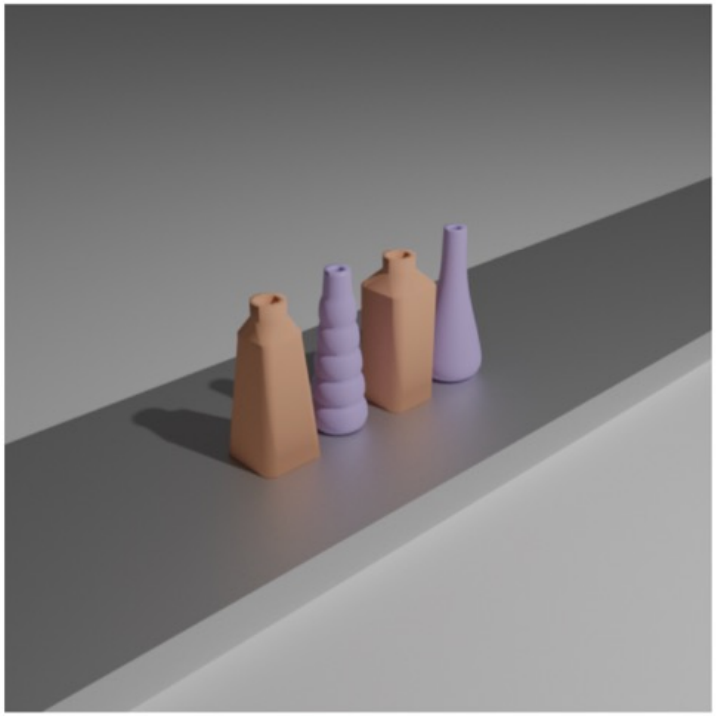
Feature & Corner Detection

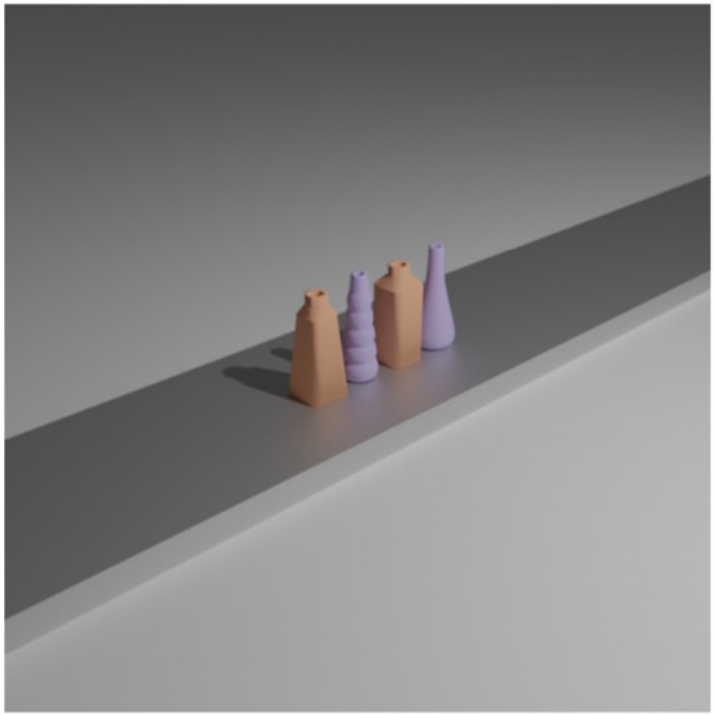
CS 482, Prof. Stein

Lecture 05

Motivation: Similar to last time… but we know more now.

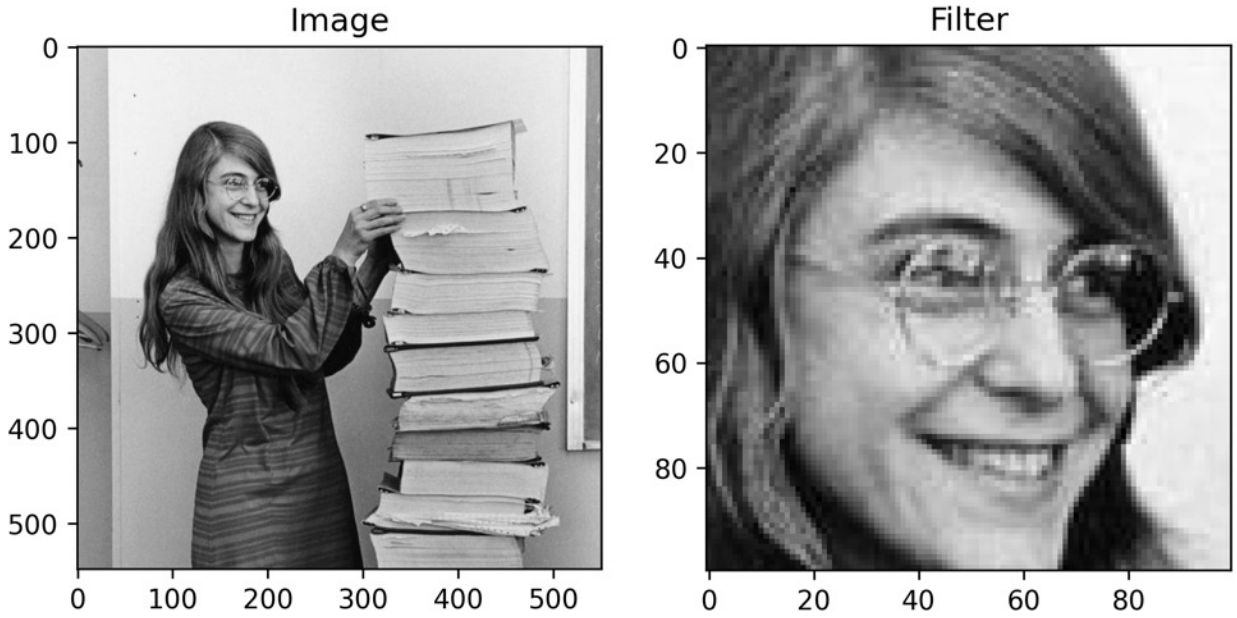
What if we want to find objects common to two images:





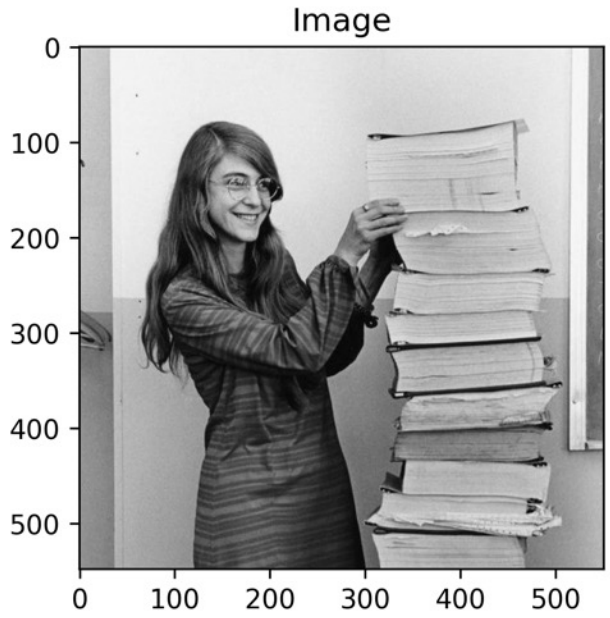
Motivation: What about a *match filter*?

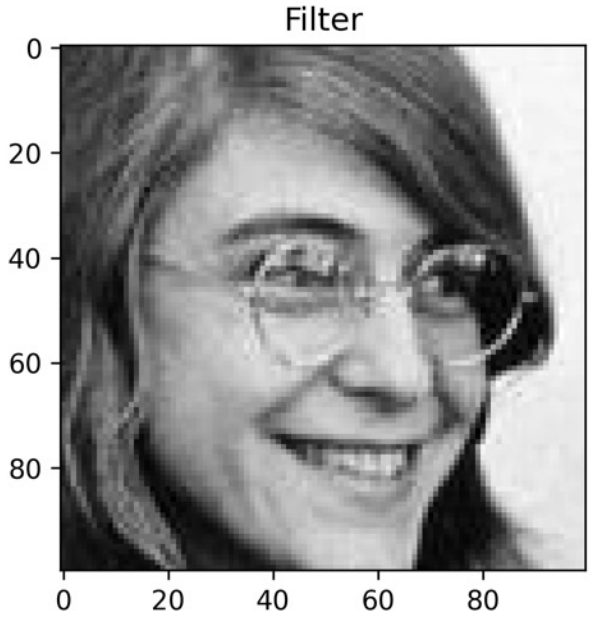
Let's compute the correlation with a part of the image itself:

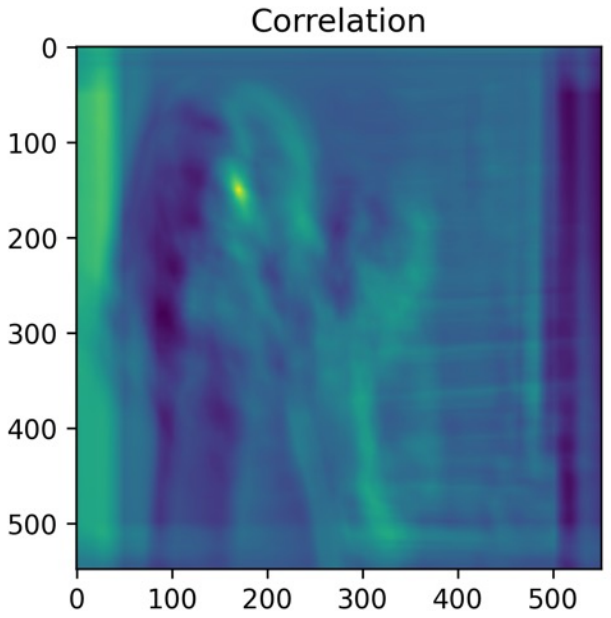


Motivation: What about a *match filter*?

