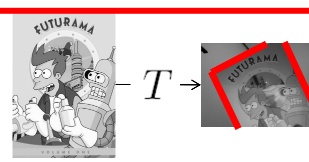


2. Match features: find "similar-looking" features in the two images



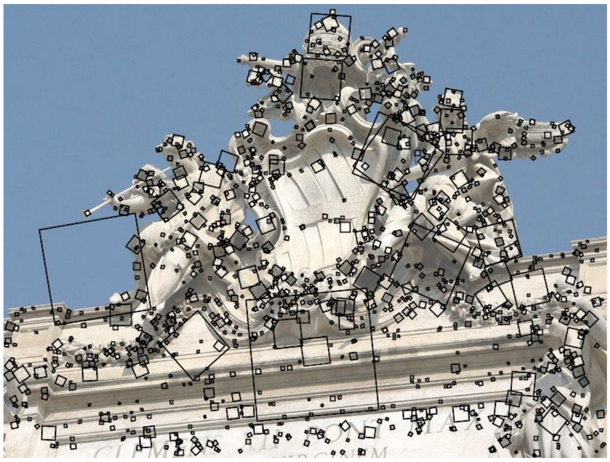
How do we do this part?

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



SIFT Features

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

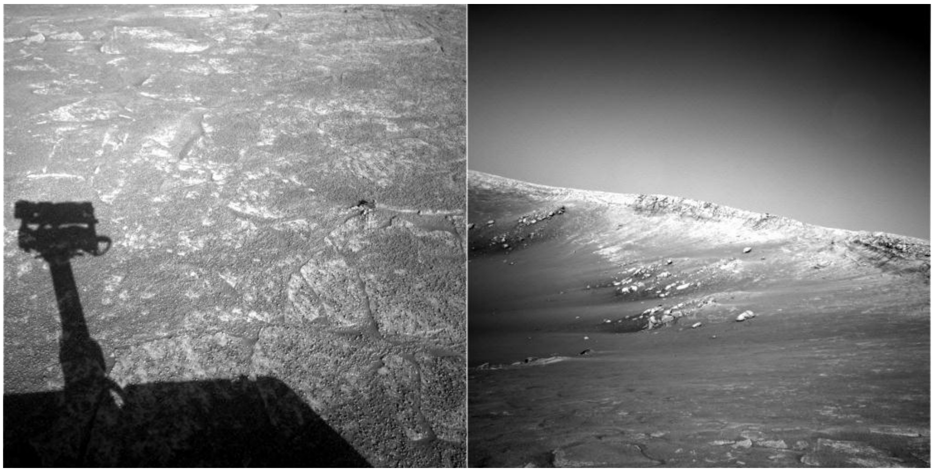


Cornell University

Properties of SIFT

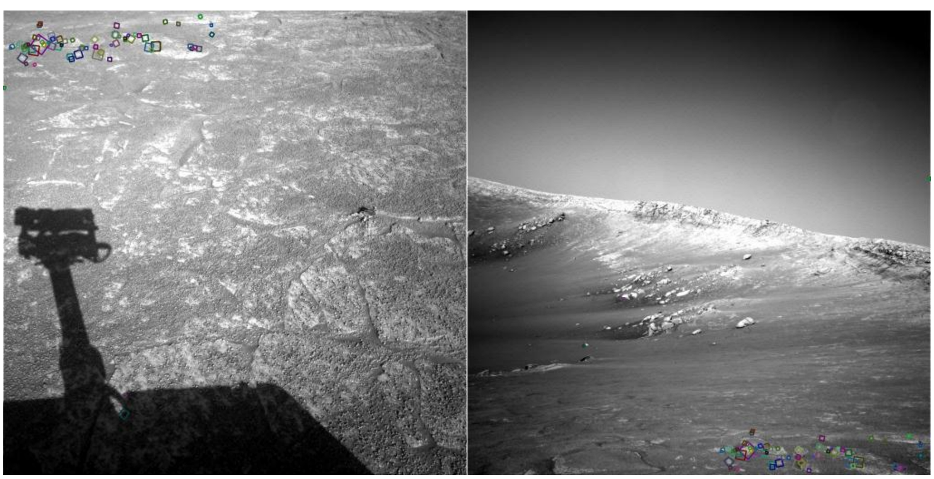
- Extraordinarily robust matching technique
  - Can handle significant changes in illumination
    - Sometimes even day vs. night (below)
  - Fast and efficient—can run in real time
  - Lots of code available
    - http://people.csail.mit.edu/albert/ladypack/wiki/index.php/Known_implementations_of_SIFT

Do these two images overlap?



NASA Mars Rover images

Answer below



NASA Mars Rover images

Cornell University