

ECE 1390/2390

Image Processing and Computer Vision – Fall 2021

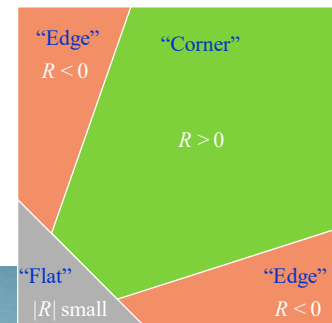
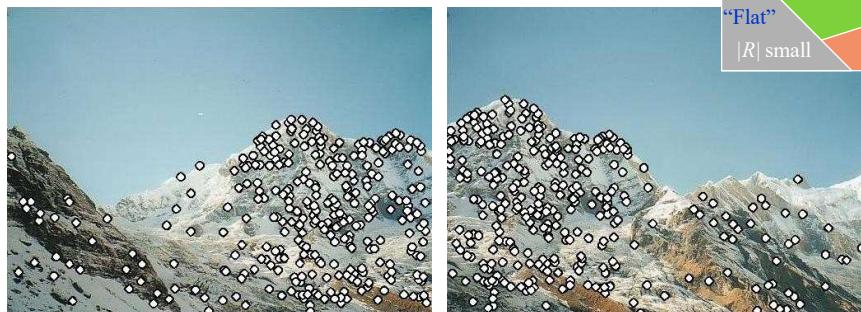
RANSAC

Ahmed Dallal

Feature-based alignment to find transforms

Overall strategy:

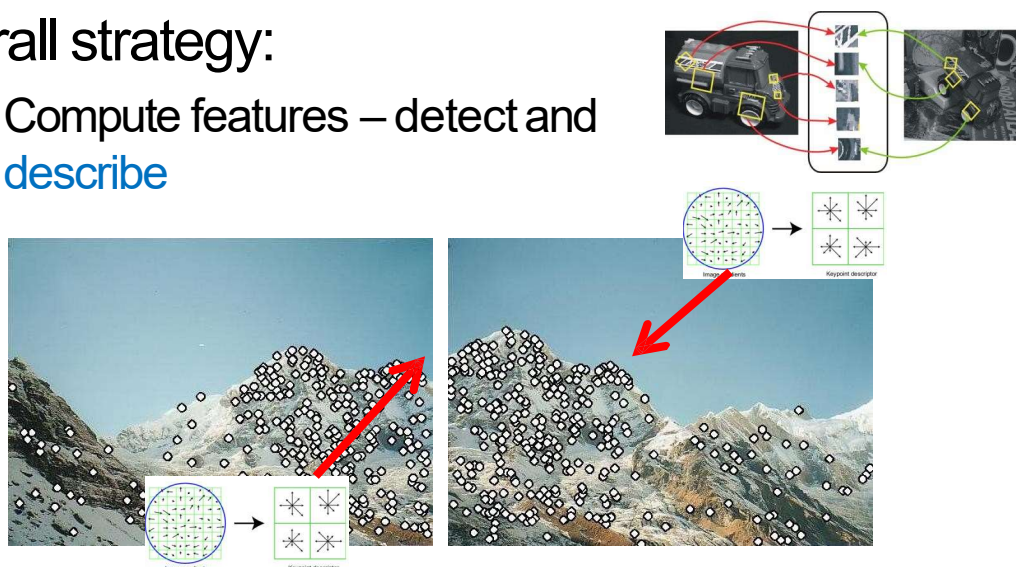
1. Compute features – **detect** and describe



Feature-based alignment to find transforms

Overall strategy:

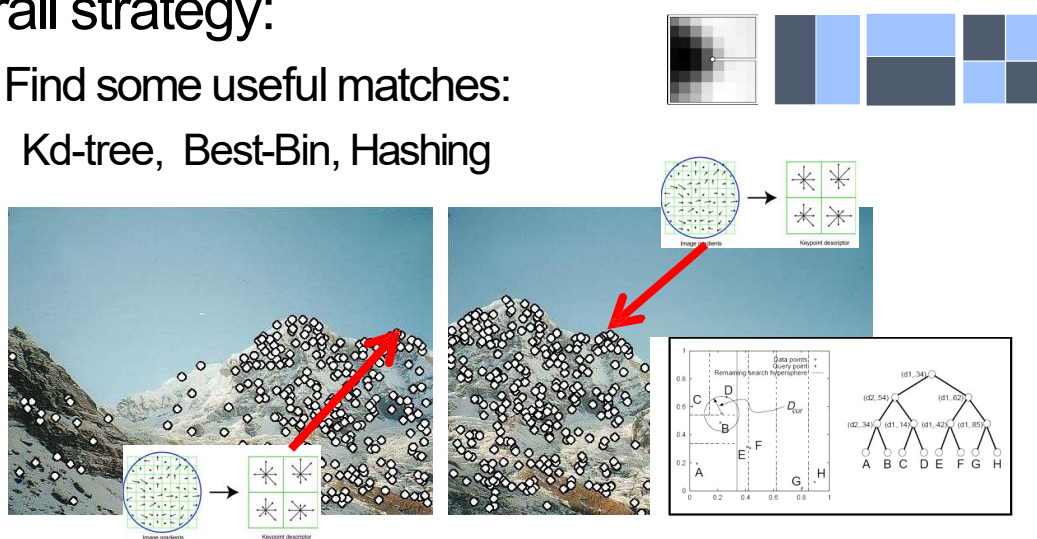
1. Compute features – detect and describe



