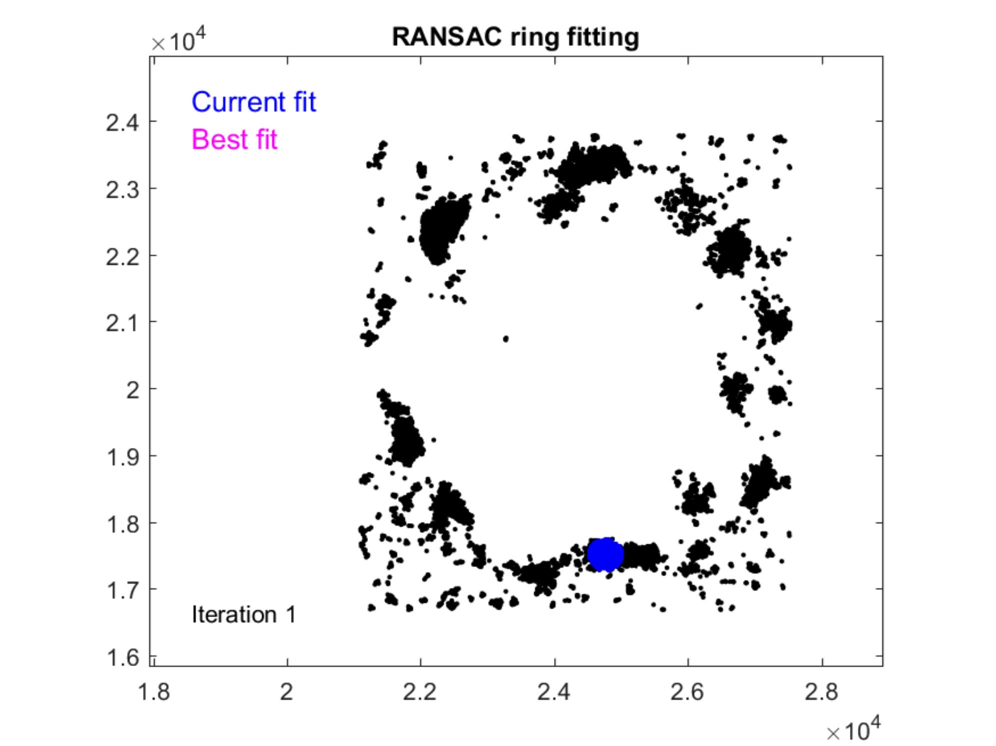
RANSAC

March 6th, 2019

Presented by Dr. Sander Ali Khowaja



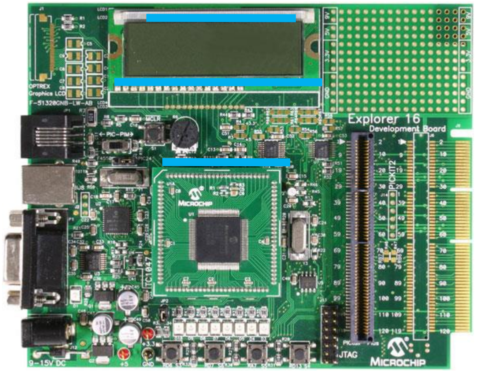
Fitting as search in parametric space

- Choose a parametric model to represent a set of features
- Membership criterion is not local
  - Can’t tell whether a point belongs to a given model just by looking at that point.
- Three main questions:
  - What model represents this set of features best?
  - Which of several model instances gets which feature?
  - How many model instances are there?
- Computational complexity is important
  - It is infeasible to examine every possible set of parameters and every possible combination of features

Source: L. Lazebnik

Example: Line Fitting

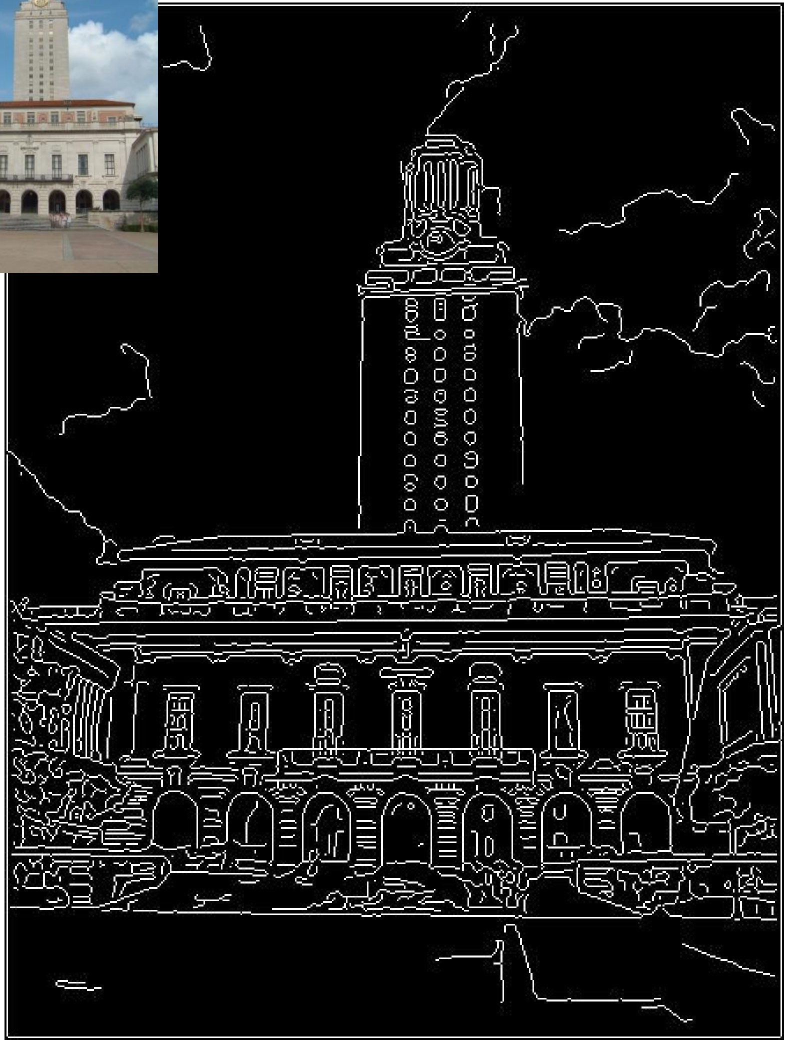
- Why fit lines?
  Many objects characterized by presence of straight lines

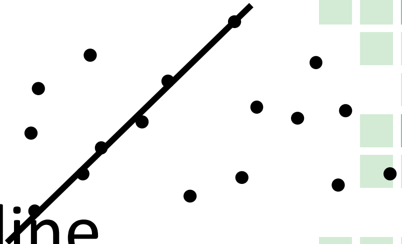


- Wait, why aren’t we done just by running edge detection?

Slide credit: Kristen Grauman

Difficulty of Line Fitting





- Extra edge points (clutter), multiple models:
  - Which points go with which line, if any?
- Only some parts of each line detected, and some parts are missing:
  - How to find a line that bridges missing evidence?
- Noise in measured edge points, orientations:
  - How to detect true underlying parameters?