Let's compute the correlation with a part of the image itself:



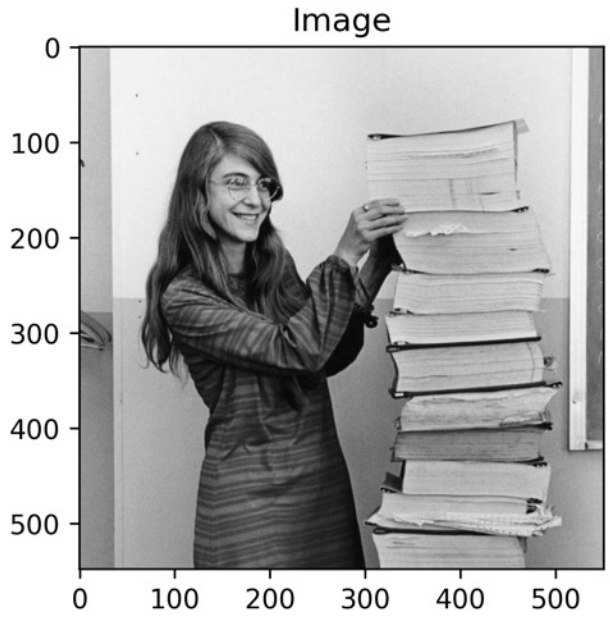


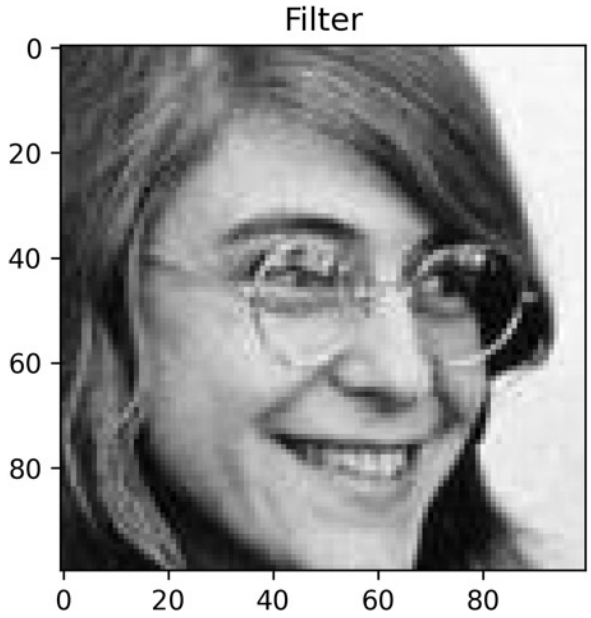


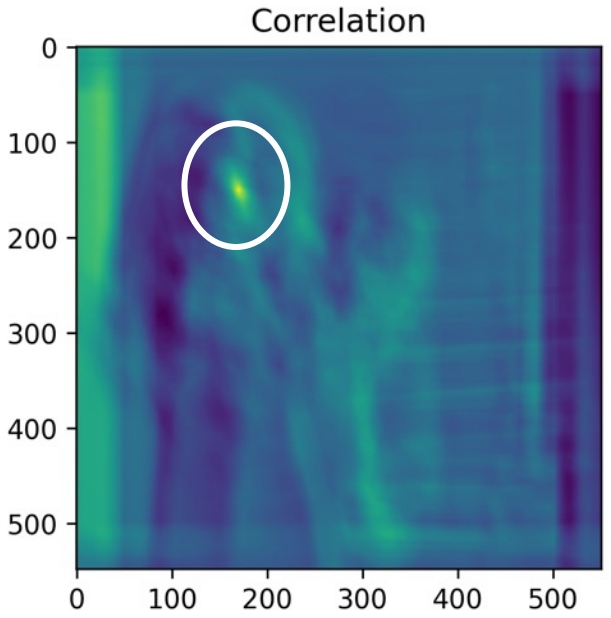
Note: I have *mean shifted* the filter

Motivation: What about a *match filter*?

Let's compute the correlation with a part of the image itself:





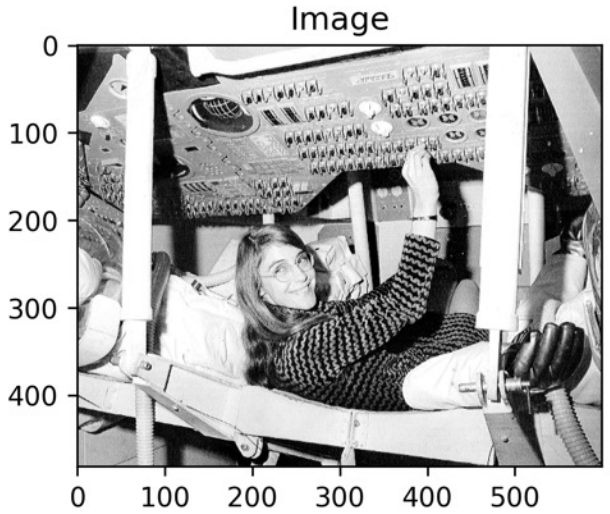


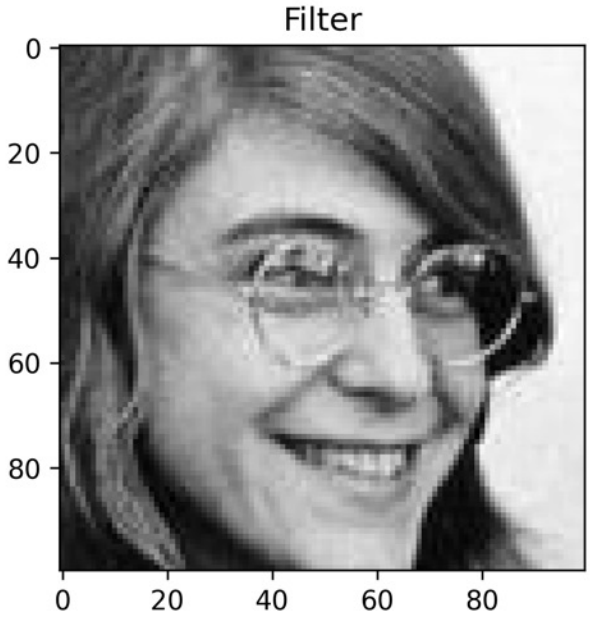
The bright spot corresponds to the filter. Are we done?

Note: I have *mean shifted* the filter

Motivation: What about a *match filter*?

What if we want to find M. Hamilton in a new image?

