Initial Applications

Image Processing Lecture 1

Nicholas Dwork

Images

Recall that images are 2D arrays of numbers.

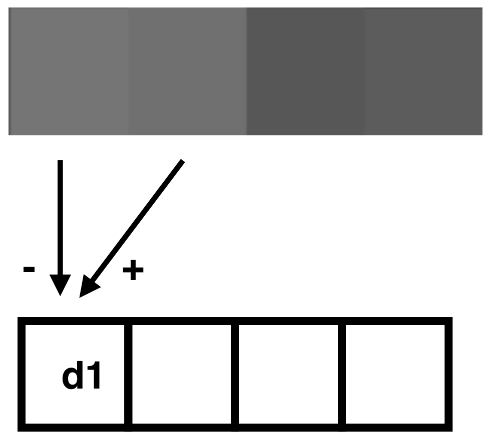
48	47	60	71	121	105	64	89	125	97	18	108
43	49	59	49	72	63	60	90	65	78	54	103
51	65	59	85	123	118	92	61	50	65	77	128
61	74	71	50	44	75	104	109	93	59	49	143
79	82	88	38	57	71	21	40	106	64	45	145
94	100	106	57	76	94	61	34	109	61	65	152
105	123	99	68	98	93	69	74	111	65	82	160
132	153	102	128	153	146	91	141	141	74	96	167
144	136	76	107	154	173	119	131	150	102	109	171
123	125	112	88	92	96	109	121	119	117	114	170
154	153	160	158	152	153	156	152	147	146	143	186

We can do math with them!

First order difference

Old Image

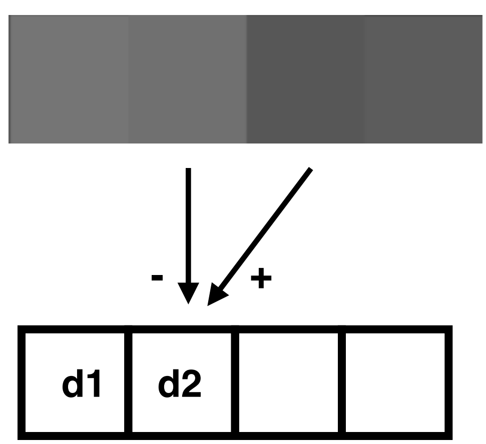
New Image



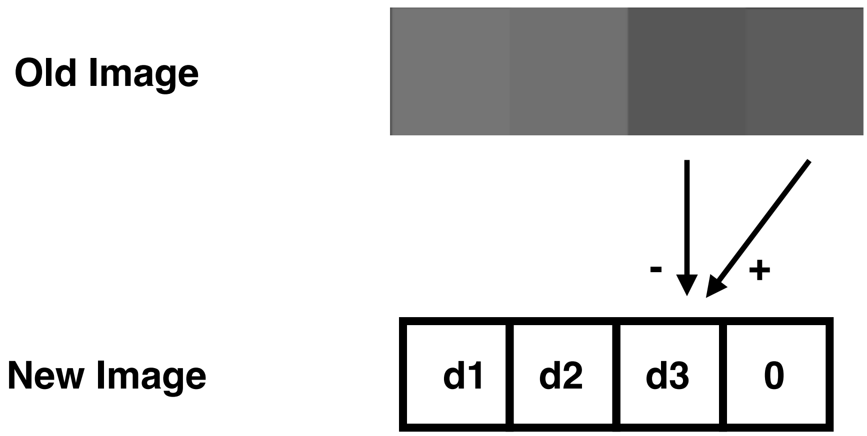
First order difference

Old Image

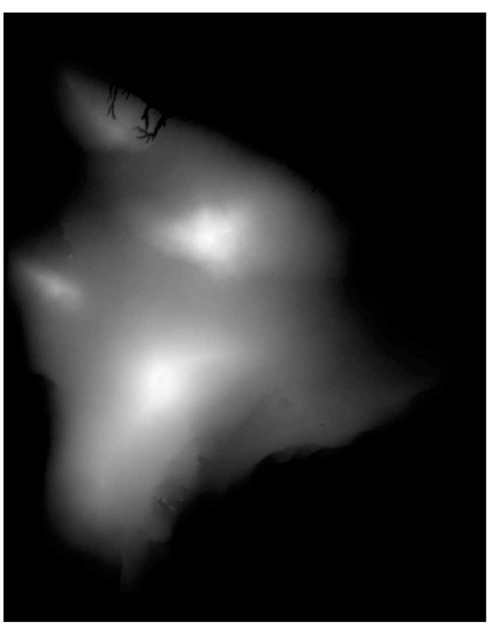
New Image



First Order Difference



Digital Elevation Map



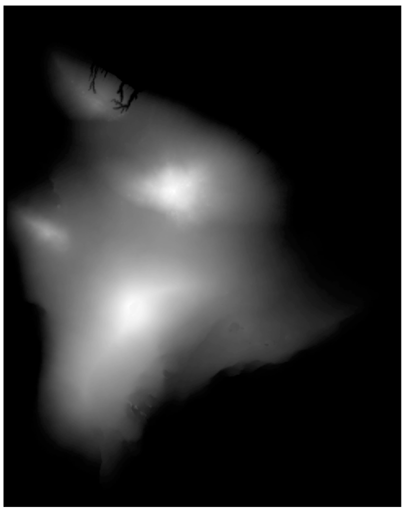
Each pixel is a number designating the location's height.