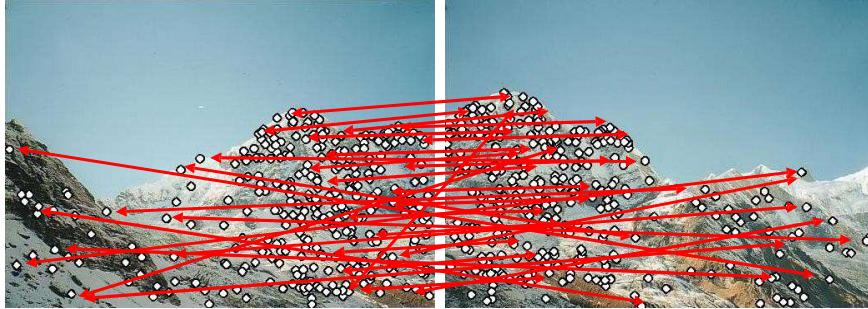
Feature-based alignment to find transforms

Overall strategy:

2. Find some useful matches:
Kd-tree, Best-Bin, Hashing



Feature-based alignment algorithm



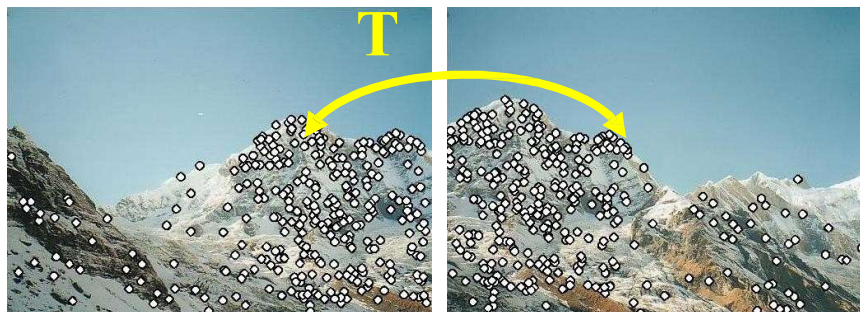
2. Compute **putative** matches - e.g. “closest descriptor”

Kd-tree, best bin, etc...

Feature-based alignment to find transforms

Overall strategy:

3. **Compute** and apply the best transformation:
e.g. affine, translation, or homography



Feature-based alignment to find transforms

Overall strategy:

3. Compute and **apply** the best transformation:
e.g. affine, translation, or homography



How to get “putative” matches?

Feature matching

- Exhaustive search – one against another
- Hashing
- Nearest neighbor techniques

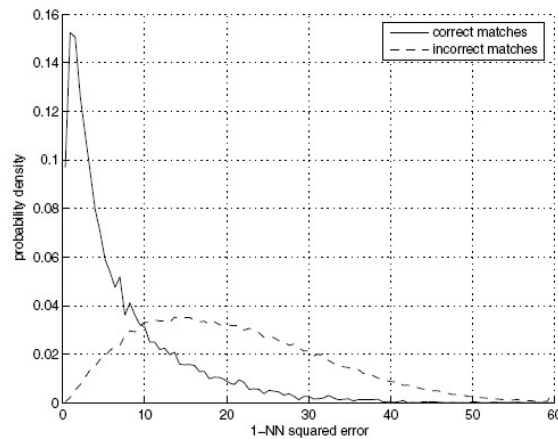
...but these give the best match. How do we know it's a good one?

Feature-space outlier rejection

- Let's not match all features, but only these that have “**similar enough**” matches?
- How can we do it?
 - $\text{SSD}(\text{patch1}, \text{patch2}) < \text{threshold}$
 - How to set threshold?

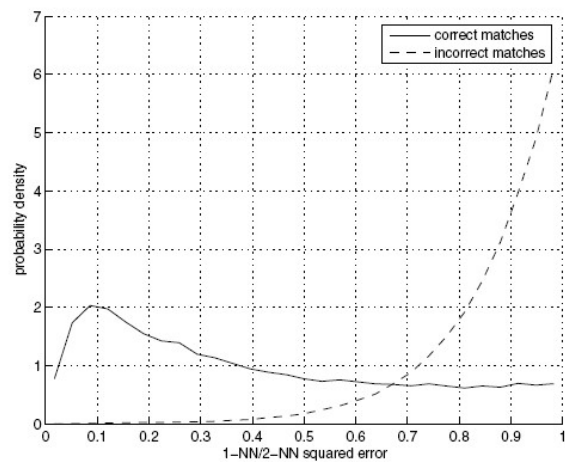
Feature-space outlier rejection

- How to set threshold?



A better way [Lowe, 1999]:

- 1-NN: SSD of the closest match
- 2-NN: SSD of the second-closest match
- Look at how much better 1-NN is than 2-NN, e.g. $\frac{1\text{-NN SSD}}{2\text{-NN SSD}}$
 - That is, is our best match much better than the next?