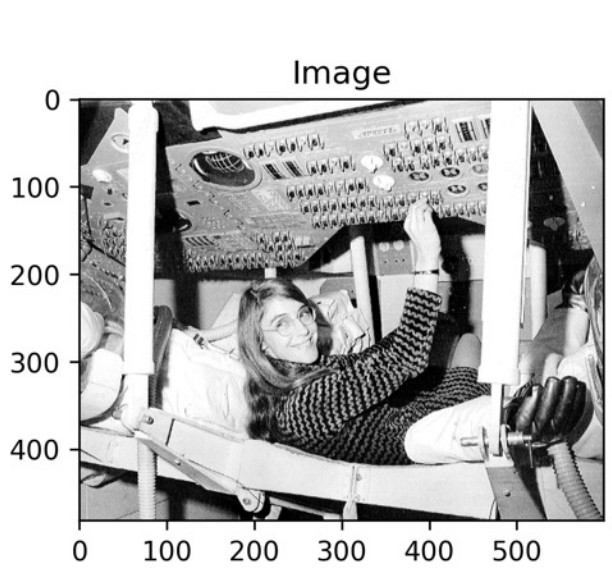


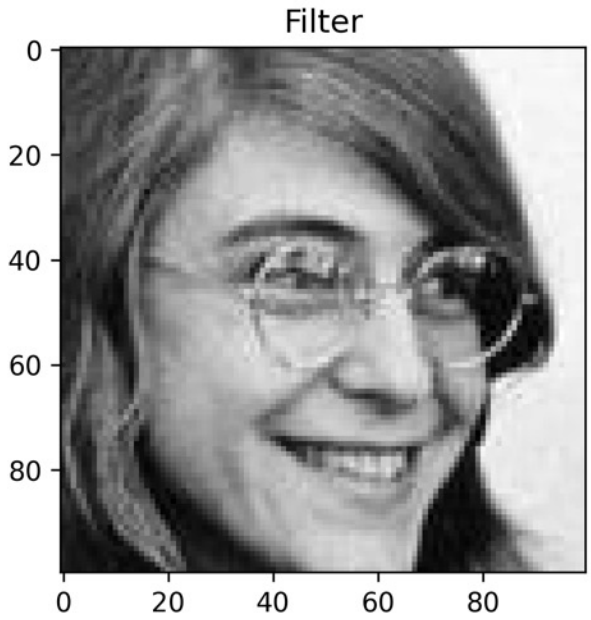


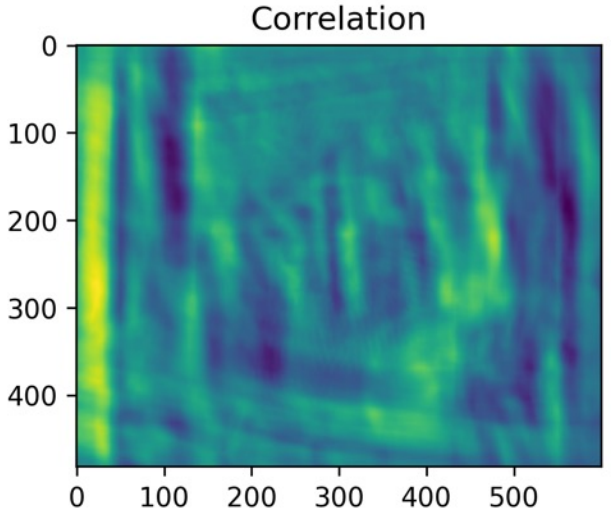
Note: I have *mean shifted* the filter

Motivation: What about a *match filter*?

What if we want to find M. Hamilton in a new image?





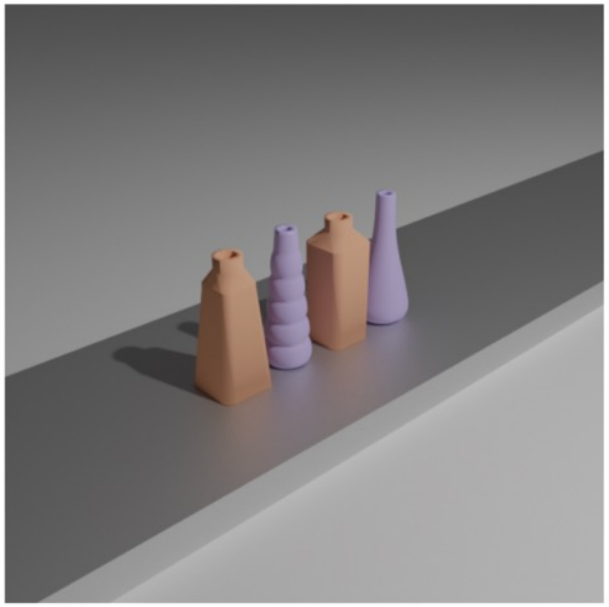


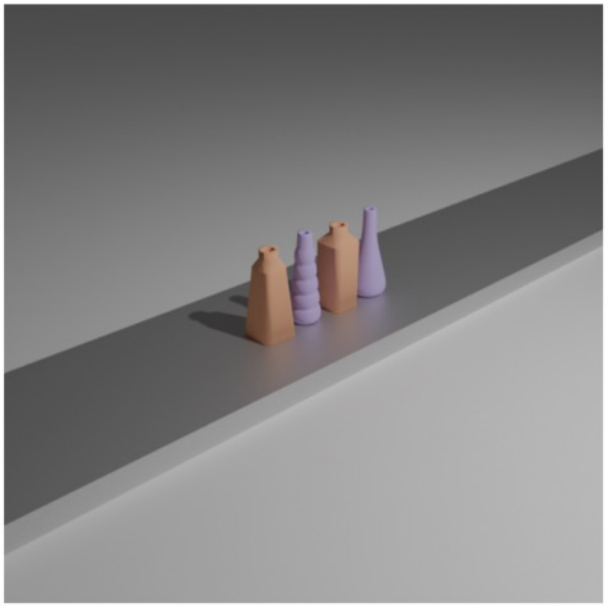
This filter is *not general* enough.

Note: I have *mean shifted* the filter

Motivation: simply resizing the images is insufficient for matching

If we never orbit the objects, filtering may work





However, in general, we move around the objects and rotate the camera, and filtering will likely not succeed.

Motivation: simply resizing the images is insufficient for matching

I want to be able to recognize familiar “things” between images for a ton of different applications:

- Object detection
- Tracking (knowing where I am compared to recent observations)
- Localization (knowing where I am compared to any observations)
- Object Recognition and Detection
- Image Classification

And many, many more… (some of which we’ll talk about!)

Motivation: Computing Generic Features

Today we'll discuss how to compute more general features, that we will later use to find objects or track geometry across images.

Reading & Slide Credits

Readings:

- Szeliski 4.1

Slide Credits (from which many of these slides are either directly taken or adapted)

- CMU Computer Vision Course (Yannis Gkioulekas)
- Cornell Tech Computer Vision Course (Noah Snavely)
- Toronto CV Course

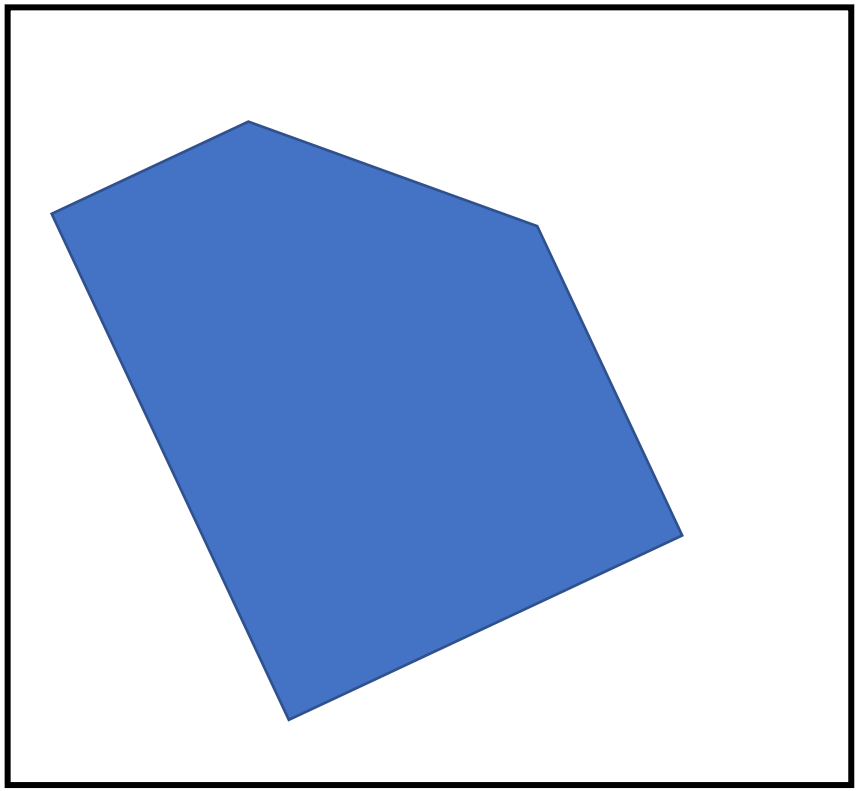
Local Features
and their applications

So what makes for a good feature?

What features will allow me to figure out where I am or match buildings in this image to other images of those buildings?

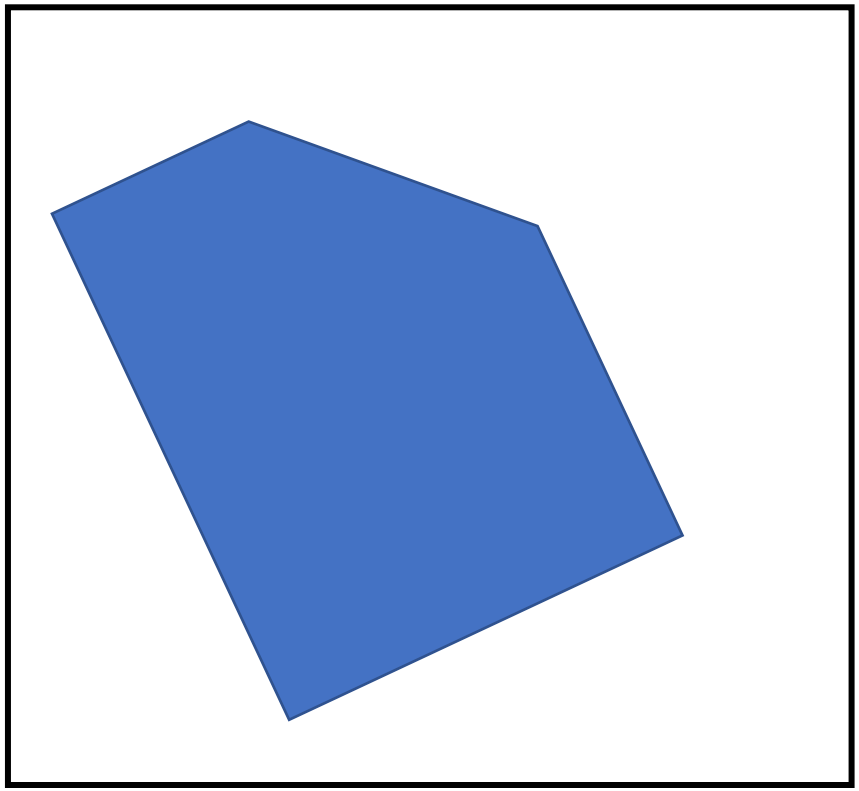
A Simple Example: How might you "track" this object?