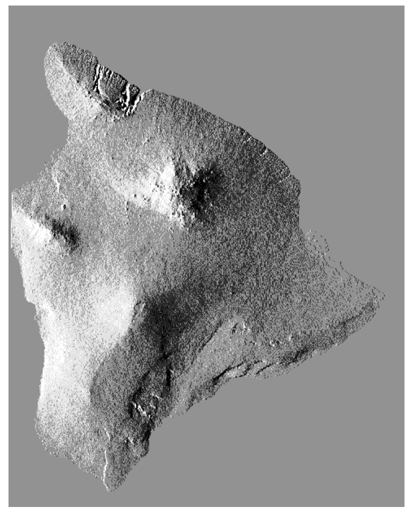
The brighter the pixel, the higher the point.

Hawaii

Relief Distortion Map

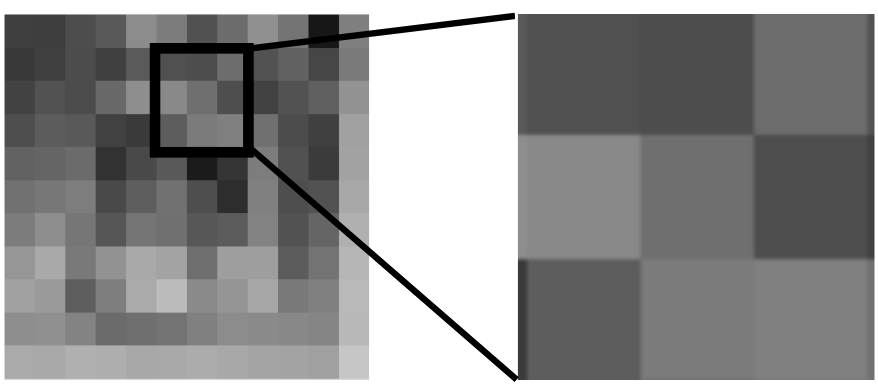
First order difference applied to Digital Elevation Map





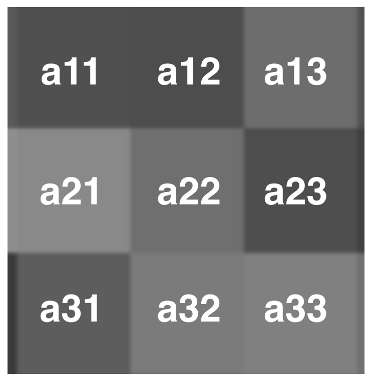
7

Isolating a small region



Mean of Square Region

Mean = $\frac{1}{9}(a_{11} + a_{12} + a_{13} + a_{21}$
$+ a_{22} + a_{23} + a_{31} + a_{32} + a_{33})$



Mean Filtering

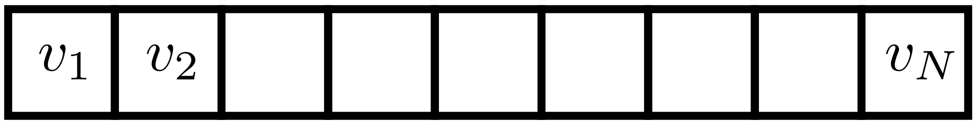
Replace each pixel with its local mean.

25x25 pixel kernel

Also called "Box Car Averaging"

Mean

Image with values v **:**

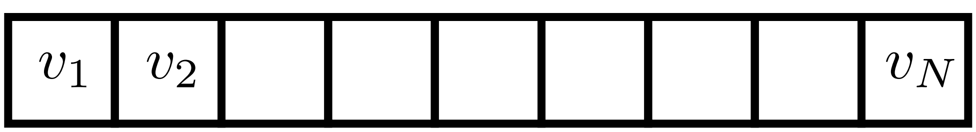


Weights:

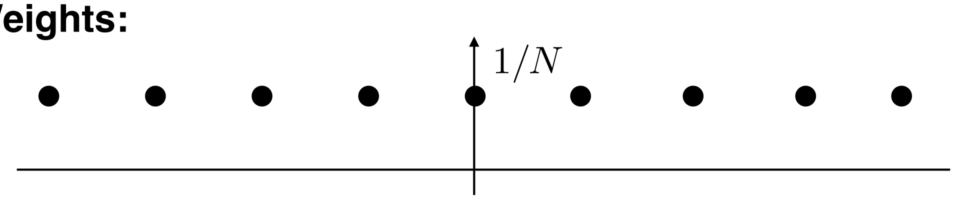
$1/N \quad 1/N \quad 1/N \quad \cdots \quad 1/N$