Feature matching

- Exhaustive search
- Hashing
- Nearest neighbor techniques
- But...remember the distinctive vs invariant competition? Implies:
 - *Problem: Even when pick best match, still lots (and lots) of wrong matches - “outliers”. What should we do?*

Another way to remove mistakes

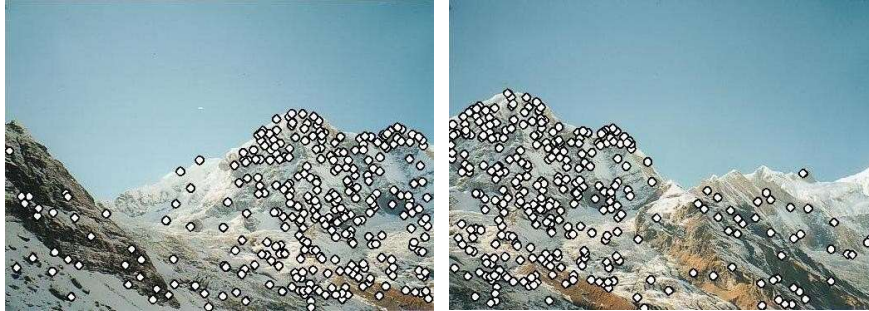
- Why are we doing matching?
 - To compute a model of the relation between entities

‘Find consistent matches’???

- Some points (many points) are static in the world
- Some are not
- Need to find the right ones so can compute pose.
- Well tried approach:
 - *Random Sample Consensus (RANSAC)*

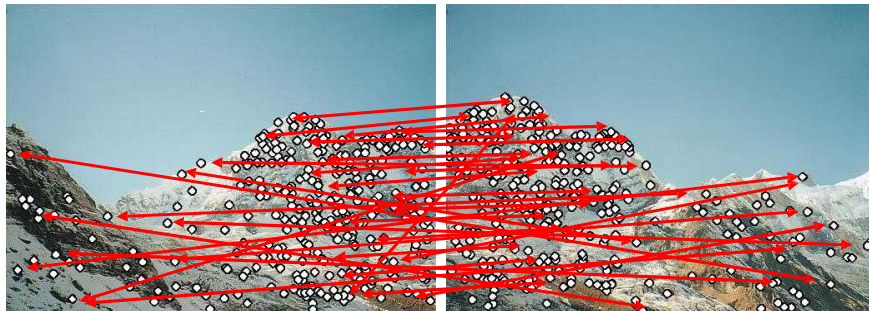
RANSAC

Feature-based alignment algorithm



1. Extract features

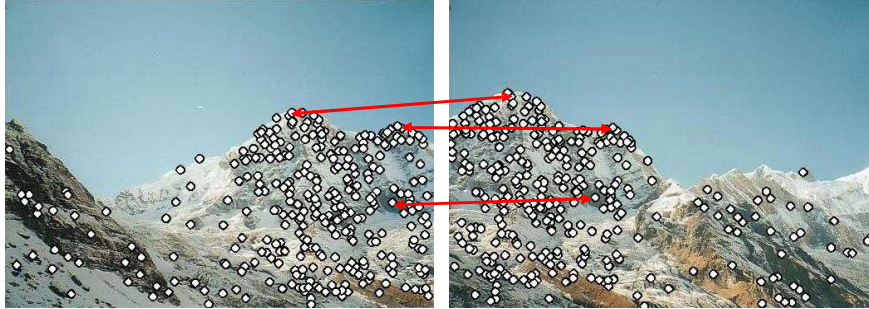
Feature-based alignment algorithm



2. Compute *putative* matches - e.g. "closest descriptor"

Kd-tree, best bin, etc...

Feature-based alignment algorithm



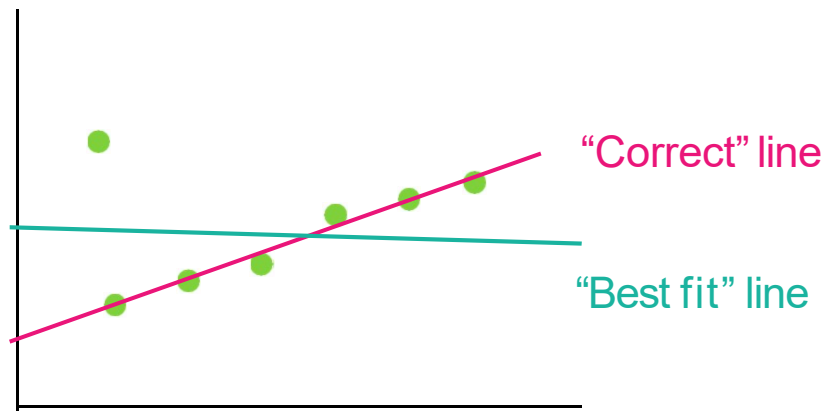
3. Loop until happy:

- *Hypothesize* transformation T from **some** matches
- *Verify* transformation (search for other matches **consistent** with T) – mark best

“Find consistent matches”?

- Some “best” matches are correct
- Some are not. And the “not” are not part of any other consistent match...
- Need to find the right ones so can compute the pose/transform/fundamental... *the model*.
- Today: Random Sample Consensus (RANSAC)

Simple Example: Fitting a line



RANSAC: Main idea

- Fitting a line (model) is easy if we know which points belong and which do not. (duh...)
- If we had a proposed line (model), we could probably guess which points belong to that line (model): *inliers*.
- **RAN**dom **SA**mples **C**onsensus: randomly pick some points to define your line (model). Repeat enough times until you find a good line (model) – one with many inliers.
- Fischler & Bolles 1981 – Coping with a large proportion of outliers

RANSAC for general model

A given model type has a *minimal set* – the smallest number of samples from which the model can be computed.