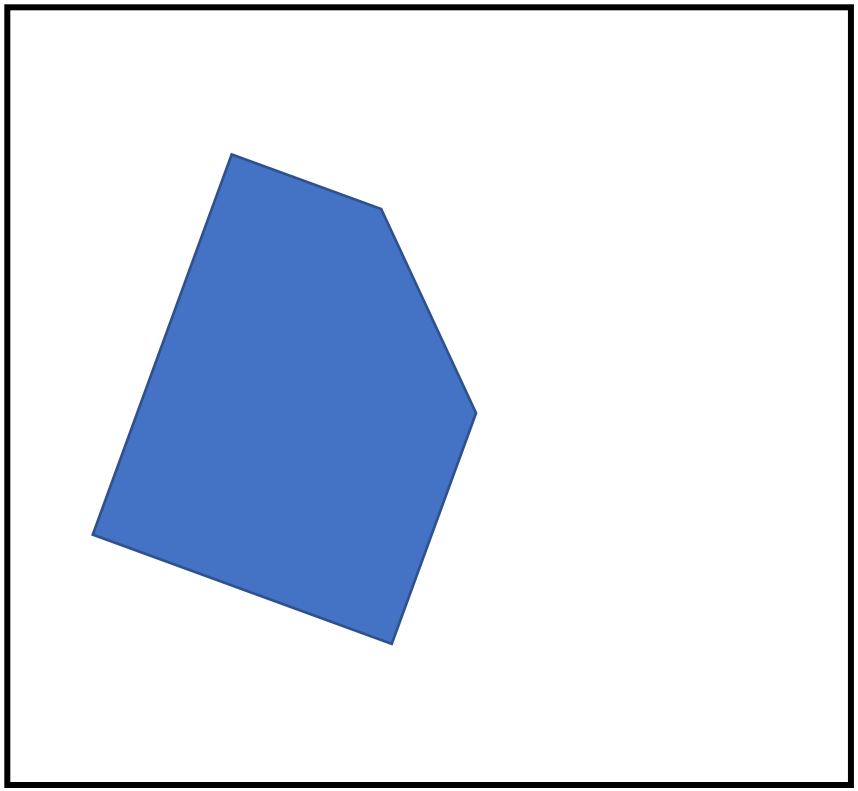
Pick some points on this image and look for them in another.



A Simple Example: How might you "track" this object?

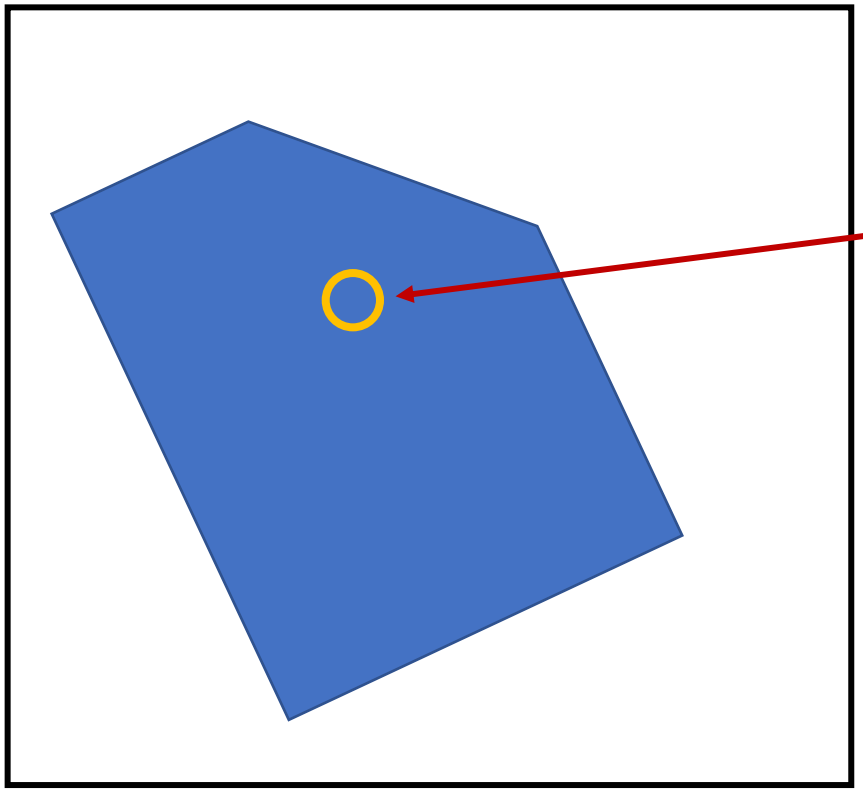
Pick some points on this image and look for them in another.





A Simple Example: How might you "track" this object?

Pick some points on this image and look for them in another.

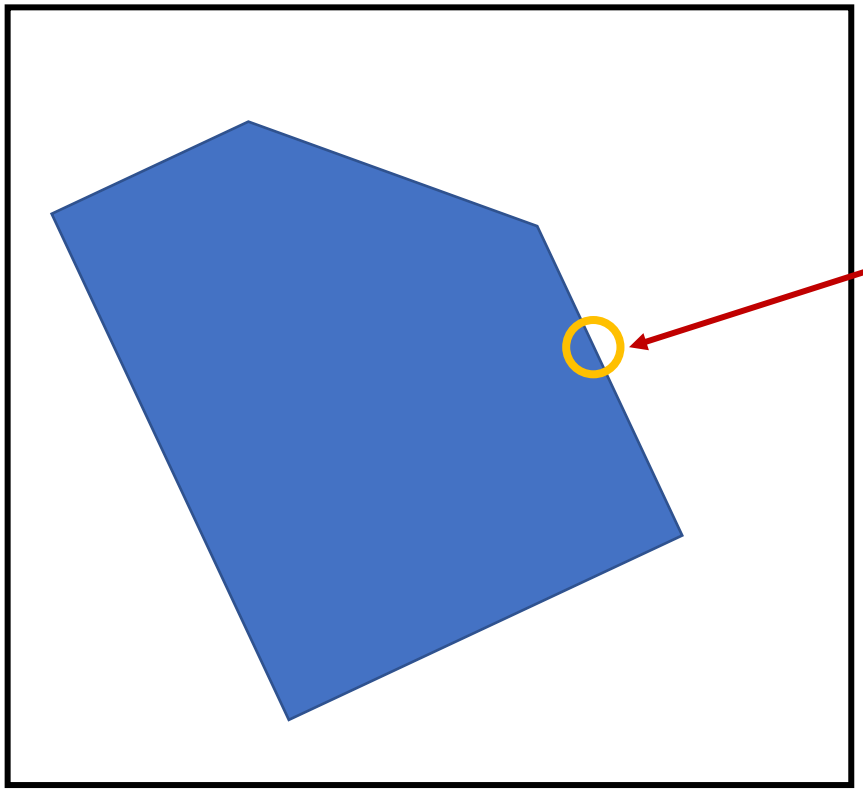


Is this a good feature?

No! This region is *flat*; not at all distinct.

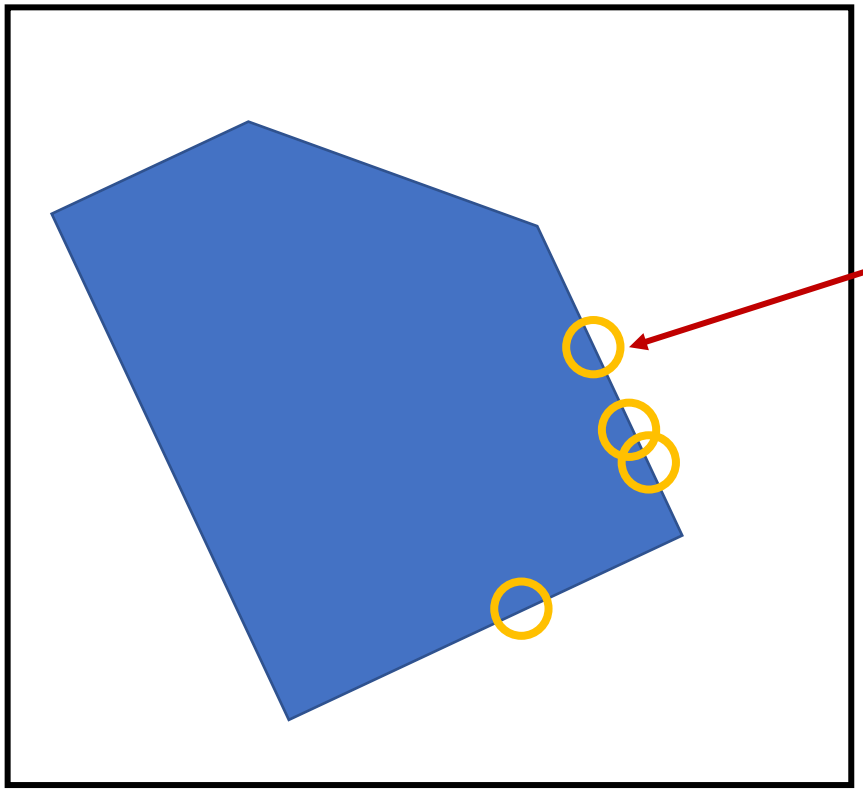
A Simple Example: How might you "track" this object?