Slide credit: Kristen Grauman

Voting

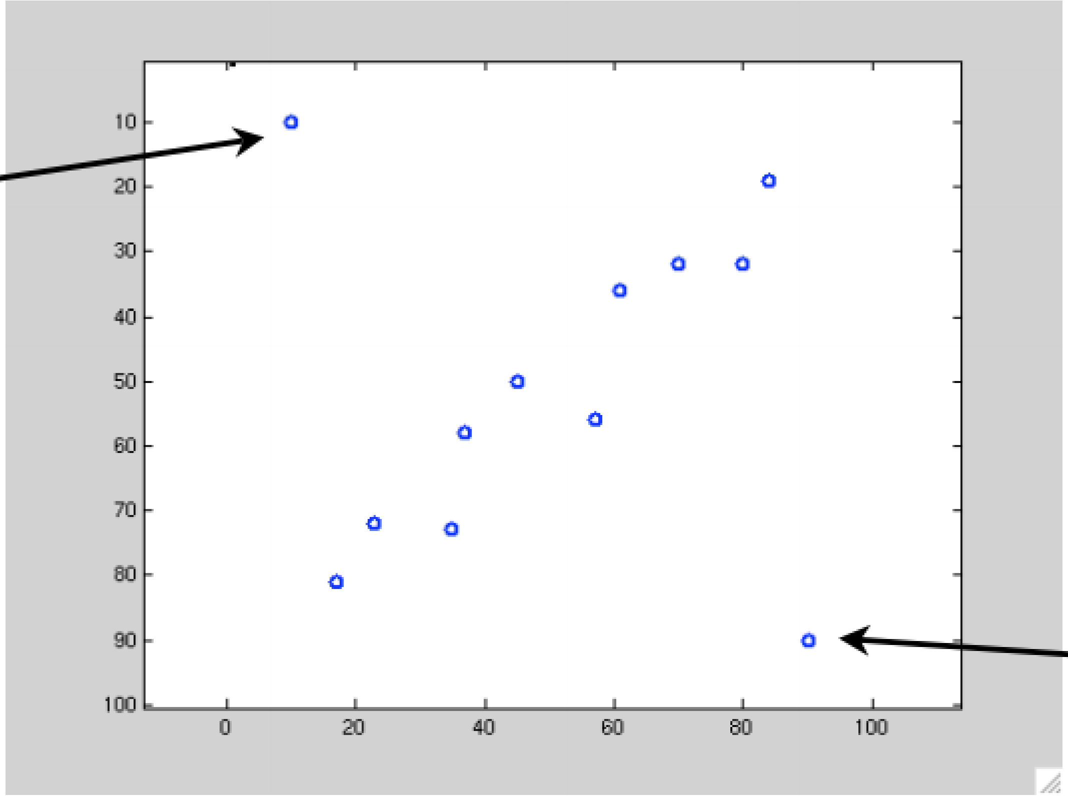
- It's not feasible to check all combinations of features by fitting a model to each possible subset.
- Voting is a general technique where we let the features vote for all models that are compatible with it.
  - Cycle through features, cast votes for model parameters.
  - Look for model parameters that receive a lot of votes.
- Noise & clutter features will cast votes too, *but* typically their votes should be inconsistent with the majority of "good" features.
- Ok if some features not observed, as model can span multiple fragments.

Slide credit: Kristen Grauman

Outliers

Loosely speaking, outliers are points that don't "fit" the model.

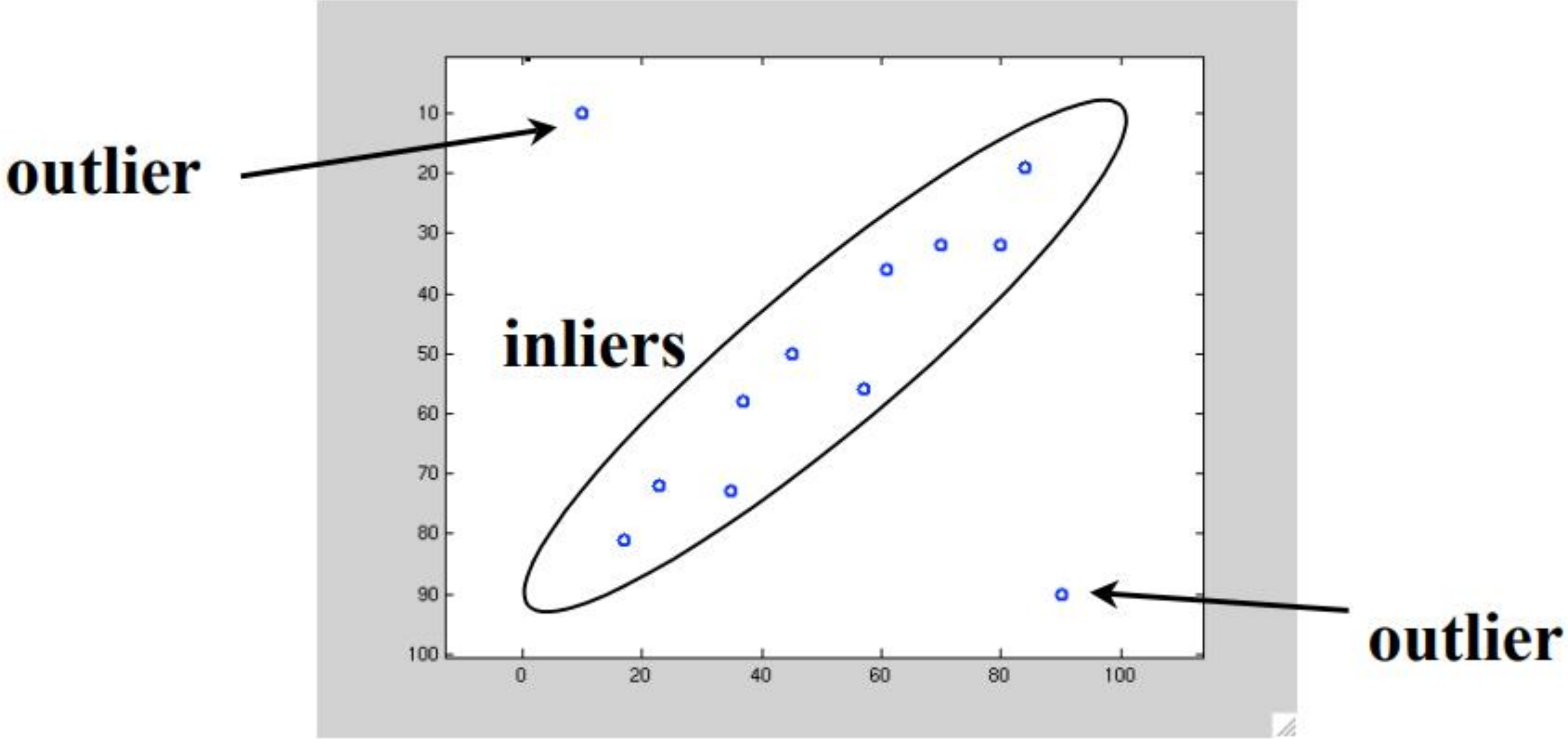
outlier



outlier

Bad Data Outliers

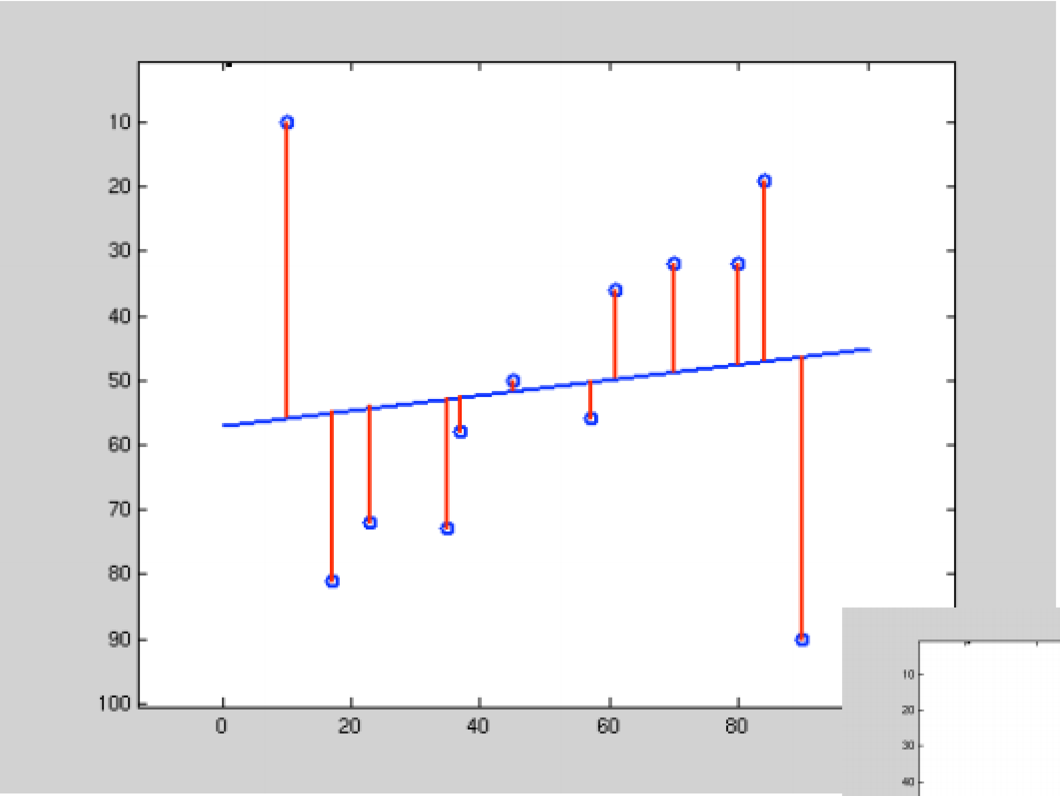
Loosely speaking, outliers are points that don't "fit" the model.
Points that do fit are called "inliers"



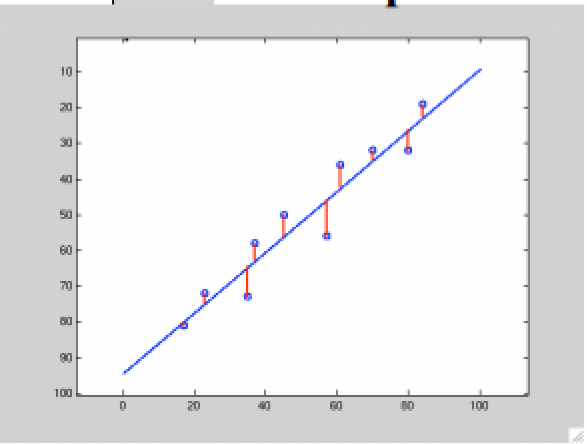
Problem with Outliers

Least squares estimation is sensitive to outliers, so that a few outliers can greatly skew the result.