- Line: 2 points

RANSAC for general model

Image transformations are models. Minimal set of *s* of point pairs/matches:

- Translation: pick one pair of matched points
- Homography (for plane) – pick 4 point pairs
- Fundamental matrix – pick 8 point pairs (really 7 but lets not go there)

RANSAC for general model

General RANSAC algorithm

- Randomly select s points (or point pairs) to form a *sample*
- Instantiate the model
- Get consensus set C_i : The points within error bounds (distance threshold) of the model
- If $|C_i| > T$, terminate and return model, or
- Repeat for N trials, return model with $\max |C_i|$

RANSAC algorithm

1. **Sample** (randomly) the number of points required to fit the model
2. **Solve** for model parameters using sample
3. **Score** by the fraction of *inliers* within a preset threshold of the model
4. **Repeat** 1-3 until the best model is found with high confidence

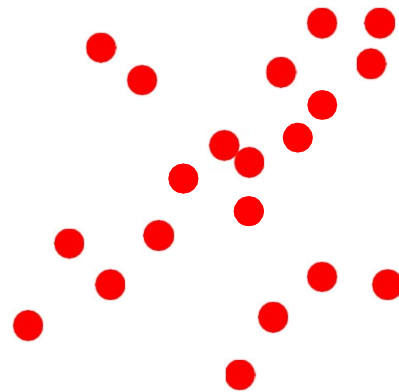
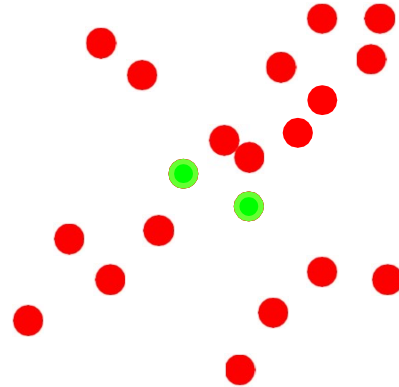


Illustration by Savarese

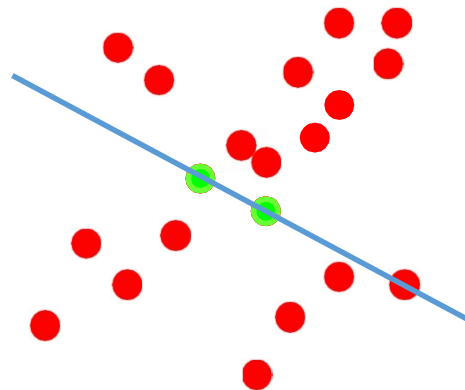
RANSAC algorithm

1. **Sample** (randomly) the number of points required to fit the model
2. **Solve** for model parameters using sample
3. **Score** by the fraction of *inliers* within a preset threshold of the model
4. **Repeat** 1-3 until the best model is found with high confidence



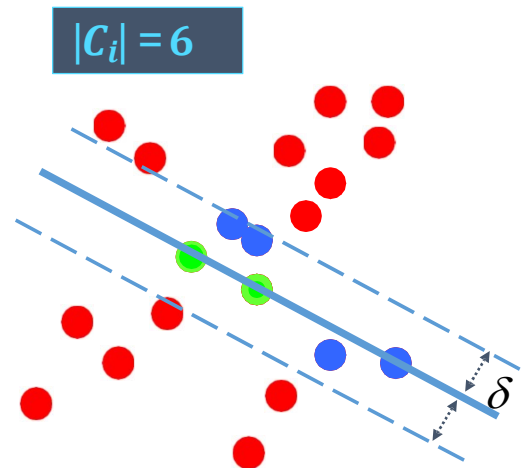
RANSAC algorithm

1. **Sample** (randomly) the number of points required to fit the model
2. **Solve** for model parameters using sample
3. **Score** by the fraction of *inliers* within a preset threshold of the model
4. **Repeat** 1-3 until the best model is found with high confidence



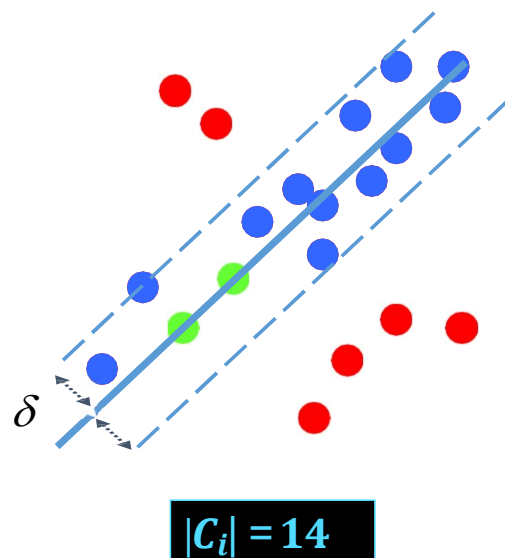
RANSAC algorithm

1. **Sample** (randomly) the number of points required to fit the model
2. **Solve** for model parameters using sample
3. **Score** by the fraction of *inliers* within a preset threshold of the model
4. **Repeat** 1-3 until the best model is found with high confidence