Pick some points on this image and look for them in another.



A Simple Example: How might you "track" this object?

Pick some points on this image and look for them in another.

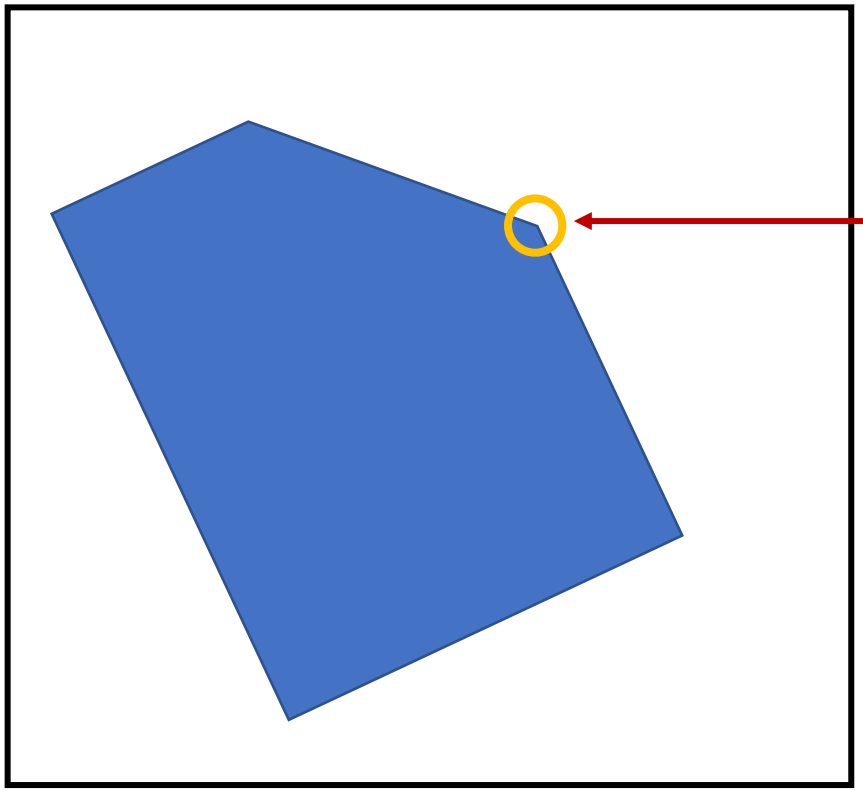


Is this edge a good feature?

Better, but still not great. Hard to distinguish between these. We can *slide along* the edge and not know the difference.

A Simple Example: How might you "track" this object?

Pick some points on this image and look for them in another.

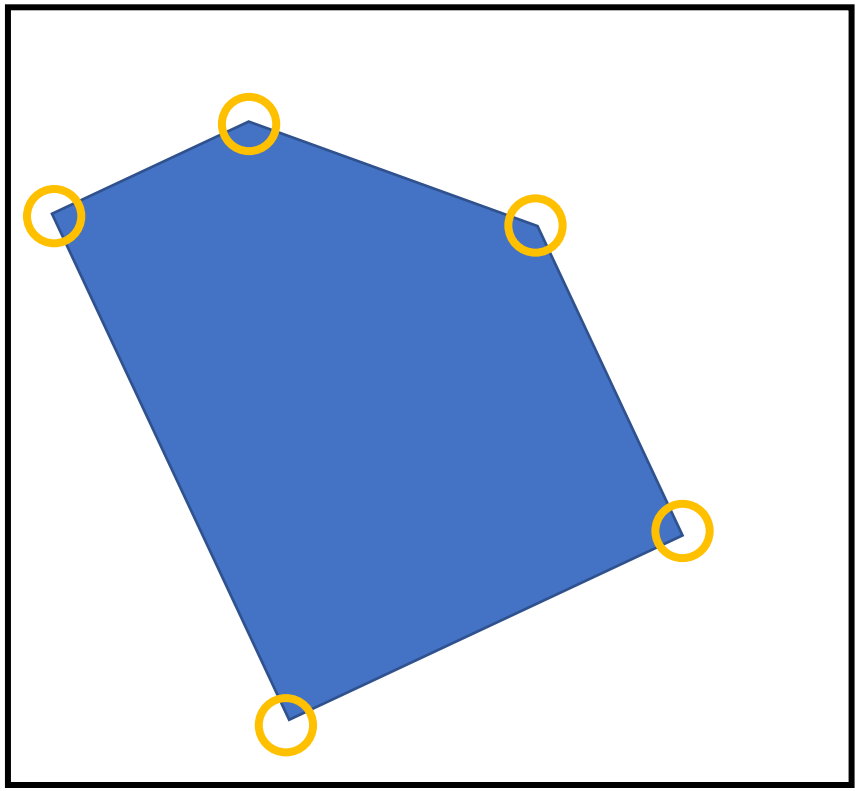


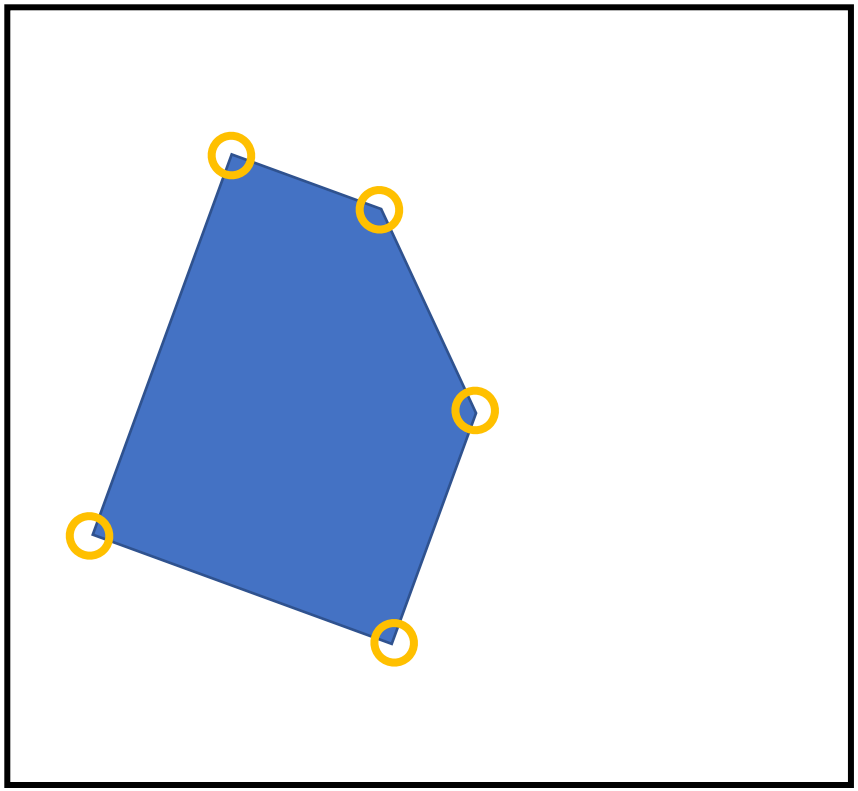
Is this corner a good feature?

Much better. Corners are much easier to localize (and don't "slide" like edges do).

A Simple Example: How might you "track" this object?