Mean

Image with values v **:**



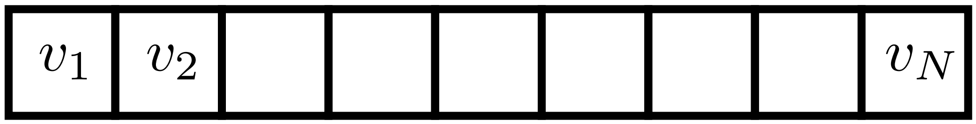
Weights:



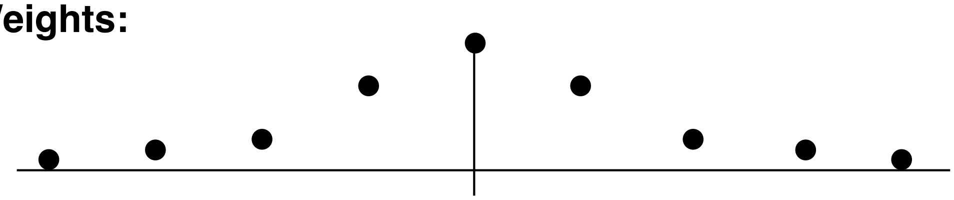
This seems weird. The values in the middle should matter more than values far away.

Weighted Mean

Rather than weighting each point equally, weight them differently.

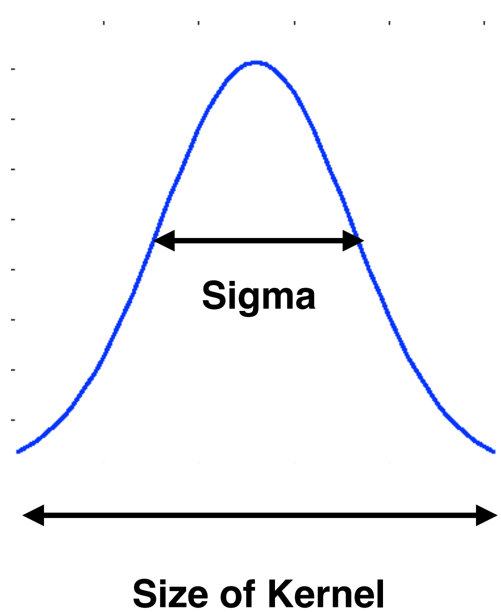


Weights:



Modifying the weights can solve this.
A Gaussian function is a good choice (fspecial **in Matlab).**

Gaussian Function



Sigma tells you how flat the weights are.
The higher the sigma, the flatter the weights.

The size of the kernel tells you how many pixels you're including.

Weighted Mean Filtering

Box Car Filter

25x25 pixel kernel

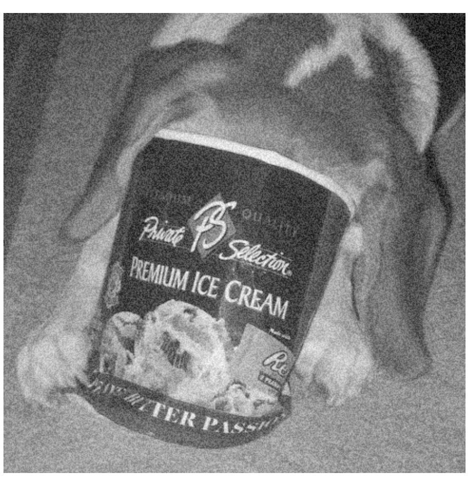
Gaussian Filter

25x25 pixel kernel
sigma = 5

Gaussian Filtering retains a lot more of the information.

Image Denoising

Noisy Image

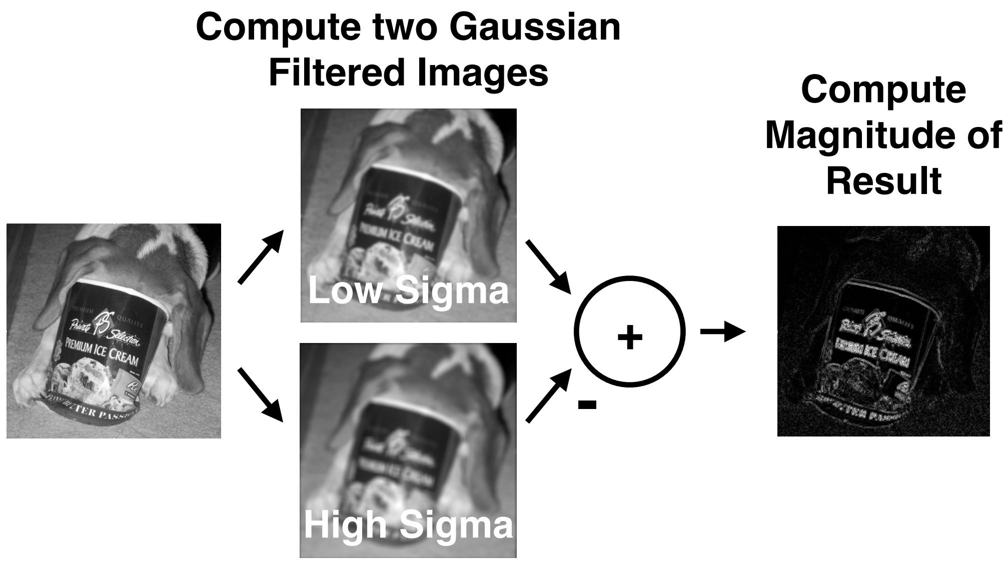


Gaussian Filter

9x9 pixel kernel
sigma = 3

Difference of Gaussians

To find features automatically, we will use this algorithm.