Least squares regression with outliers

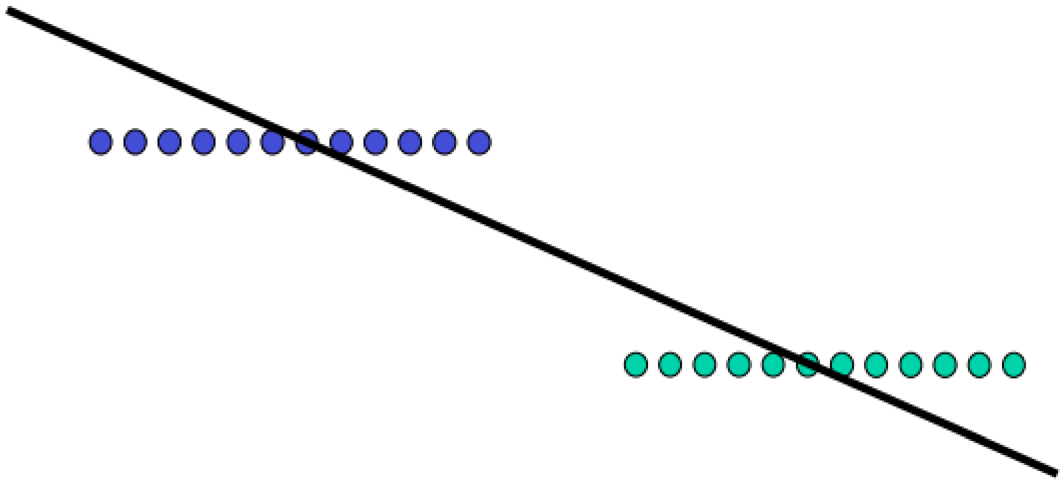


compare



Solution: Estimation methods that are robust to outliers.

Outliers aren’t the only problem



Multiple structures can also skew the results. (the fit procedure implicitly assumes there is only one instance of the model in the data).

RANSAC [Fischler & Bolles 1981]

- RANdom SAmple Consensus

- Approach: we want to avoid the impact of outliers, so let's look for "inliers", and use only those.

- Intuition: if an outlier is chosen to compute the current fit, then the resulting line won't have much support from rest of the points.

Slide credit: Kristen Grauman

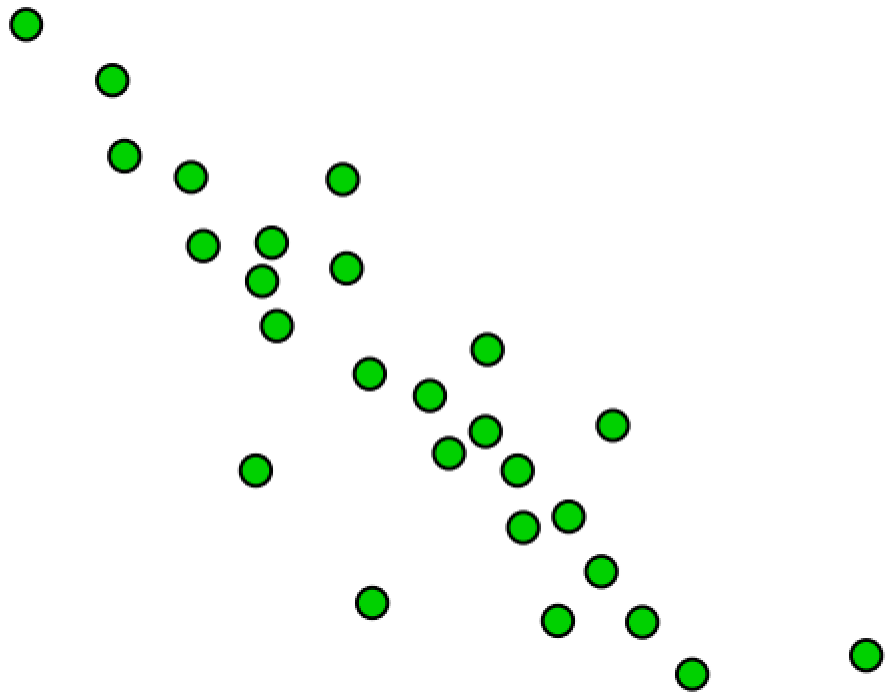
RANSAC Loop

1. Randomly select a *seed group* of points on which to base transformation estimate (e.g., a group of matches)
2. Compute transformation from seed group
3. Find *inliers* to this transformation
4. If the number of inliers is sufficiently large, re-compute least-squares estimate of transformation on all of the inliers

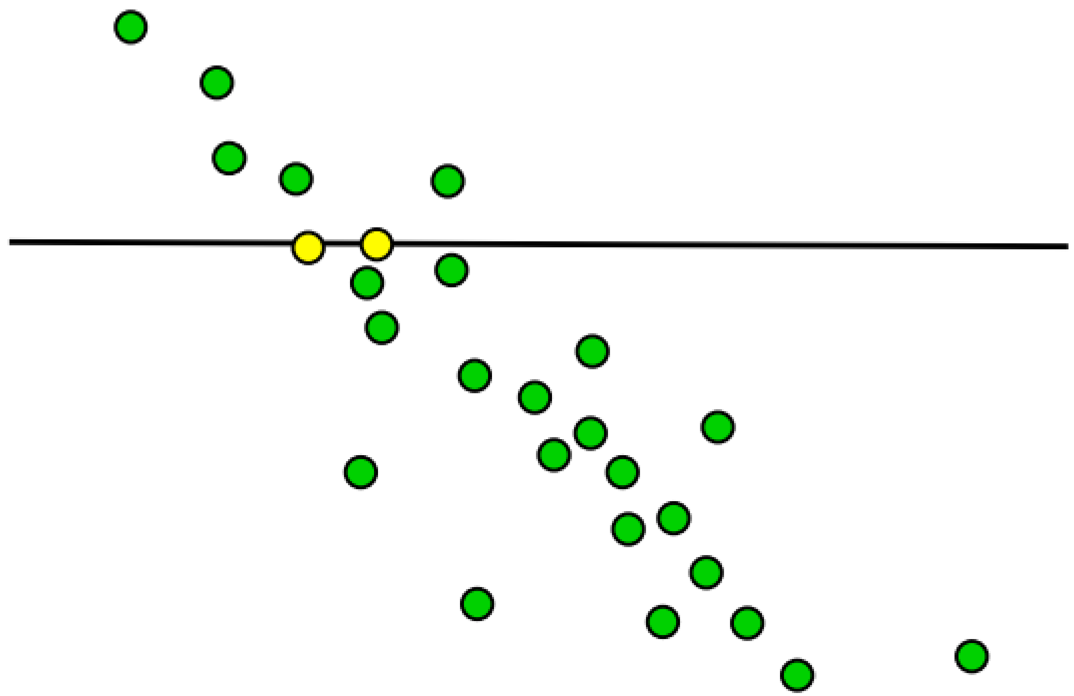
- Keep the transformation with the largest number of inliers