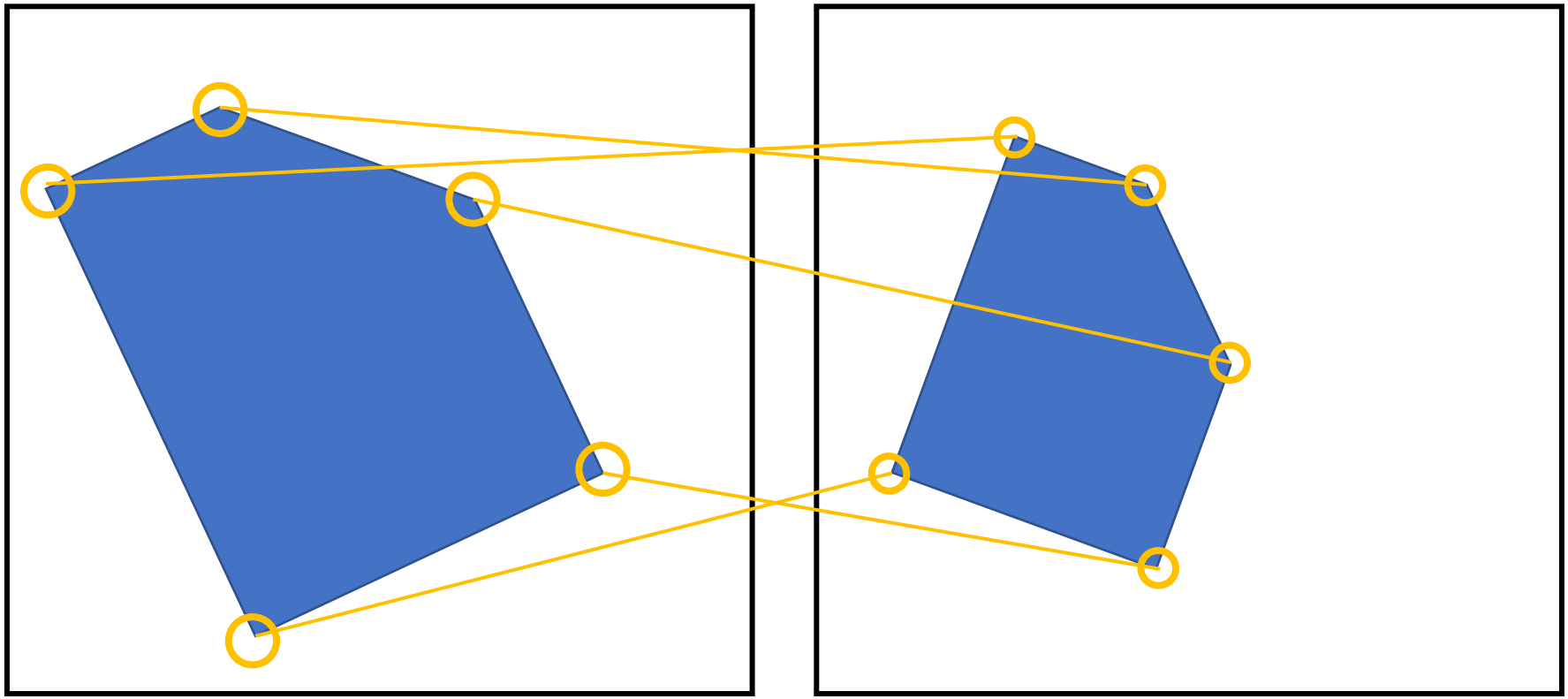
Pick some points on this image and look for them in another.





A Simple Example: How might you "track" this object?

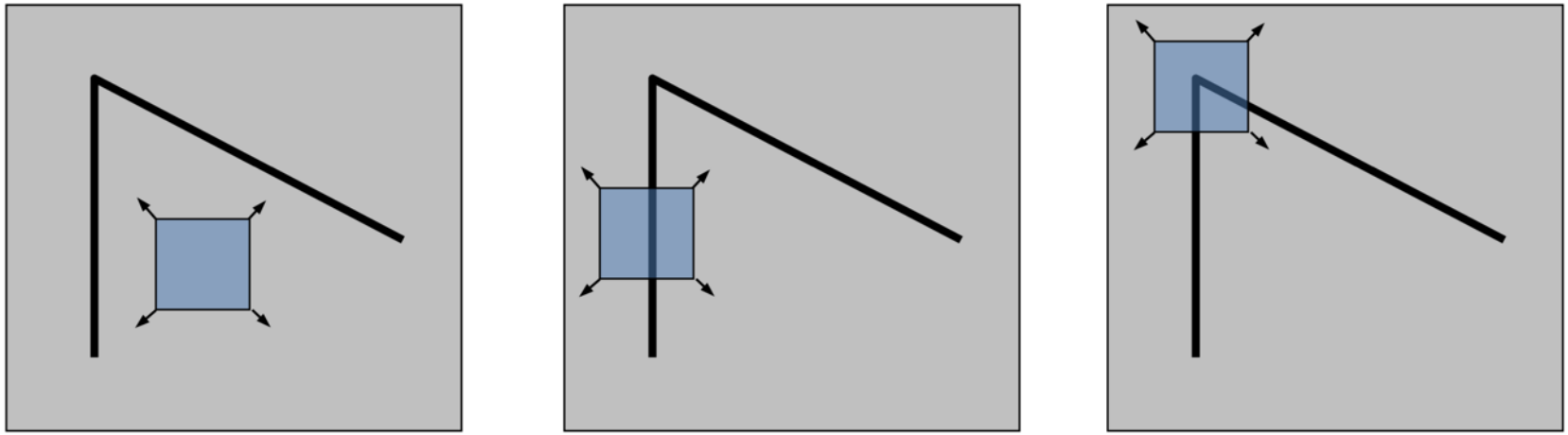
Pick some points on this image and look for them in another.



Features should be as *unique* as is reasonably achievable.

This will help us make less ambiguous matches between images.

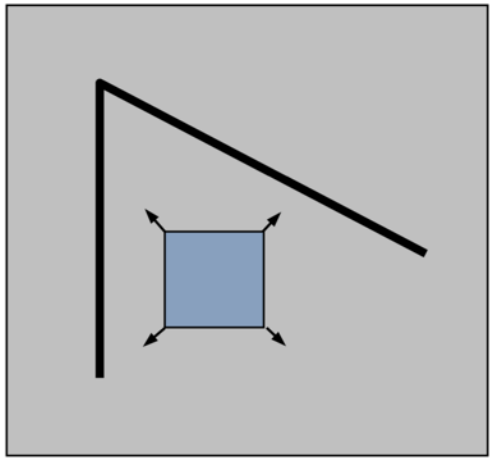
Let's consider looking at only a small window of pixels:

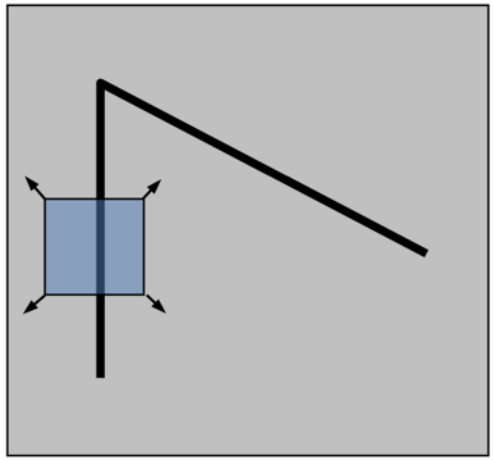


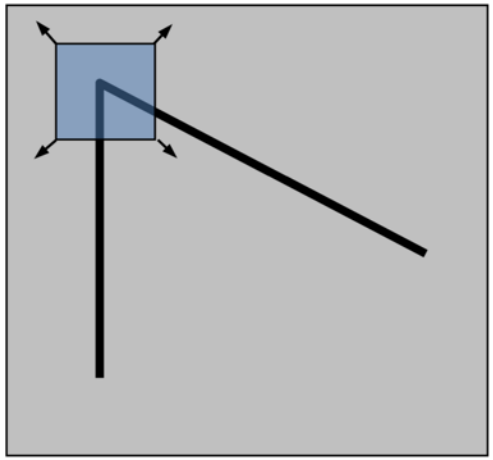
Corners are easier to localize, which will make them easier to match across images.

How can we detect where a corner exists?

Can we just detect regions in which the image derivative is high in both X and Y?



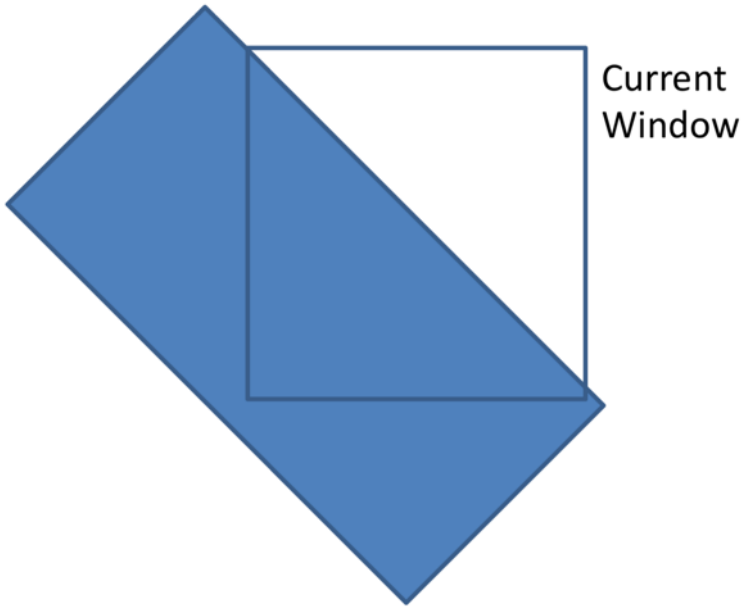




How can we detect where a corner exists?