RANSAC algorithm

1. **Sample** (randomly) the number of points required to fit the model
2. **Solve** for model parameters using sample
3. **Score** by the fraction of *inliers* within a preset threshold of the model
4. **Repeat** 1-3 until the best model is found with high confidence



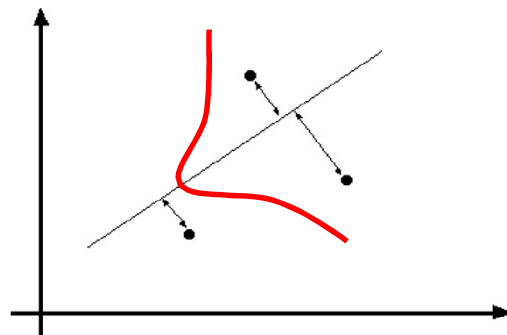
Choosing the parameters

1. Initial number of points in the minimal set s
 - Typically minimum number needed to fit the model
2. Distance threshold δ (sometimes called t)
You need a noise model

Distance Threshold

- Let's assume **location** noise is Gaussian with σ^2
- Then the **distance** d has **Chi** distribution with k degrees of freedoms where k is the dimension of the Gaussian.
- If one dimension, e.g. distance off a line, then 1 DOF

$$f(d) = \frac{2e^{-\left(\frac{d^2}{2\sigma^2}\right)}}{\pi\sigma}, d \geq 0$$



Distance Threshold

For 95% cumulative threshold t when
Gaussian with σ^2 : $t^2 = 3.84\sigma^2$

That is: if $t^2 = 3.84\sigma^2$ then 95% probability
that $d < t$ when point is inlier

Choosing the parameters

Initial number of points s

- Typically minimum number needed to fit the model

Distance threshold t

- Choose t so probability for inlier is high (e.g. 0.95)
- Zero-mean Gaussian noise with std. dev. σ : $t^2 = 3.84\sigma^2$

Number of samples N

- Choose N so that, with probability p , at least one random sample set is free from outliers (e.g. $p = 0.99$)
- Need to set N based upon the outlier ratio e

Calculate N

1. s – number of points to compute solution
2. p – probability of success
3. e – proportion outliers, so % inliers = $(1 - e)$
4. $P(\text{sample set with all inliers}) = (1 - e)^s$
5. $P(\text{sample set will have at least one outlier}) =$
 $(1 - (1 - e)^s)$
6. $P(\text{all } N \text{ samples have outlier}) = (1 - (1 - e)^s)^N$
7. We want $P(\text{all } N \text{ samples have outlier}) < (1 - p)$
8. So $(1 - (1 - e)^s)^N < (1 - p)$

Calculate N

$$N > \log(1 - p) / \log(1 - (1 - e)^s)$$

$$N > \log(1 - p) / \log(1 - (1 - e)^s)$$

- Set $p=0.99$ – chance of getting good sample

$s=2, e=5\%$ $\Rightarrow N=2$

$s=2, e=50\%$ $\Rightarrow N=17$

$s=4, e=5\%$ $\Rightarrow N=3$

$s=4, e=50\%$ $\Rightarrow N=72$

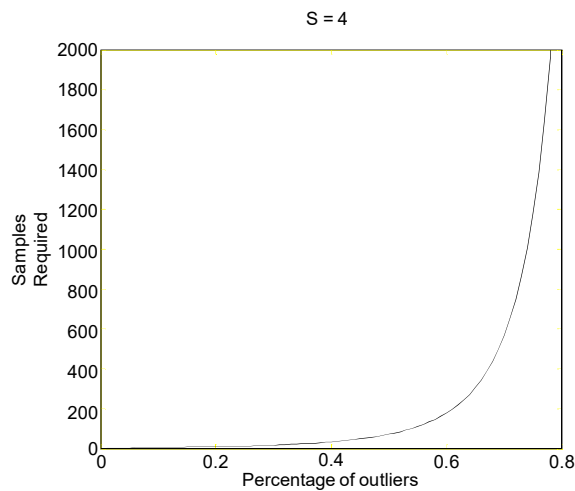
$s=8, e=5\%$ $\Rightarrow N=5$

$s=8, e=50\%$ $\Rightarrow N=1177$

s	proportion of outliers e						
	5%	10%	20%	25%	30%	40%	50%
2	2	3	5	6	7	11	17
3	3	4	7	9	11	19	35
4	3	5	9	13	17	34	72
5	4	6	12	17	26	57	146
6	4	7	16	24	37	97	293
7	4	8	20	33	54	163	588
8	5	9	26	44	78	272	1177