Interesting pixels are bright.

We can use some of these pixels as feature points.

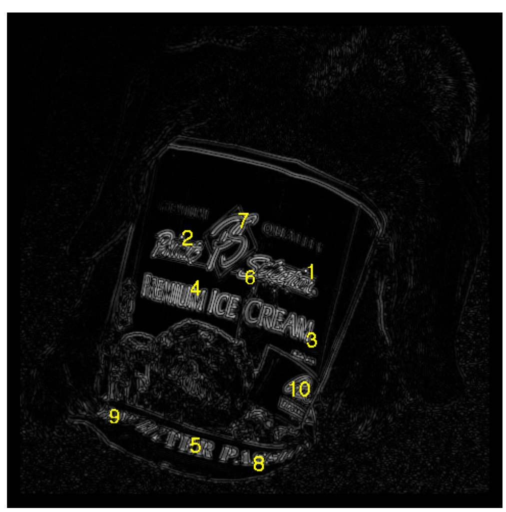
Finding Feature Points

1) **Zero out the region near the border of the image.**
   These points don't make good features.

2) **Find the brightest point in the DoG image.**
   This is your first feature point.
   Use ind2sub**.**

3) **You don't want points that are too close to your current point.**
   Set the DoG image to zero anywhere close to your feature.

4) **Return to step 2.**
   Do this until you get the number of features you want.



Tracking Features

Now that we've found features, we need to track them into the other image.

That is, we want to find those same features in the next image.

Metric of Fit

We can use $||\vec{a} - \vec{b}||$ as a metric of how well two regions of pixels match.

a11	a12	a13
a21	a22	a23
a31	a32	a33

b11	b12	b13
b21	b22	b23
b31	b32	b33

The value of the metric in this case will be high.
The best value possible is 0.

Tracking the Feature

Identify a small region around the feature, called the kernel (e.g. 15 x 15).

Img 1 **Img 2**

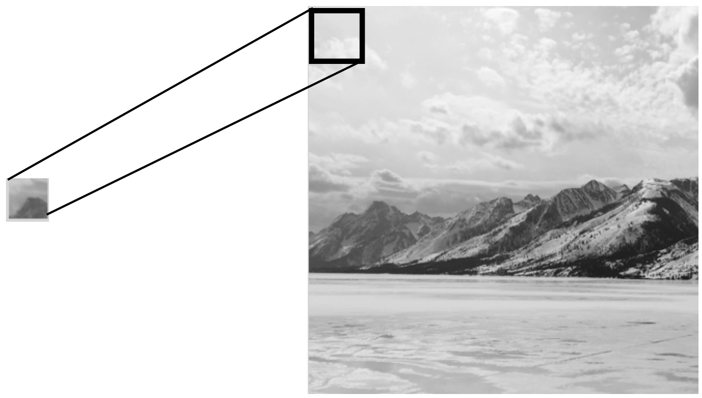
Tracking the Feature

Identify a larger search region centered on the feature in the second image.

Img 1 **Img 2**

Tracking the Feature

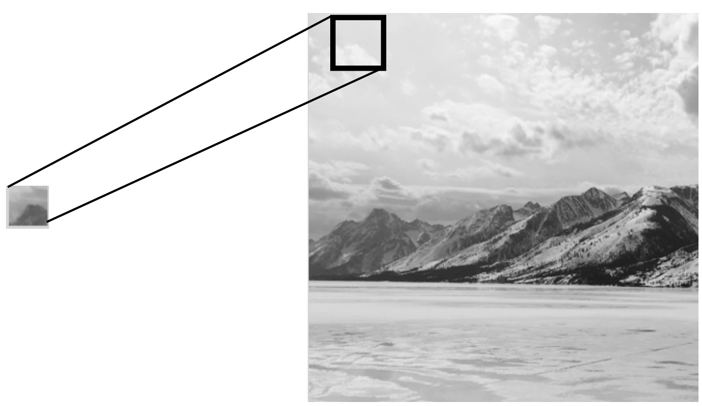
Calculate the metric of fit between the kernel and every possible subset in the Search Space.