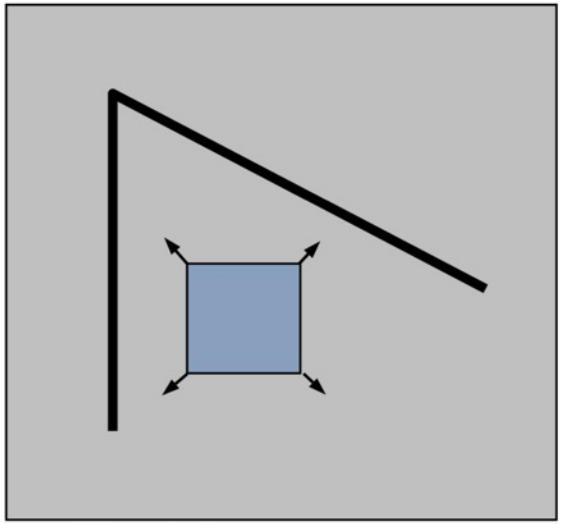
Can we just detect regions in which the image derivative is high in both X and Y?

No! This would detect diagonal edges as well as corners:

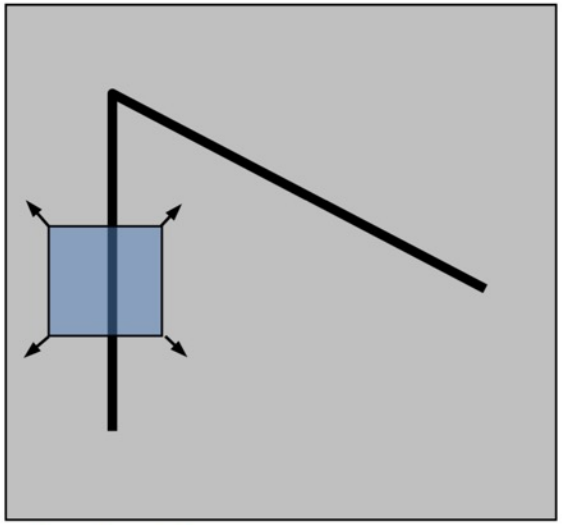


How can we detect where a corner exists?

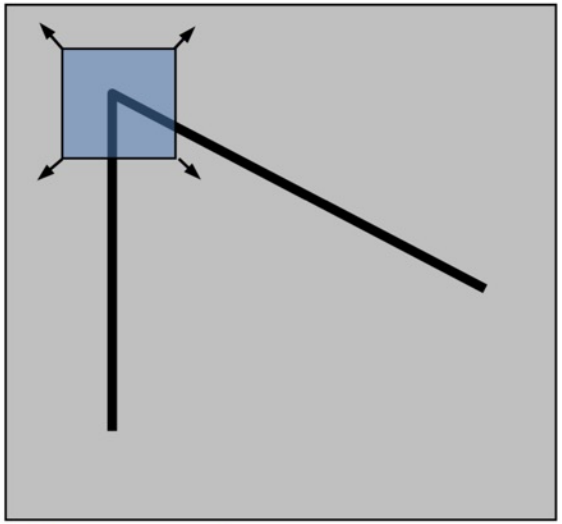
Let's consider the pixels in a small window: how much does this window change when you move it around the image.



"flat" region:
no change in all directions



"edge":
no change along the edge direction



"corner":
significant change in all directions

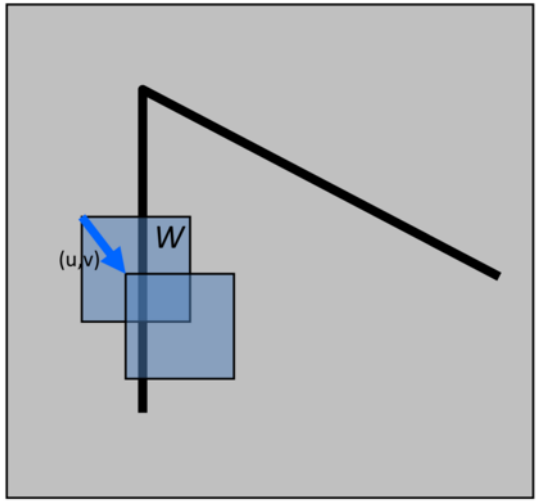
Harris Corner Detection: Finding corners with a sliding local window

- Let's move a window W around an image I.
- We can ask ourselves *how do the pixels change within the window?* and compare the difference between before the move and after the move with a Sum of Squares Error:

$$E(u,v) = \sum_{(x,y)\in W} [I(x+u, y+v) - I(x,y)]^2$$

At corners, this error is expected to be particularly high.

Can we use this to detect corners?



It can be slow to compute this error, but we can use a Taylor Series expansion

Taylor Series expansion of *I*:

$$I(x+u, y+v) = I(x,y) + \frac{\partial I}{\partial x}u + \frac{\partial I}{\partial y}v + \text{higher order terms}$$

If the motion (u,v) is small, then first order approximation is good

$$\begin{aligned} I(x+u, y+v) &\approx I(x,y) + \frac{\partial I}{\partial x}u + \frac{\partial I}{\partial y}v \\ &\approx I(x,y) + \begin{bmatrix} I_x & I_y \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} \end{aligned}$$

$$\text{shorthand: } I_x = \frac{\partial I}{\partial x}$$

Harris Corner Detection: Derivation

With the Taylor Series expansion, we can approximate this *sliding window error*

$$
\begin{aligned}
E(u,v) &= \sum_{(x,y)\in W} \left[I(x+u,y+v) - I(x,y)\right]^2 \\
&\approx \sum_{(x,y)\in W} \left[I(x,y) + I_x u + I_y v - I(x,y)\right]^2 \\
&\approx \sum_{(x,y)\in W} \left[I_x u + I_y v\right]^2
\end{aligned}
$$

Harris Corner Detection: Derivation

With the Taylor Series expansion, we can approximate this *sliding window error*

$$E(u,v) \approx \sum_{(x,y)\in W} [I_x u + I_y v]^2$$

$$\approx Au^2 + 2Buv + Cv^2$$

where

$$A = \sum_{(x,y)\in W} I_x^2 \qquad B = \sum_{(x,y)\in W} I_x I_y \qquad C = \sum_{(x,y)\in W} I_y^2$$

The surface E can be approximated by a quadratic form:

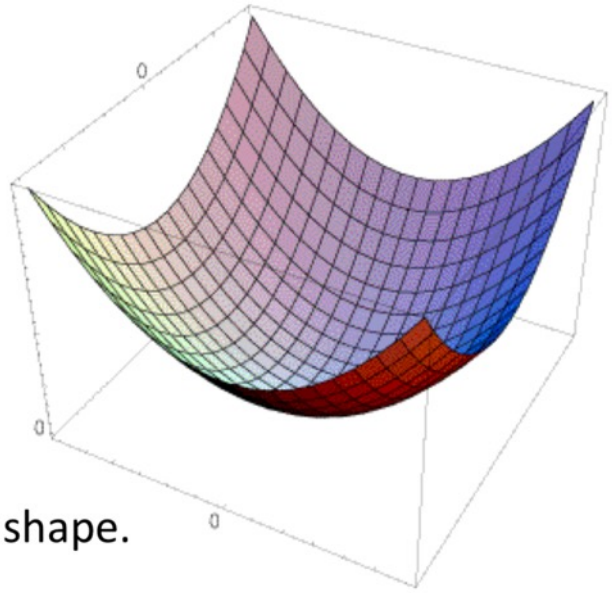
$$E(u,v) \approx Au^2 + 2Buv + Cv^2$$

$$\approx \begin{bmatrix} u & v \end{bmatrix} \underbrace{\begin{bmatrix} A & B \\ B & C \end{bmatrix}}_{H} \begin{bmatrix} u \\ v \end{bmatrix}$$

$$A = \sum_{(x,y)\in W} I_x^2$$

$$B = \sum_{(x,y)\in W} I_x I_y$$

$$C = \sum_{(x,y)\in W} I_y^2$$



Let's try to understand its shape.