- N increases steeply with s

$$N > \log(1 - p) / \log(1 - (1 - e)^s)$$

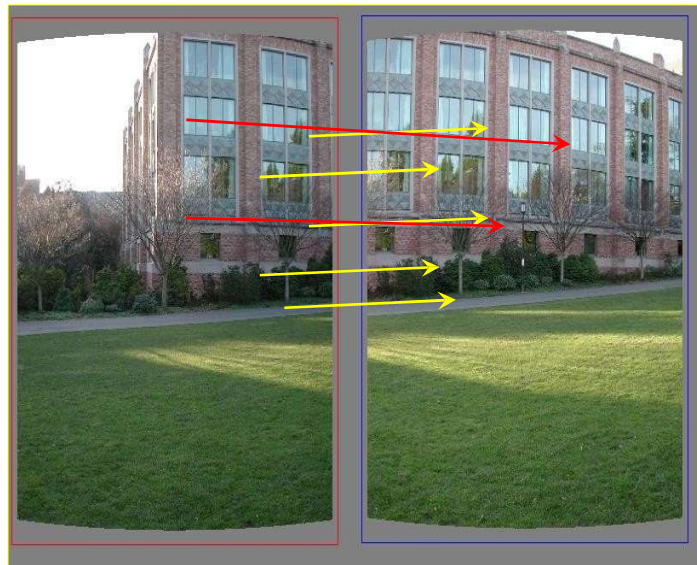


How big does N need to be?

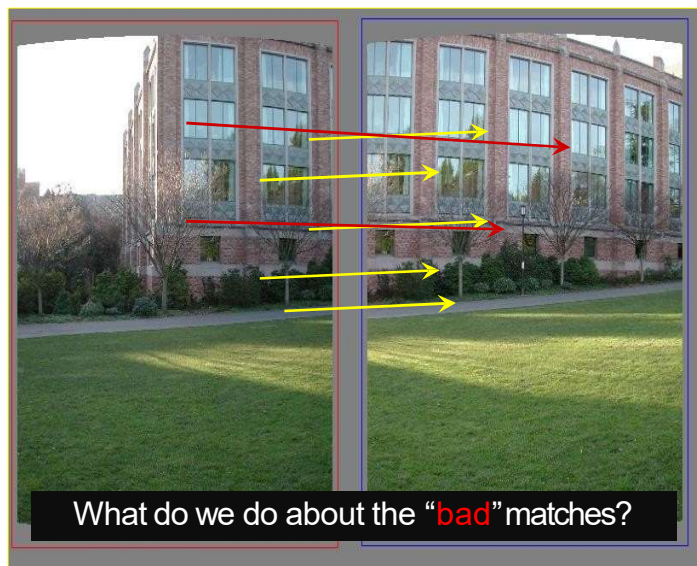
$$N > \log(1 - p) / \log(1 - (1 - e)^s)$$

$N = f(e, s, p)$, *but not the number of points/features!*

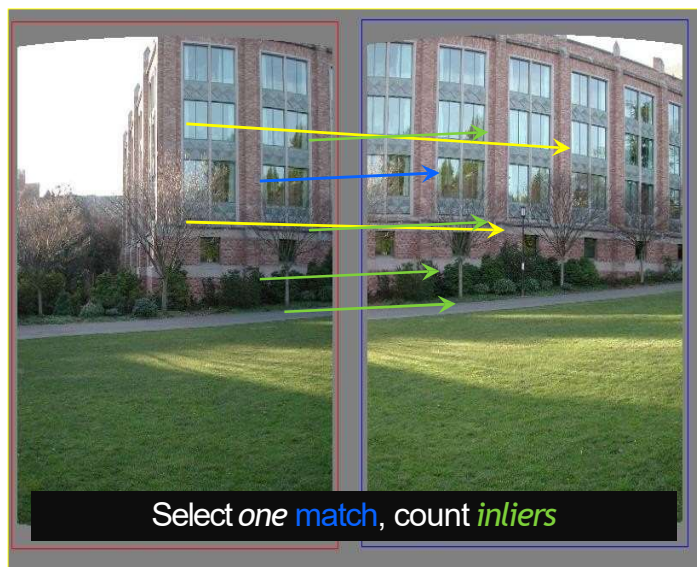
Matching features



Matching features - Example

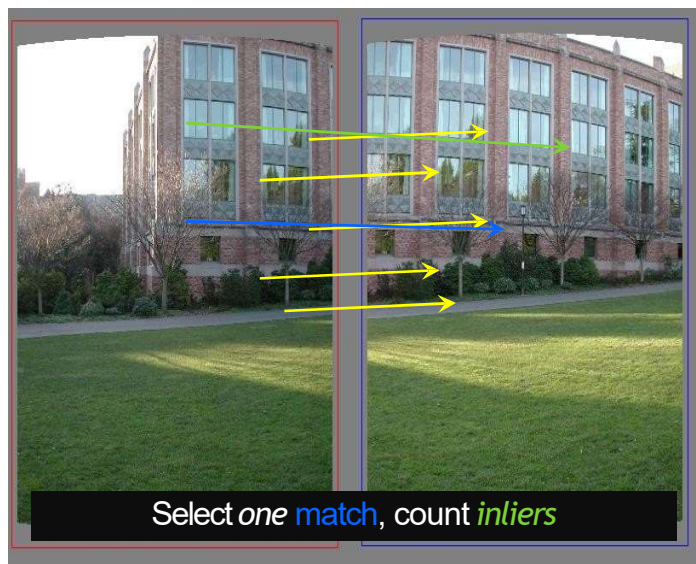


RAndom SAmples Consensus(1)