Slide credit: Kristen Grauman

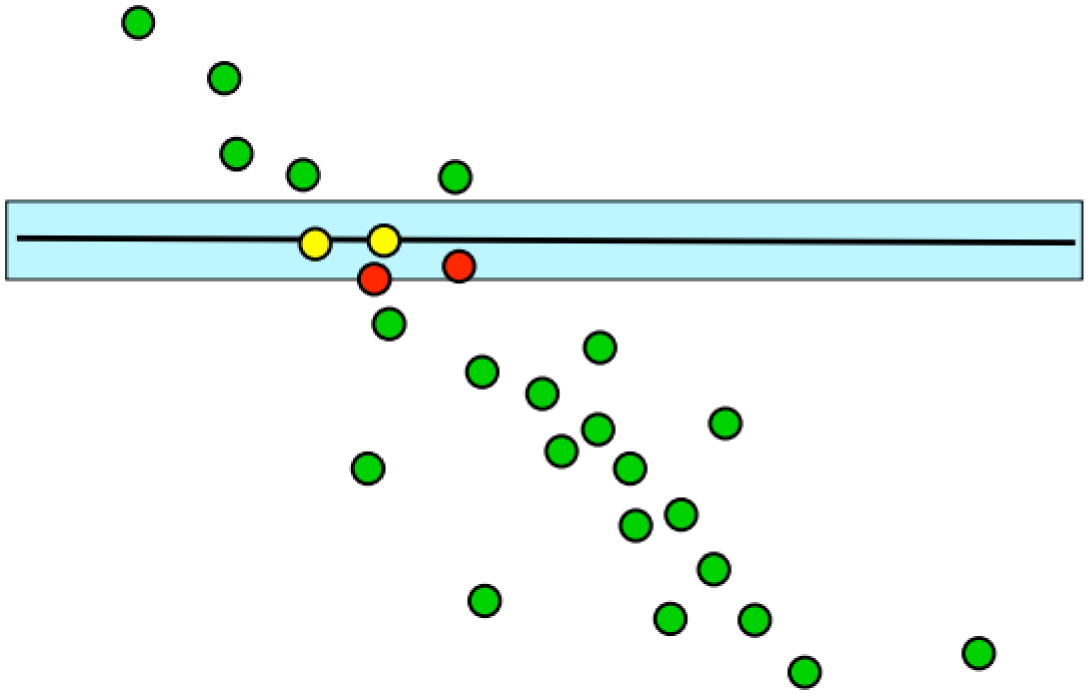
RANSAC Line Fitting Example



RANSAC Line Fitting Example

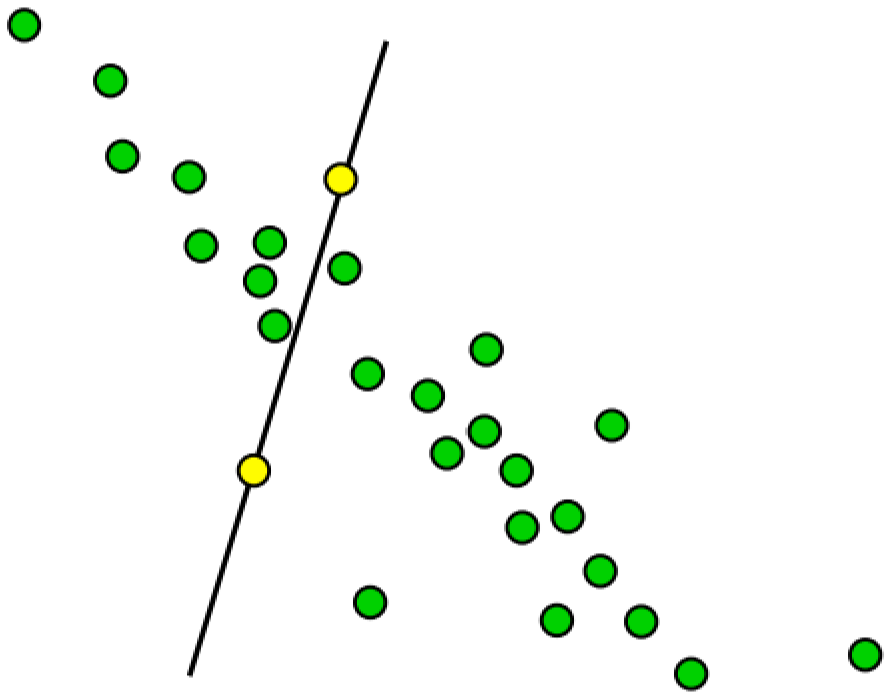


RANSAC Line Fitting Example

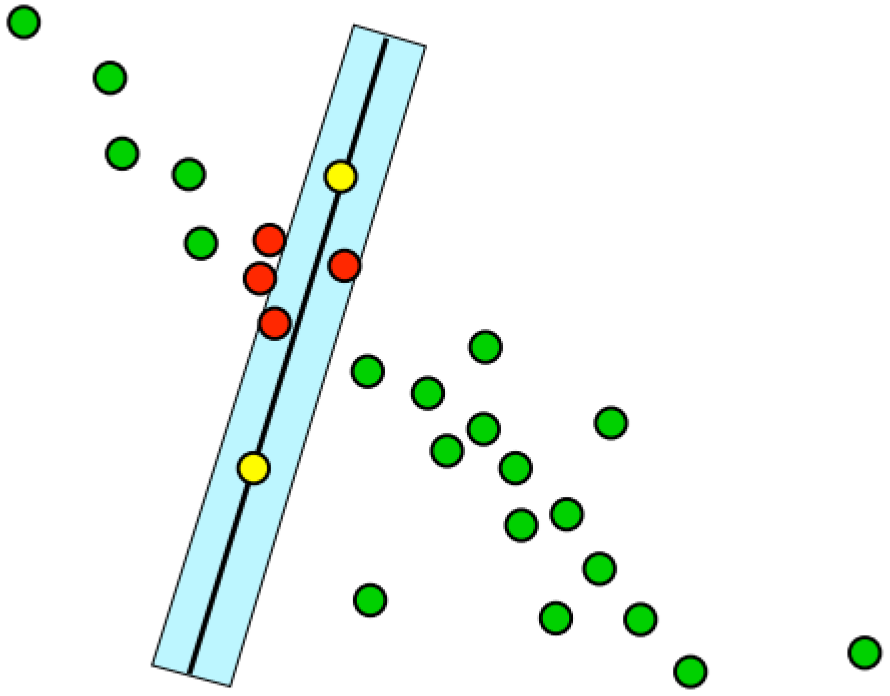


Count = 4

RANSAC Line Fitting Example

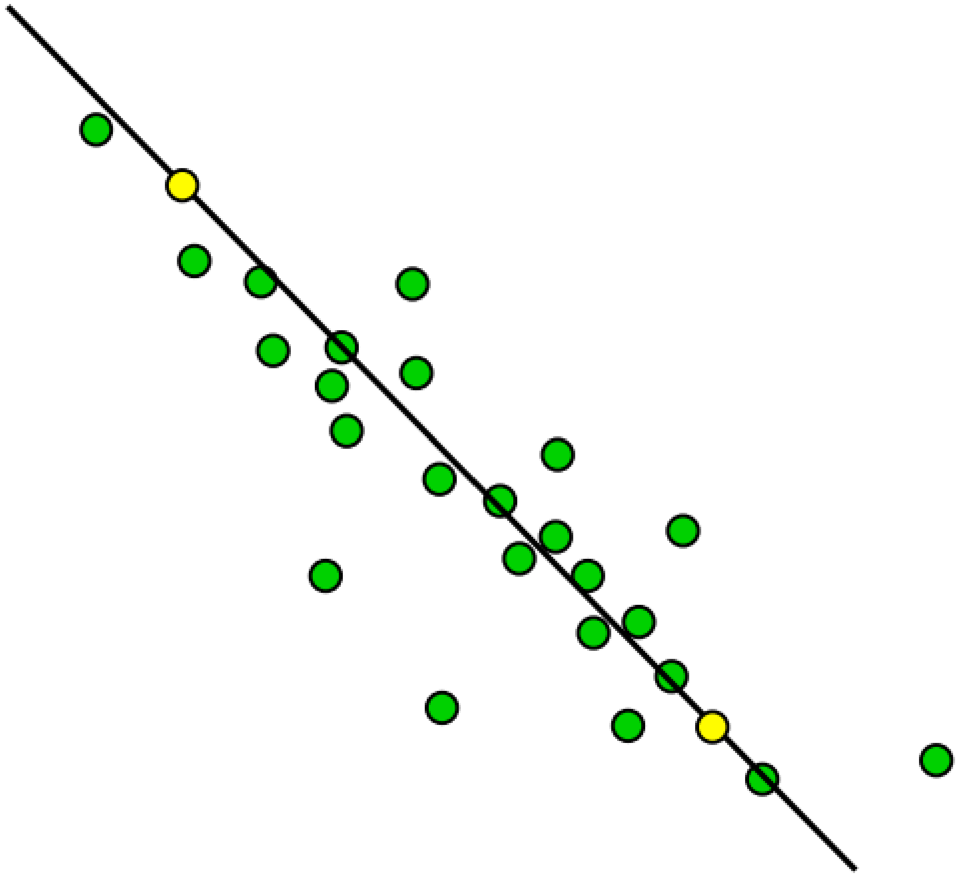


RANSAC Line Fitting Example

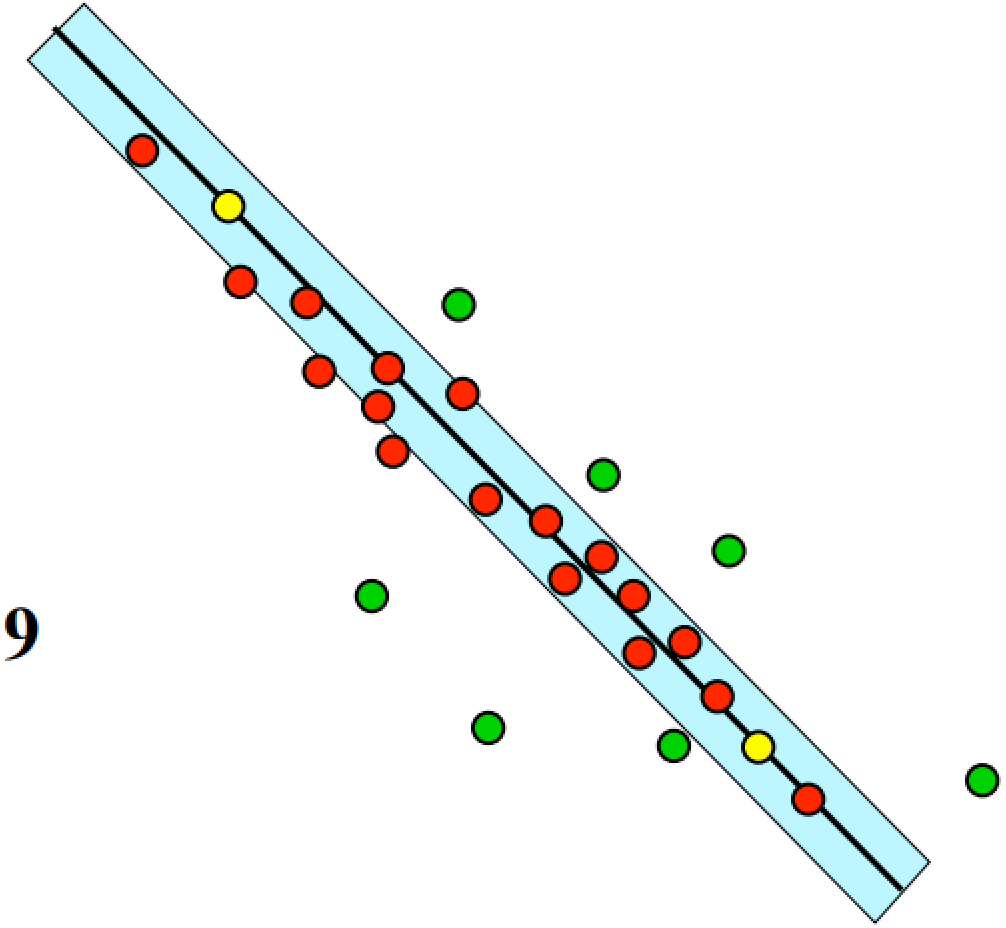


Count = 6

RANSAC Line Fitting Example

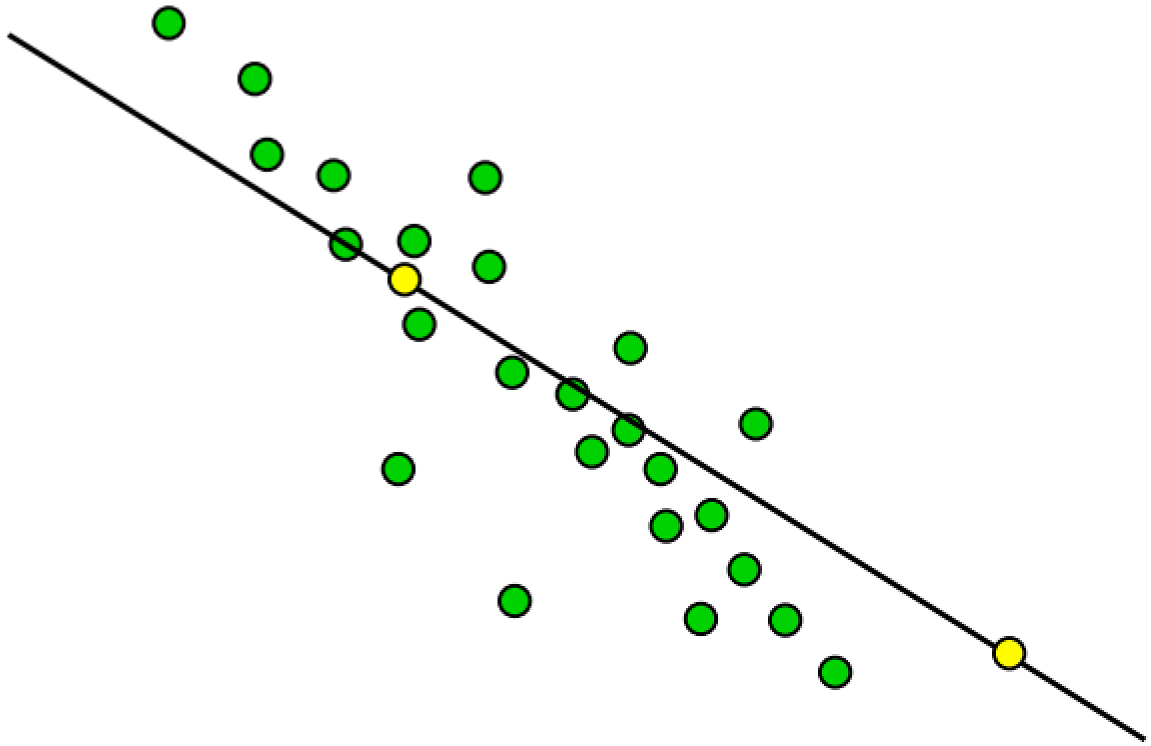


RANSAC Line Fitting Example

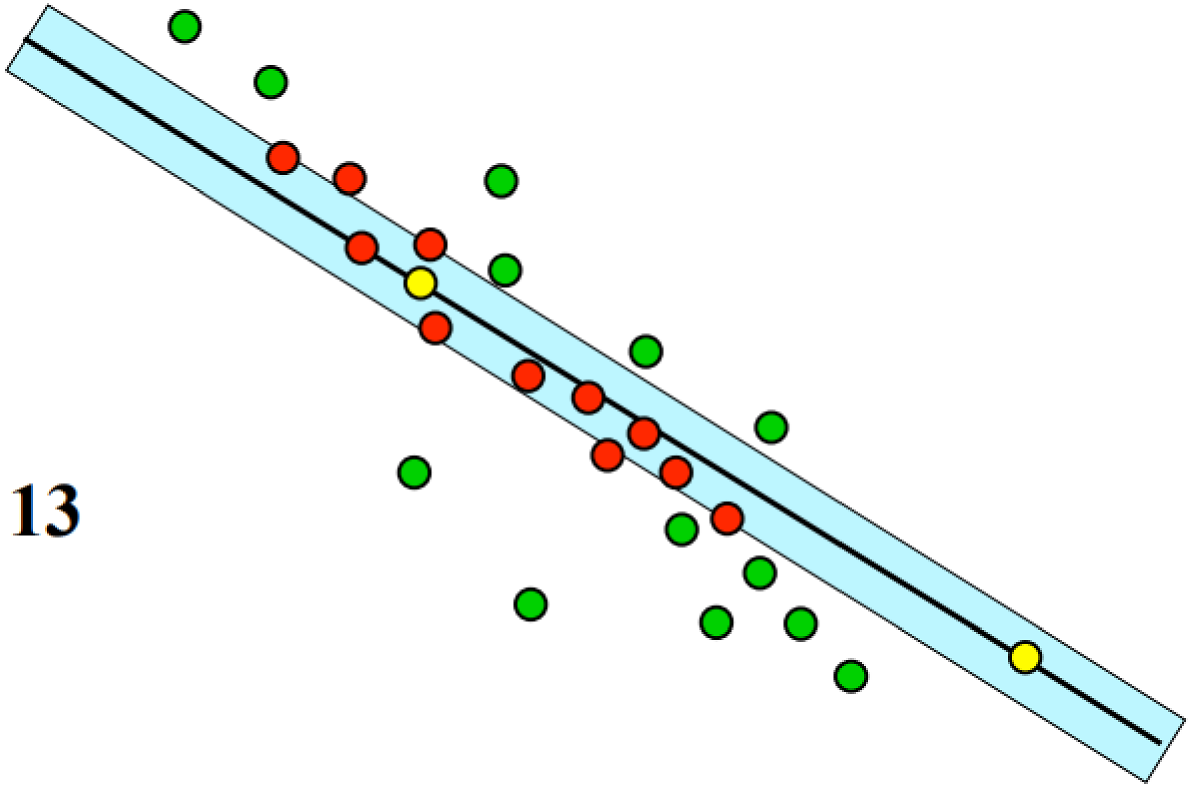


Count = 19

RANSAC Line Fitting Example

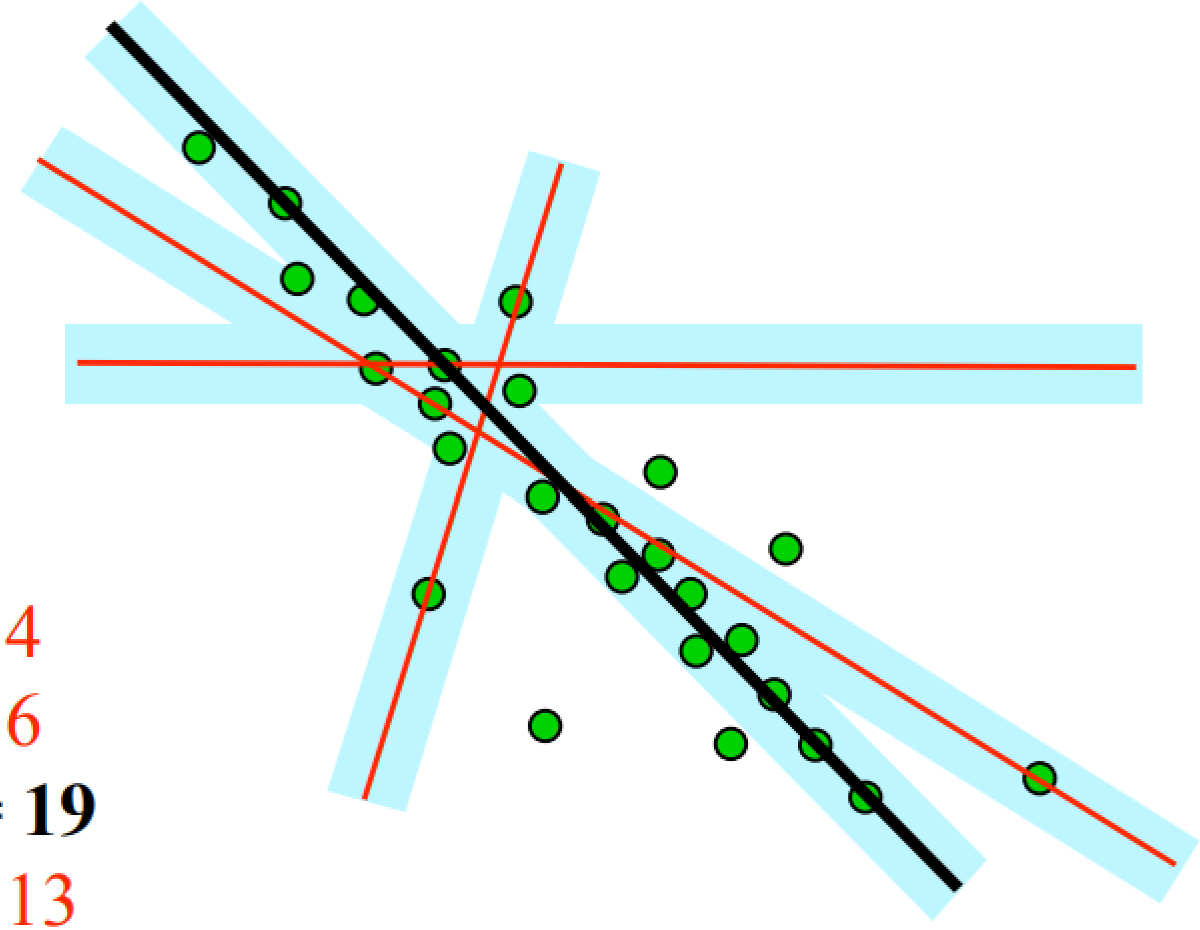


RANSAC Line Fitting Example



Count = 13

RANSAC Line Fitting Example