Feature **Search Space**

Tracking the Feature

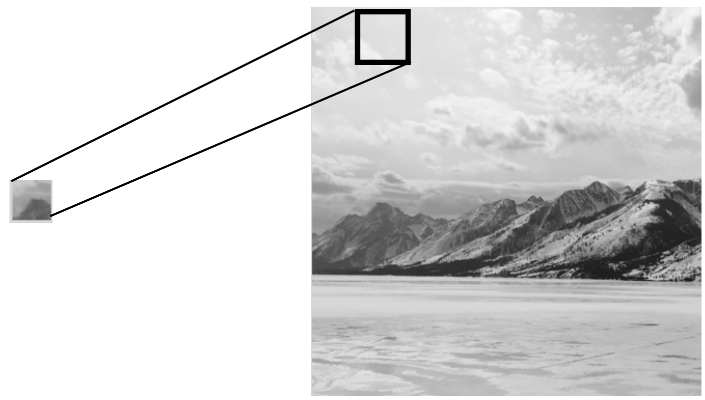
Calculate the metric of fit between the kernel and every possible subset in the Search Space.



Feature **Search Space**

Tracking the Feature

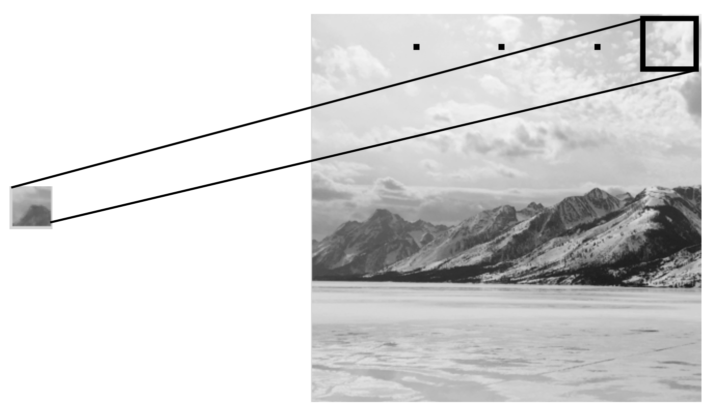
Calculate the metric of fit between the kernel and every possible subset in the Search Space.



Feature **Search Space**

Tracking the Feature

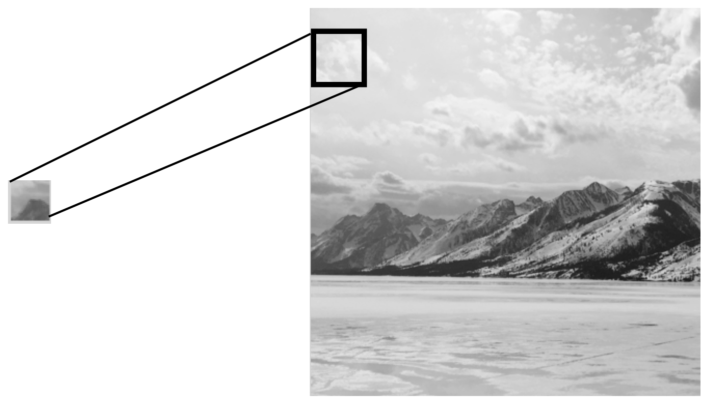
Calculate the metric of fit between the kernel and every possible subset in the Search Space.



Feature **Search Space**

Tracking the Feature

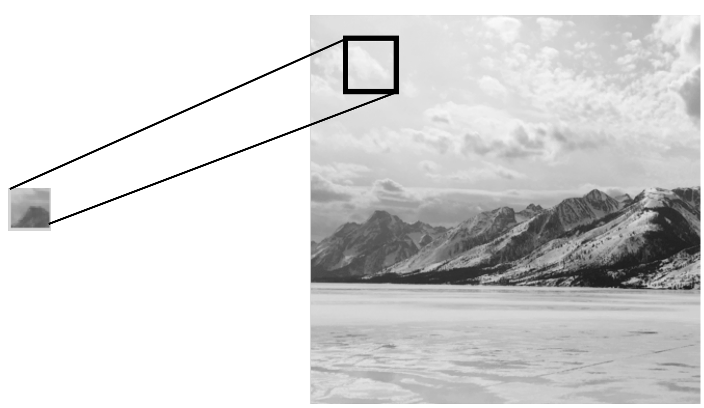
Calculate the metric of fit between the kernel and every possible subset in the Search Space.



Feature **Search Space**

Tracking the Feature

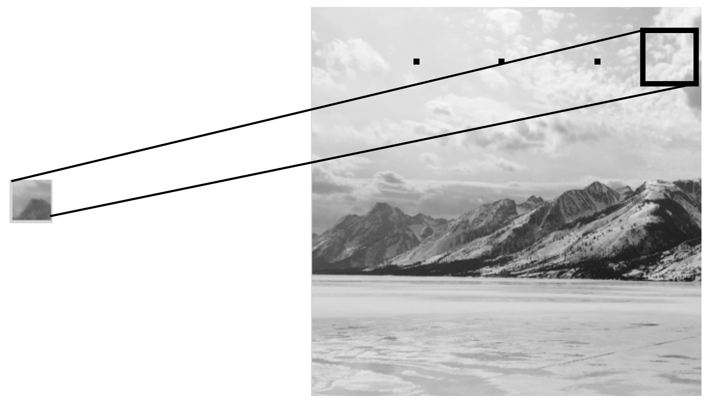
Calculate the metric of fit between the kernel and every possible subset in the Search Space.