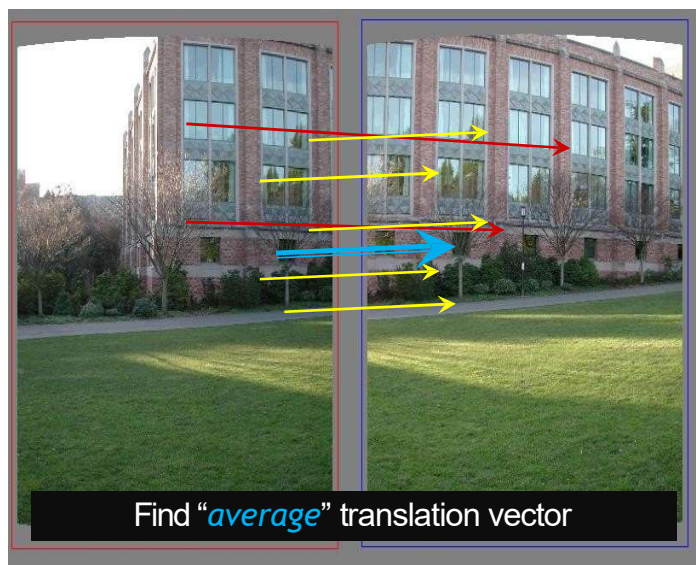
5 total inliers
(including selected)

RAndom SAmples Consensus(2)



2 total inliers

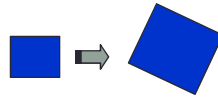
Least squares fit



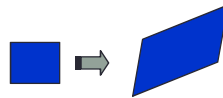
Note: We use the first set of matches as it had stronger consensus (5 inliers)

2D transformation models

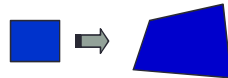
2 matches:
Similarity
(translation, scale, rotation)



3 matches:
Affine



4 matches:
Projective
(homography)



Source: S. Lazebnik

RANSAC for estimating homography

RANSAC loop:

1. Select four feature pairs (at random)
2. Compute homography H (exact)
3. Compute *inliers* where $SSD(p_i', H p_i) < \varepsilon$
4. Keep largest set of inliers
5. Re-compute least-squares H estimate on all of the inliers

Adaptively determining the number of samples

- Inlier ratio e is often unknown a priori
- Pick worst case, e.g. 50% ($e = 0.5$) and adapt if more inliers are found
e.g. 80% inliers would yield $e = 0.2$

Source: M. Pollefeys

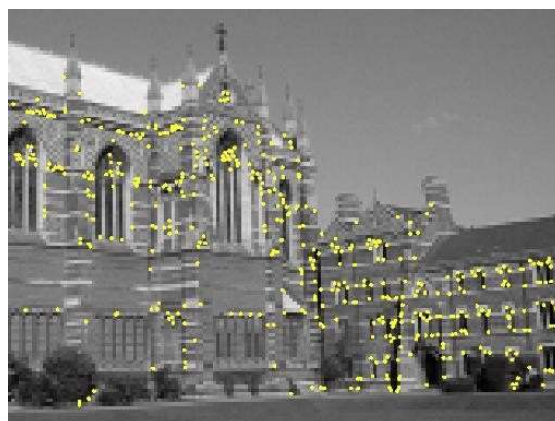
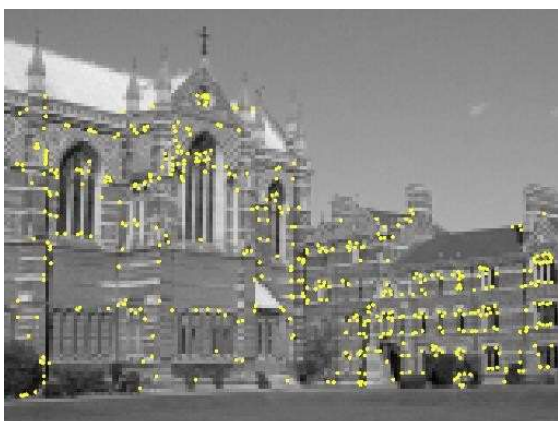
Adaptive procedure

- $N = \infty$, sample_count = 0, $e = 1.0$
- While $N > \text{sample_count}$
 - Choose a sample and count the number of inliers
 - Set $e_0 = 1 - \frac{\text{number of inliers}}{\text{total number of points}}$
 - If $e_0 < e$ Set $e = e_0$ and recompute N from e :
$$N = \log(1 - p) / \log(1 - (1 - e)^s)$$
 - Increment the sample_count by 1