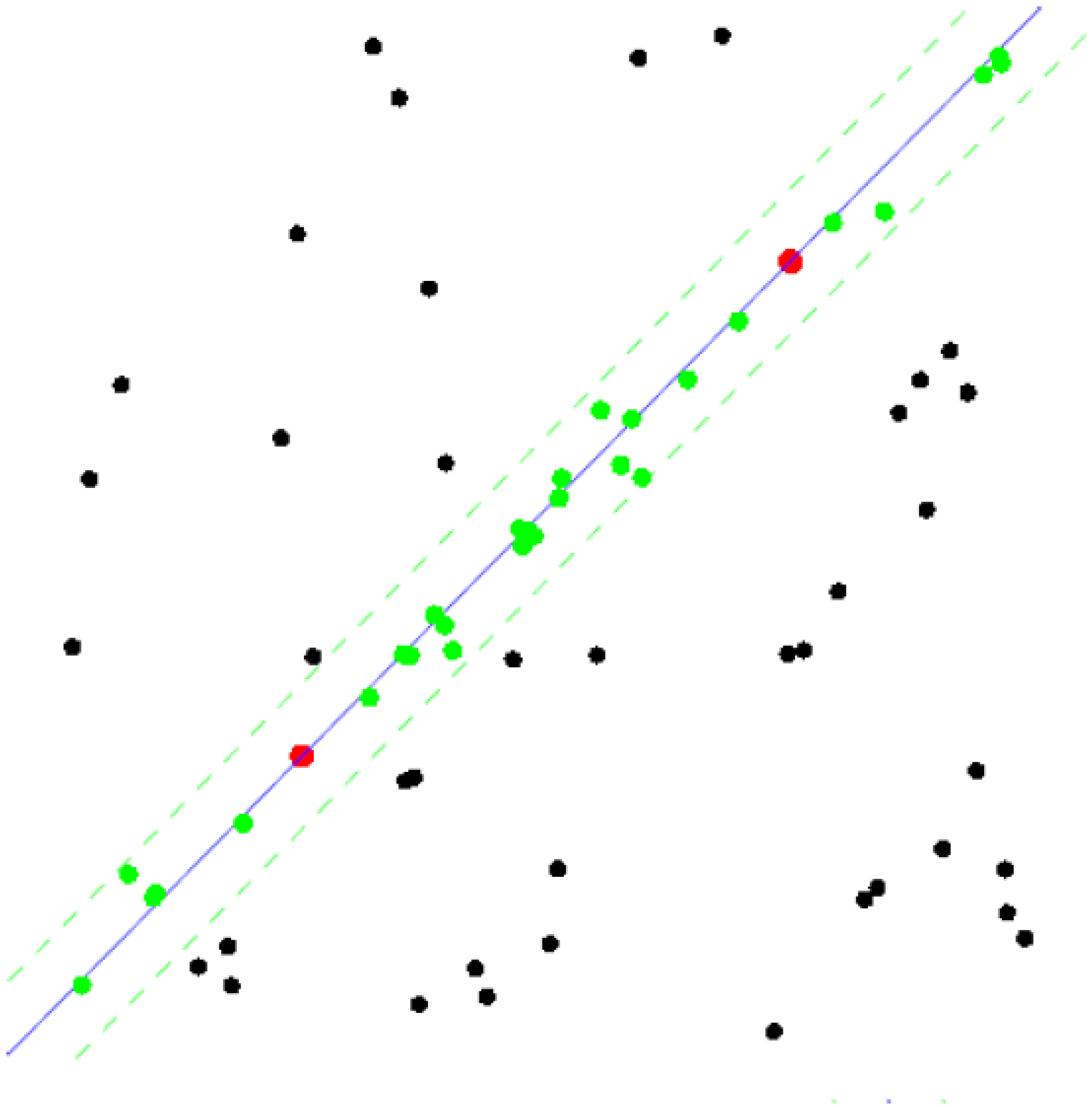
Count = 4

Count = 6

Count = 19

Count = 13

RANSAC Line Fitting Example (2)



$c=15$

RANSAC Algorithm

Algorithm 15.4: RANSAC: fitting lines using random sample consensus

Determine:
- n — the smallest number of points required
- k — the number of iterations required
- t — the threshold used to identify a point that fits well
- d — the number of nearby points required to assert a model fits well

Until k iterations have occurred
- Draw a sample of n points from the data uniformly and at random
- Fit to that set of n points
- For each data point outside the sample
  - Test the distance from the point to the line against t; if the distance from the point to the line is less than t, the point is close
- end
- If there are d or more points close to the line then there is a good fit. Refit the line using all these points.

end

Use the best fit from this collection, using the fitting error as a criterion

Slide credit: Forsyth & Pence

RANSAC

- The algorithm in simple words

1. Randomly select a sample of *s* data points to initiate the model.
2. Determine the set of data points which are within a distance threshold *t* of the model.
3. If the size of *s* (number of inliers) is greater than some threshold *t*, re-estimate the model using all the points in consensus set.
4. After *N* trials the largest consensus set is selected and the model is re-estimated using all the points.