Feature **Search Space**

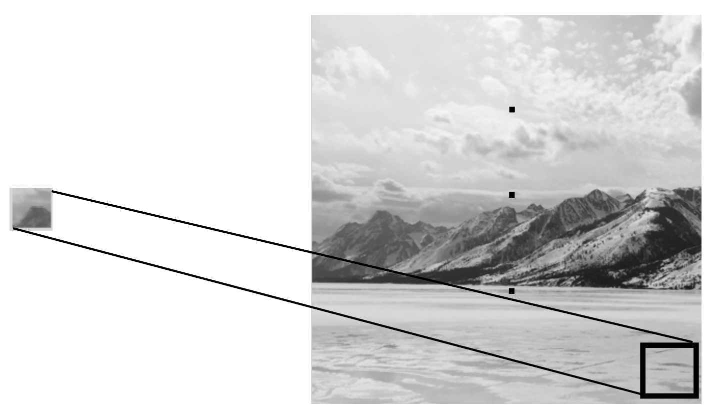
Tracking the Feature

Calculate the metric of fit between the kernel and every possible subset in the Search Space.



Tracking the Feature

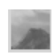
Calculate the metric of fit between the kernel and every possible subset in the Search Space.



Tracking the Feature

We have filtered the search space with the feature image.

The minimum of the metric image is the feature's location.



Feature **Metric Image**

Rejecting Bad Matches

Sometimes our algorithm will make errors in the tracking.

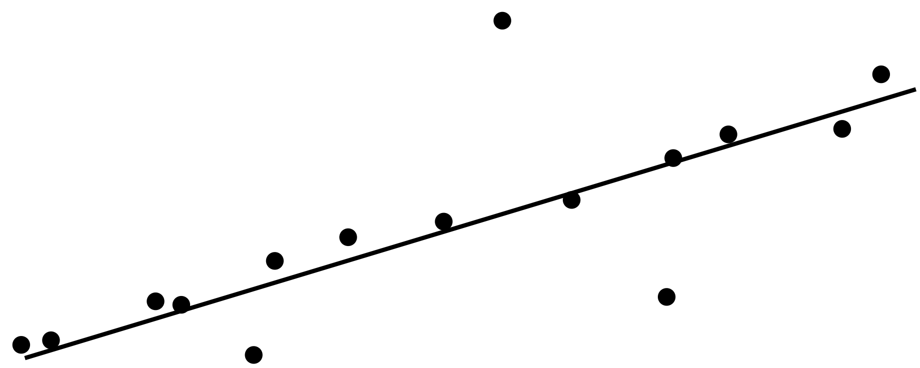
We need a way to reject these outliers.

The RANSAC algorithm is a way of figuring out which features are matched well and which are erroneous.

Measuring Points on a Line

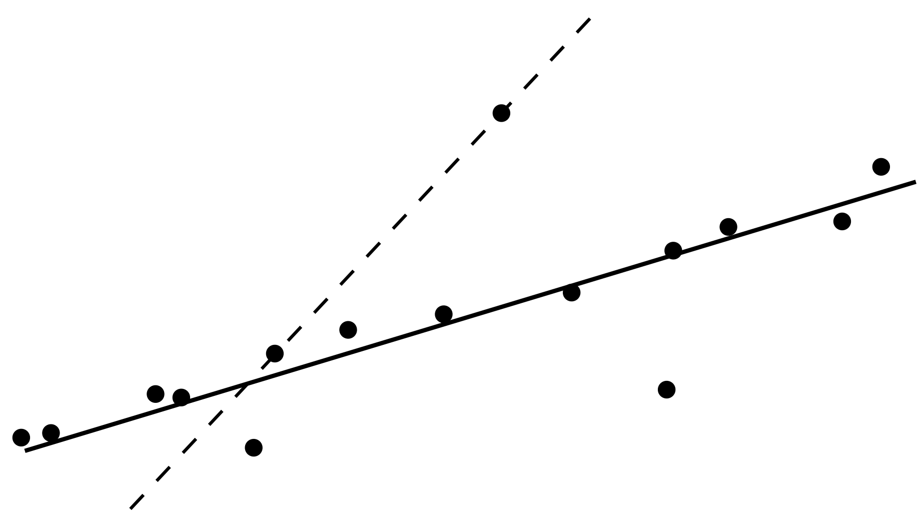
We measure points on a line. Due to noise, the points don't lie on the line exactly. Some are very bad.

Goal: Find the line.