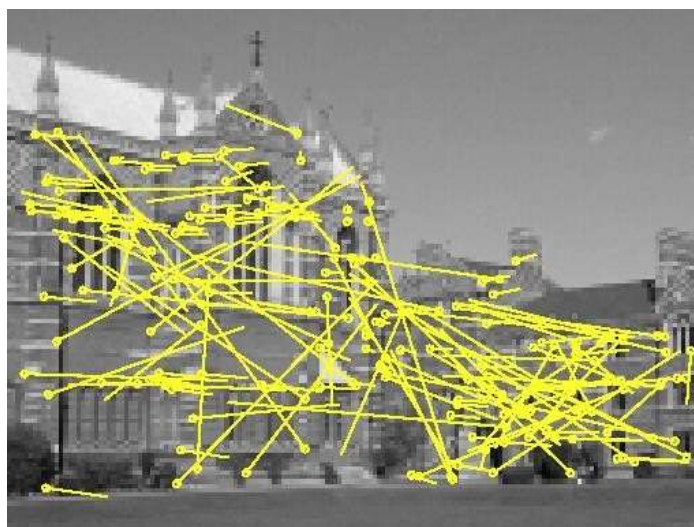
RANSAC for recognition



RANSAC for finding fundamental matrix



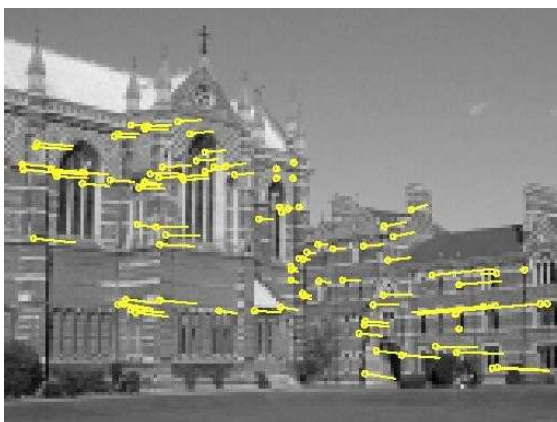
Putative matches (motion) by cross-correlation



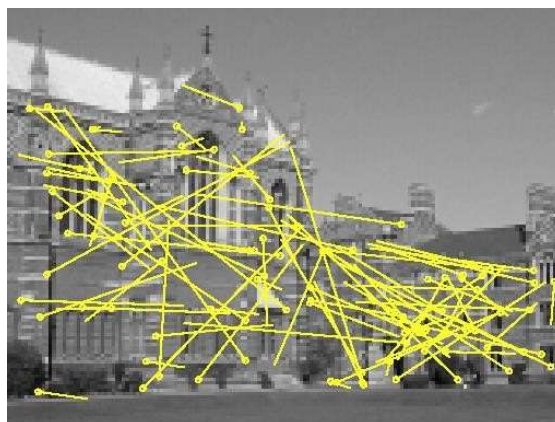
188 matches

RANSAC for fundamental matrix

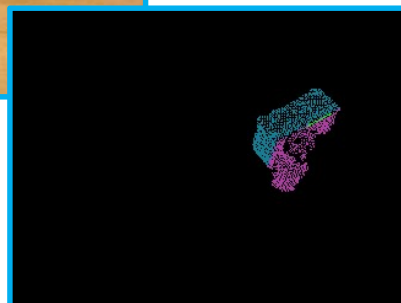
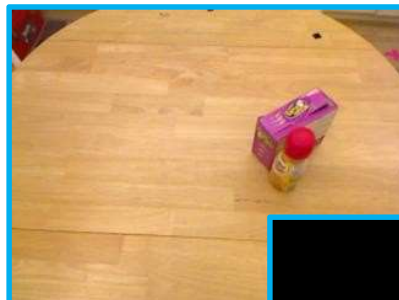
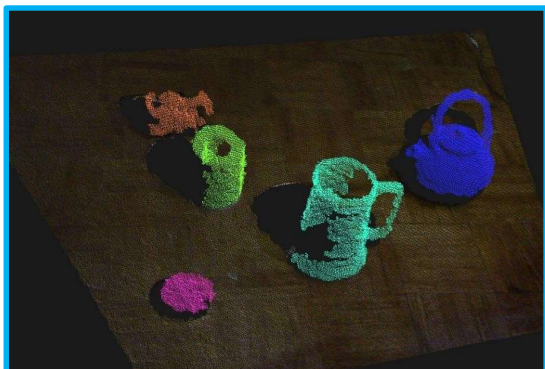
Inliers (99)



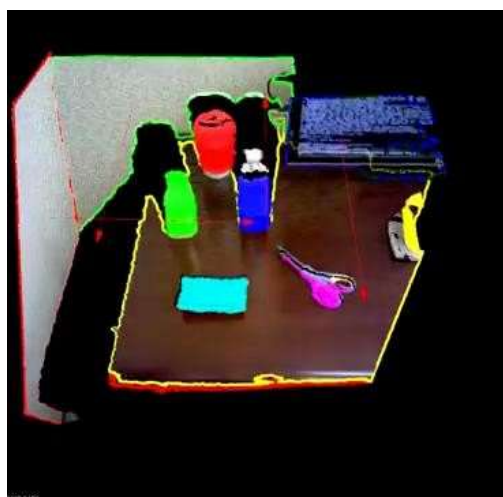
Outliers (89)



Point cloudplanes



Find the plane and object in realtime



RANSAC: Conclusions

The good...

- Simple and general
- Applicable to many different problems, often works well in practice
- Robust to large numbers of outliers
- Applicable for larger number of parameters than Hough transform
- Parameters are easier to choose than Hough transform

RANSAC: Conclusions

The not-so-good...

- Computational time grows quickly with the number of model parameters
- Not as good for getting multiple fits
- Really not good for approximate models

RANSAC: Conclusions

Common applications

- Computing a homography (e.g., image stitching) or other image transform
- Estimating fundamental matrix (relating two views)
- *Pretty much every problem in robot vision*