Fischler, M.A, and Bolles, R. C, 1981. Random Sampling Consensus: A paradigm for model fitting with applications to image analysis and automated cartography. Communications of the association for computing machinery, 24(26), 381-395.

How many samples to choose

e = probability that a point is an outlier
s = number of points in a sample
N = number of samples (we want to compute this)
p = desired probability that we get a good sample

Solve the following for N:

$$1 - (1 - (1 - e)^s)^N = p$$

Where in the world did that come from? ….

Let's Dissect

Dissection

e = probability that a point is an outlier
s = number of points in a sample
N = number of samples (we want to compute this)
p = desired probability that we get a good sample

$$1 - (1 - (\underbrace{1 - e}_{\text{Probability that choosing one point yields an inlier}})^s)^N = p$$

Dissection

e = probability that a point is an outlier
s = number of points in a sample
N = number of samples (we want to compute this)
p = desired probability that we get a good sample

$$1 - \left(1 - \underbrace{(1 - e)^s}_{\text{Probability of choosing s inliers in a row (sample only contains inliers)}}\right)^N = p$$

Dissection

e = probability that a point is an outlier
s = number of points in a sample
N = number of samples (we want to compute this)
p = desired probability that we get a good sample

$$1 - \left(\underbrace{1 - (1 - e)^{s}}_{\text{Probability that one or more points in the sample were outliers (sample is contaminated).}}\right)^{N} = p$$

Probability that one or more points in the sample were outliers (sample is contaminated).

Dissection

e = probability that a point is an outlier
s = number of points in a sample
N = number of samples (we want to compute this)
p = desired probability that we get a good sample

$$1 - \underbrace{\left(1 - (1 - e)^{s}\right)^{N}}_{\text{Probability that N samples were contaminated.}} = p$$

Probability that N samples were contaminated.

Dissection

e = probability that a point is an outlier
s = number of points in a sample
N = number of samples (we want to compute this)
p = desired probability that we get a good sample

$$\underbrace{1 - (1 - (1 - e)^{s})^{N}} = p$$

Probability that at least one sample was not contaminated
(at least one sample of s points is composed of only inliers).

How many samples?

Choose N so that, with probability p, at least one random sample is free from outliers. e.g. p=0.99

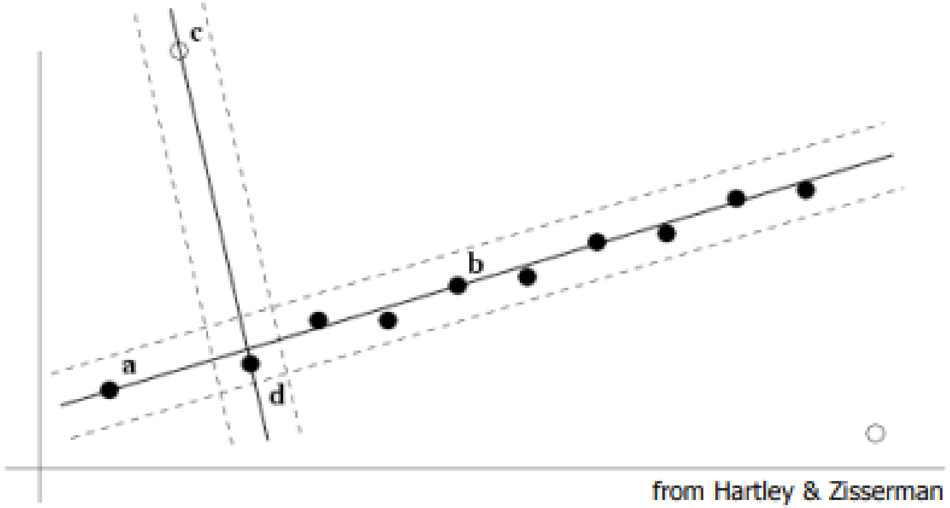
$$\left(1-(1-e)^s\right)^N = 1-p$$

$$N = \frac{\log(1-p)}{\log\left(1-(1-e)^s\right)}$$

	proportion of outliers e						
s	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

Example: N for the line fitting problem

- $n = 12$ points
- Minimal sample size $s = 2$
- 2 outliers: $e = 1/6$ => 20%
- So $N = 5$ gives us a 99% chance of getting a pure-inlier sample
  - Compared to $N = 66$ by trying every pair of points



from Hartley & Zisserman

Acceptable Consensus Set?

- We have seen that we don't have to exhaustively sample subsets of points, we just need to randomly sample N subsets.

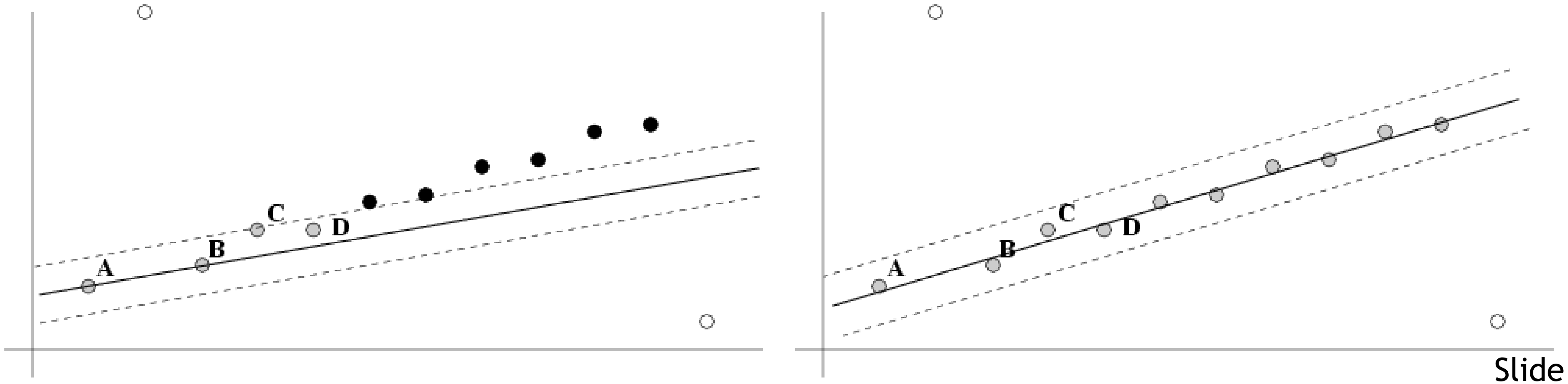
- However, typically, we don't even have to sample N sets!

- Early termination: terminate when inlier ratio reaches expected ratio of inliers

$$T = (1 - e) * (\textit{total number of data points})$$

After RANSAC

- RANSAC divides data into inliers and outliers and yields estimate computed from minimal set of inliers.
- Improve this initial estimate with estimation over all inliers (e.g. with standard least-squares minimization).
- But this may change inliers, so alternate fitting with re-classification as inlier/outlier.



Slide credit: David Lowe

RANSAC: pros and cons

Pros:

General method suited for a wide range of model fitting problems

Easy to implement and easy to calculate its failure rate

Cons:

Only handles a moderate percentage of outliers without cost blowing up

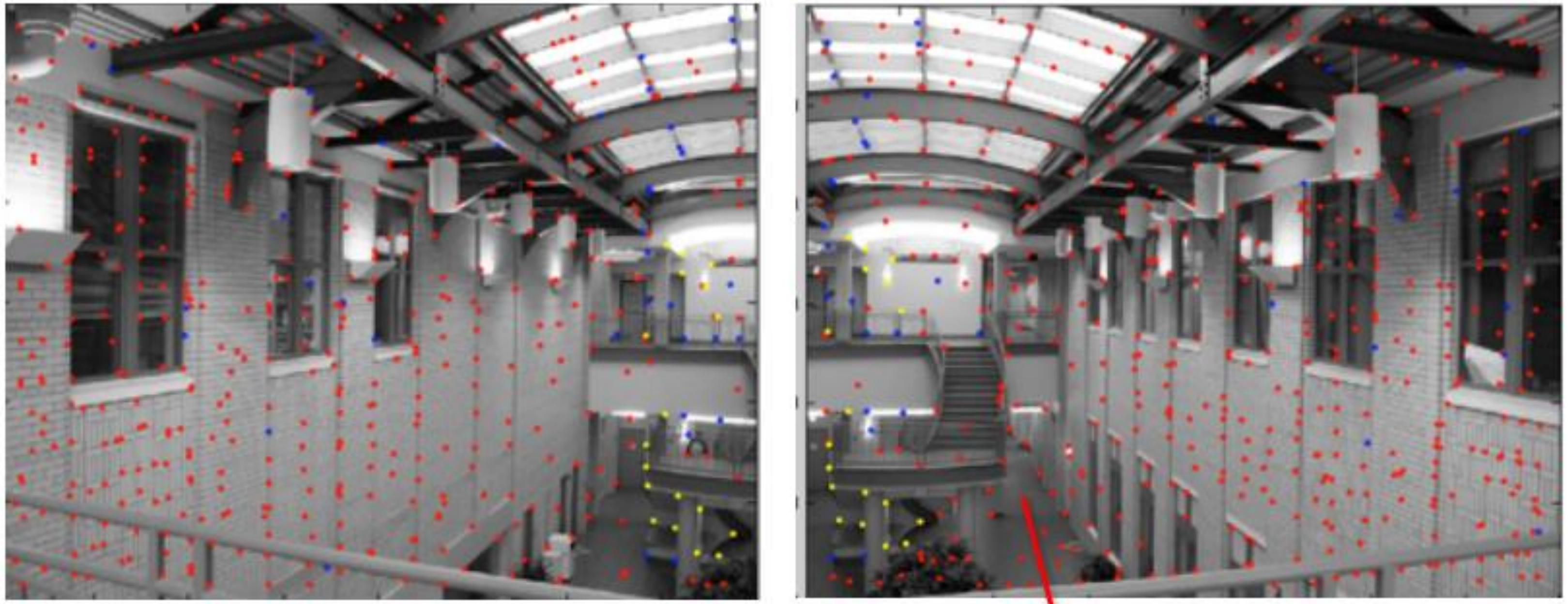
Many real problems have high rate of outliers (but sometimes selective choice of random subsets can help)

A voting strategy, The Hough transform, can handle high percentage of outliers

Applications

- Video Stabilization
- Image Stitching
- Image Mosaicking
- Line Fitting
- And more…

Case study: RANSAC for matching



Red:
    rejected by 2nd nearest neighbor criterion
Blue:
    Ransac outliers
Yellow:
    inliers

Demo (MATLAB)

```matlab
%% RANSAC DEMO 2

close all
clear all

%%
original  = imread('cameraman.tif');
imshow(original);
title('Base image');

distorted = imresize(original,0.7);
distorted = imrotate(distorted,31);
figure; imshow(distorted);
title('Transformed image');

%% Detect and extract features from both images

ptsOriginal  = detectSURFFeatures(original);
ptsDistorted = detectSURFFeatures(distorted);
[featuresOriginal,validPtsOriginal] = ...
    extractFeatures(original,ptsOriginal);
[featuresDistorted,validPtsDistorted] = ...
    extractFeatures(distorted,ptsDistorted);

%% Match Features
index_pairs = matchFeatures(featuresOriginal,featuresDistorted);
matchedPtsOriginal  = validPtsOriginal(index_pairs(:,1));
matchedPtsDistorted = validPtsDistorted(index_pairs(:,2));
figure;
showMatchedFeatures(original,distorted,...
    matchedPtsOriginal,matchedPtsDistorted);
title('Matched SURF points,including outliers');
```

```matlab
%% Exclude the outliers and compute the transformation matrix
[tform,inlierPtsDistorted,inlierPtsOriginal] = ...
    estimateGeometricTransform(matchedPtsDistorted,matchedPtsOriginal,...
    'similarity');
figure;

showMatchedFeatures(original,distorted,...
    inlierPtsOriginal,inlierPtsDistorted);
title('Matched inlier points');

%% Recover the original image from distorted image
outputView = imref2d(size(original));
Ir = imwarp(distorted,tform,'OutputView',outputView);
figure; imshow(Ir);
title('Recovered image');
```

Demo (Python)

```python
import numpy as np
from matplotlib import pyplot as plt

from skimage import data
from skimage.util import img_as_float
from skimage.feature import (corner_harris, corner_subpix, corner_peaks,
                             plot_matches)
from skimage.transform import warp, AffineTransform
from skimage.exposure import rescale_intensity
from skimage.color import rgb2gray
from skimage.measure import ransac


# generate synthetic checkerboard image and add gradient for the later matching
checkerboard = img_as_float(data.checkerboard())
img_orig = np.zeros(list(checkerboard.shape) + [3])
img_orig[..., 0] = checkerboard
gradient_r, gradient_c = (np.mgrid[0:img_orig.shape[0],
                                   0:img_orig.shape[1]]
                          / float(img_orig.shape[0]))
img_orig[..., 1] = gradient_r
img_orig[..., 2] = gradient_c
img_orig = rescale_intensity(img_orig)
img_orig_gray = rgb2gray(img_orig)

# warp synthetic image
tform = AffineTransform(scale=(0.9, 0.9), rotation=0.2, translation=(20, -10))
img_warped = warp(img_orig, tform.inverse, output_shape=(200, 200))
img_warped_gray = rgb2gray(img_warped)

# extract corners using Harris' corner measure
coords_orig = corner_peaks(corner_harris(img_orig_gray), threshold_rel=0.001,
                           min_distance=5)
coords_warped = corner_peaks(corner_harris(img_warped_gray),
                             threshold_rel=0.001, min_distance=5)

# determine sub-pixel corner position
coords_orig_subpix = corner_subpix(img_orig_gray, coords_orig, window_size=9)
coords_warped_subpix = corner_subpix(img_warped_gray, coords_warped,
                                     window_size=9)
```

```python
def gaussian_weights(window_ext, sigma=1):
    y, x = np.mgrid[-window_ext:window_ext+1, -window_ext:window_ext+1]
    g = np.zeros(y.shape, dtype=np.double)
    g[:] = np.exp(-0.5 * (x**2 / sigma**2 + y**2 / sigma**2))
    g /= 2 * np.pi * sigma * sigma
    return g


def match_corner(coord, window_ext=5):
    r, c = np.round(coord).astype(np.intp)
    window_orig = img_orig[r-window_ext:r+window_ext+1,
                           c-window_ext:c+window_ext+1, :]

    # weight pixels depending on distance to center pixel
    weights = gaussian_weights(window_ext, 3)
    weights = np.dstack((weights, weights, weights))

    # compute sum of squared differences to all corners in warped image
    SSDs = []
    for cr, cc in coords_warped:
        window_warped = img_warped[cr-window_ext:cr+window_ext+1,
                                   cc-window_ext:cc+window_ext+1, :]
        SSD = np.sum(weights * (window_orig - window_warped)**2)
        SSDs.append(SSD)

    # use corner with minimum SSD as correspondence
    min_idx = np.argmin(SSDs)
    return coords_warped_subpix[min_idx]


# find correspondences using simple weighted sum of squared differences
src = []
dst = []
for coord in coords_orig_subpix:
    src.append(coord)
    dst.append(match_corner(coord))
src = np.array(src)
dst = np.array(dst)


# estimate affine transform model using all coordinates
model = AffineTransform()
model.estimate(src, dst)

# robustly estimate affine transform model with RANSAC
model_robust, inliers = ransac((src, dst), AffineTransform, min_samples=3,
                               residual_threshold=2, max_trials=100)
```

```python
outliers = inliers == False


# compare "true" and estimated transform parameters
print("Ground truth:")
print(f"Scale: ({tform.scale[1]:.4f}, {tform.scale[0]:.4f}), "
      f"Translation: ({tform.translation[1]:.4f}, "
      f"{tform.translation[0]:.4f}), "
      f"Rotation: {-tform.rotation:.4f}")
print("Affine transform:")
print(f"Scale: ({model.scale[0]:.4f}, {model.scale[1]:.4f}), "
      f"Translation: ({model.translation[0]:.4f}, "
      f"{model.translation[1]:.4f}), "
      f"Rotation: {model.rotation:.4f}")
print("RANSAC:")
print(f"Scale: ({model_robust.scale[0]:.4f}, {model_robust.scale[1]:.4f}), "
      f"Translation: ({model_robust.translation[0]:.4f}, "
      f"{model_robust.translation[1]:.4f}), "
      f"Rotation: {model_robust.rotation:.4f}")

# visualize correspondence
fig, ax = plt.subplots(nrows=2, ncols=1)

plt.gray()

inlier_idxs = np.nonzero(inliers)[0]
plot_matches(ax[0], img_orig_gray, img_warped_gray, src, dst,
             np.column_stack((inlier_idxs, inlier_idxs)), matches_color='b')
ax[0].axis('off')
ax[0].set_title('Correct correspondences')

outlier_idxs = np.nonzero(outliers)[0]
plot_matches(ax[1], img_orig_gray, img_warped_gray, src, dst,
             np.column_stack((outlier_idxs, outlier_idxs)), matches_color='r')
ax[1].axis('off')
ax[1].set_title('Faulty correspondences')

plt.show()
```

- Questions
  - Feel Free to ask