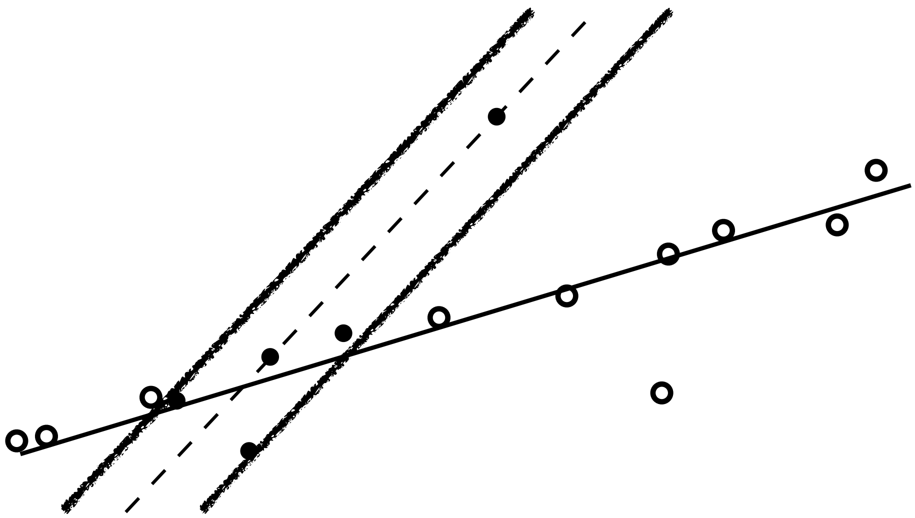
RANSAC

1) Choose two points randomly. Draw the line between those points.



RANSAC

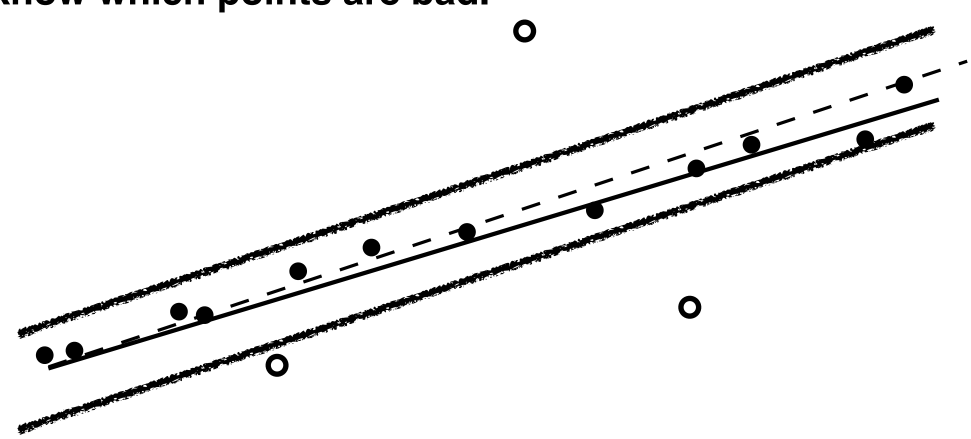
2) Count the number of points that lie within a threshold distance of that new line.



RANSAC

3) Return to step 2.

4) Iterate many times. The line with the highest number of points is the best estimate! We also know which points are bad.



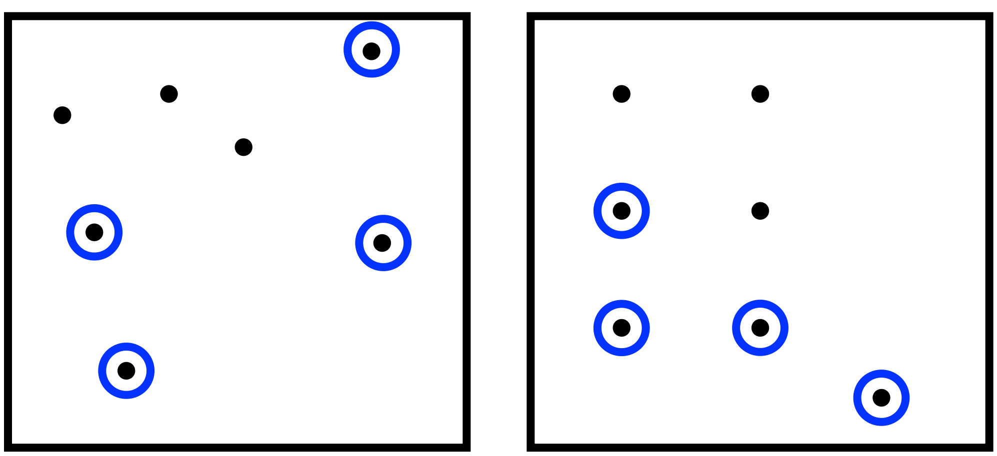
We have been discussing how to use RANSAC to find which points belong to a line.

We will now use RANSAC to identify which features were poorly tracked.

A homography will take the place of a line.
Key: a minimum of four matched points are required to determine a homography.

RANSAC with Features

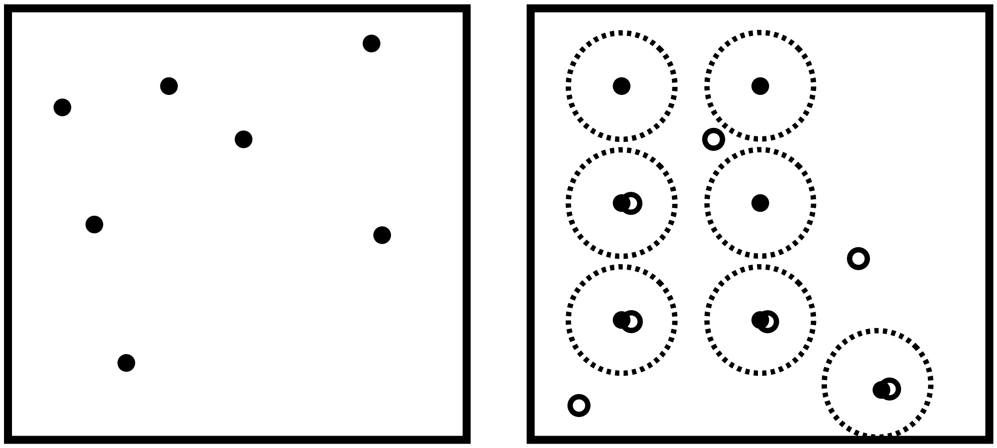
1) Choose four matched points randomly. Determine the homography for those points.



RANSAC with Features

2) Project the points from Image 1 into Image 2.

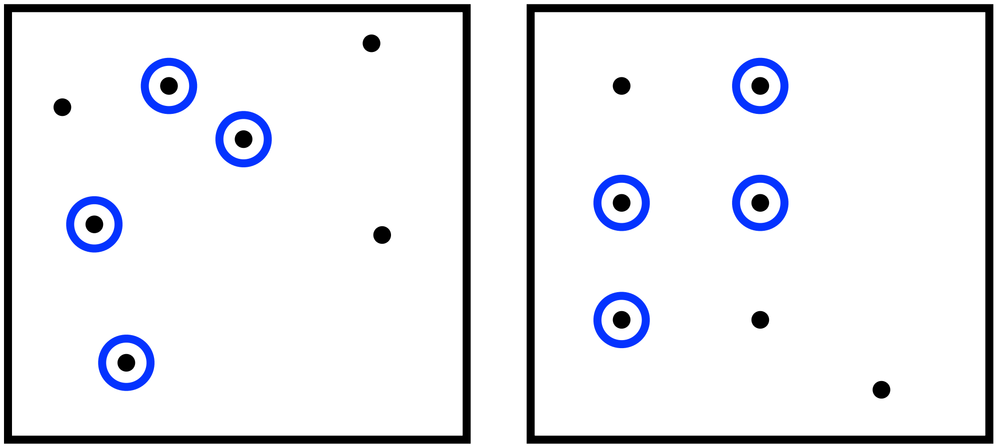
3) Count the number of features that are smaller than a distance threshold.



RANSAC with Features

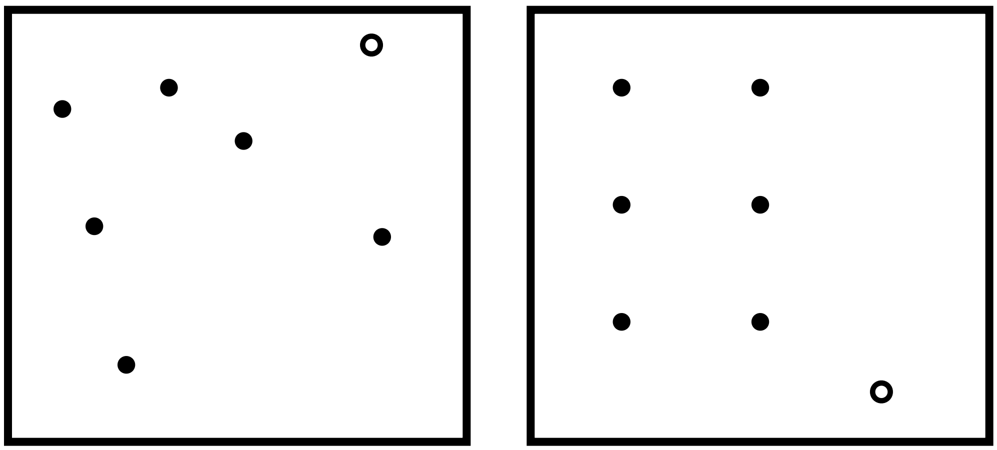
4) Go back to step 1.

5) Iterate many times. The homography with the most number of matches is your estimate!



RANSAC with Features

6) Those features that are consistent with your best homography are the inliers. Those that aren't are the outliers.



Summary

We've discussed how to find features.

We've discussed how to track those features into a second image.

And we've discussed how we can find features that were tracked well, and those that were wrong.

Now we can find and track features automatically!