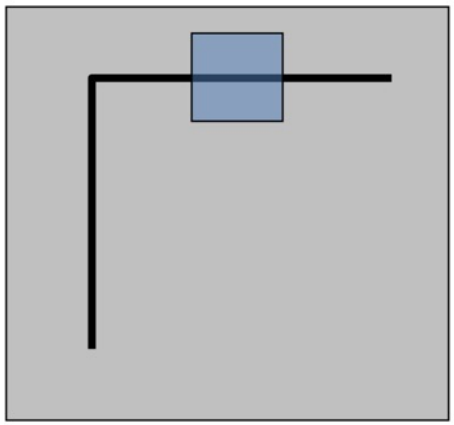
For a horizontal edge, the error surface is a quadratic ridge

$$E(u,v) \approx \begin{bmatrix} u & v \end{bmatrix} \underbrace{\begin{bmatrix} A & B \\ B & C \end{bmatrix}}_{H} \begin{bmatrix} u \\ v \end{bmatrix}$$

$$A = \sum_{(x,y)\in W} I_x^2$$

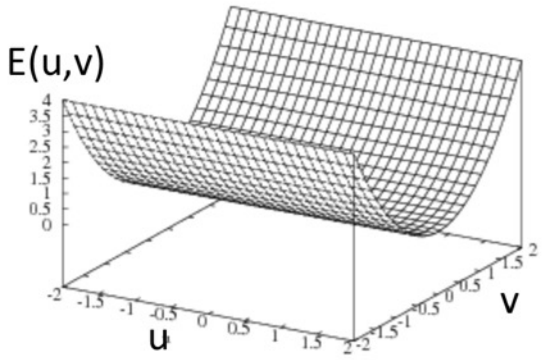
$$B = \sum_{(x,y)\in W} I_x I_y$$

$$C = \sum_{(x,y)\in W} I_y^2$$



Horizontal edge: $I_x = 0$

$$H = \begin{bmatrix} 0 & 0 \\ 0 & C \end{bmatrix}$$



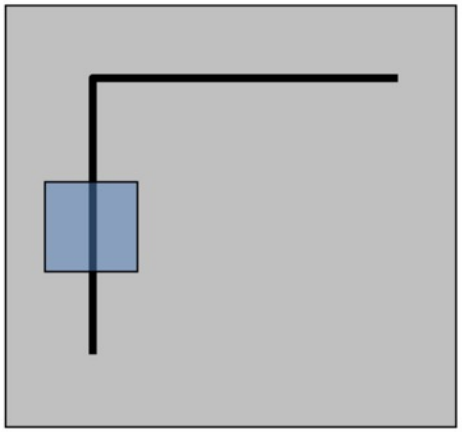
The same is true for a vertical edge, but the ridge has a different orientation

$$E(u,v) \approx \begin{bmatrix} u & v \end{bmatrix} \underbrace{\begin{bmatrix} A & B \\ B & C \end{bmatrix}}_{H} \begin{bmatrix} u \\ v \end{bmatrix}$$

$$A = \sum_{(x,y)\in W} I_x^2$$

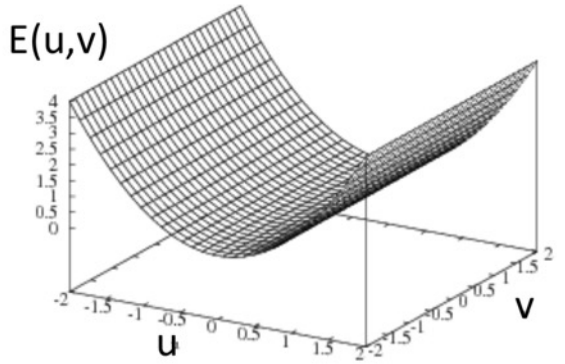
$$B = \sum_{(x,y)\in W} I_x I_y$$

$$C = \sum_{(x,y)\in W} I_y^2$$



Vertical edge: $I_y = 0$

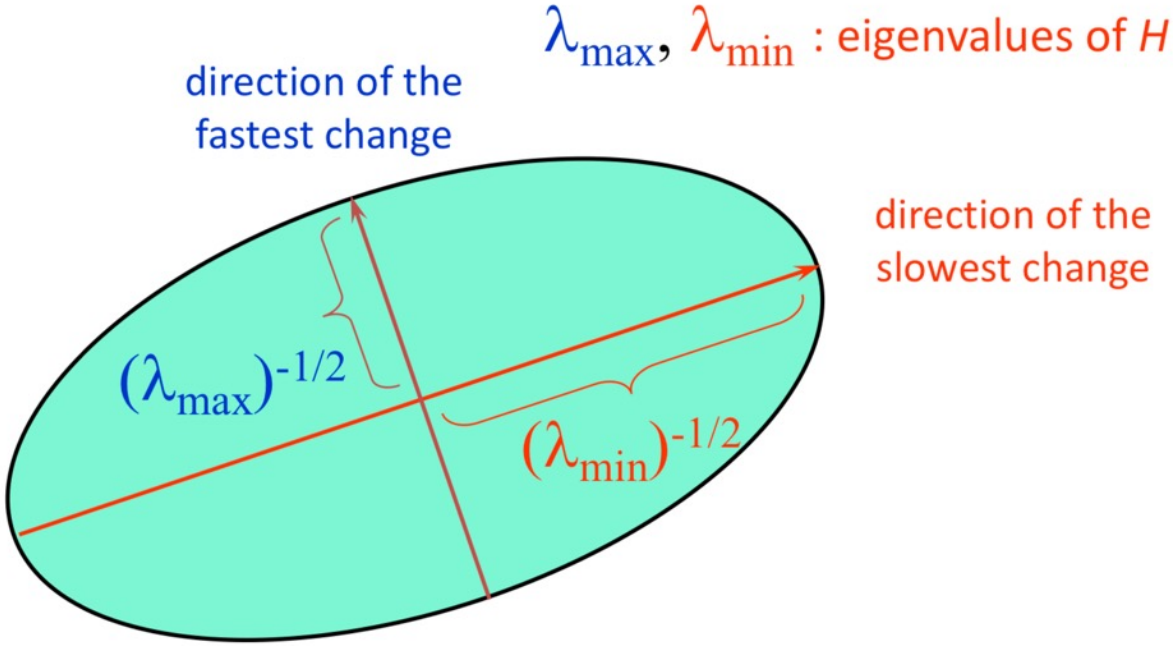
$$H = \begin{bmatrix} A & 0 \\ 0 & 0 \end{bmatrix}$$



In general, a horizontal slice of the error surface is an ellipse

Ellipse equation:

$$[u \;\; v] \;\; H \begin{bmatrix} u \\ v \end{bmatrix} = \text{const}$$



The eigenvectors indicate the directions of maximal and minimal change, and their eigenvectors indicate how how much change.

Aside: a quick eigenvalue review

The **eigenvectors** of a matrix **A** are the vectors **x** that satisfy:

$$Ax = \lambda x$$

The scalar λ is the **eigenvalue** corresponding to **x**

- The eigenvalues are found by solving:

$$det(A - \lambda I) = 0$$

- In our case, ***A*** = ***H*** is a 2x2 matrix, so we have

$$det \begin{bmatrix} h_{11} - \lambda & h_{12} \\ h_{21} & h_{22} - \lambda \end{bmatrix} = 0$$

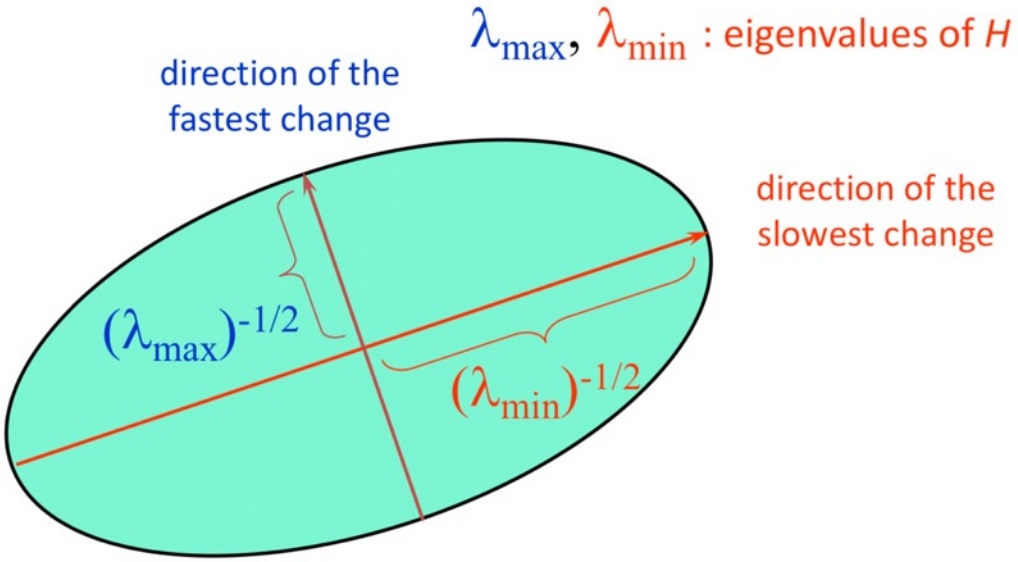
- The solution:

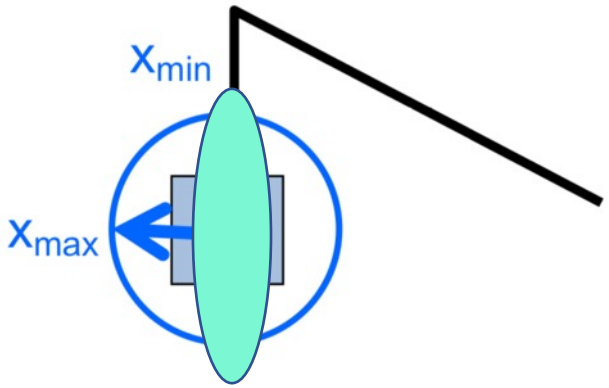
$$\lambda_{\pm} = \frac{1}{2} \left[(h_{11} + h_{22}) \pm \sqrt{4h_{12}h_{21} + (h_{11} - h_{22})^2} \right]$$

Once you know λ, you find **x** by solving

$$\begin{bmatrix} h_{11} - \lambda & h_{12} \\ h_{21} & h_{22} - \lambda \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = 0$$

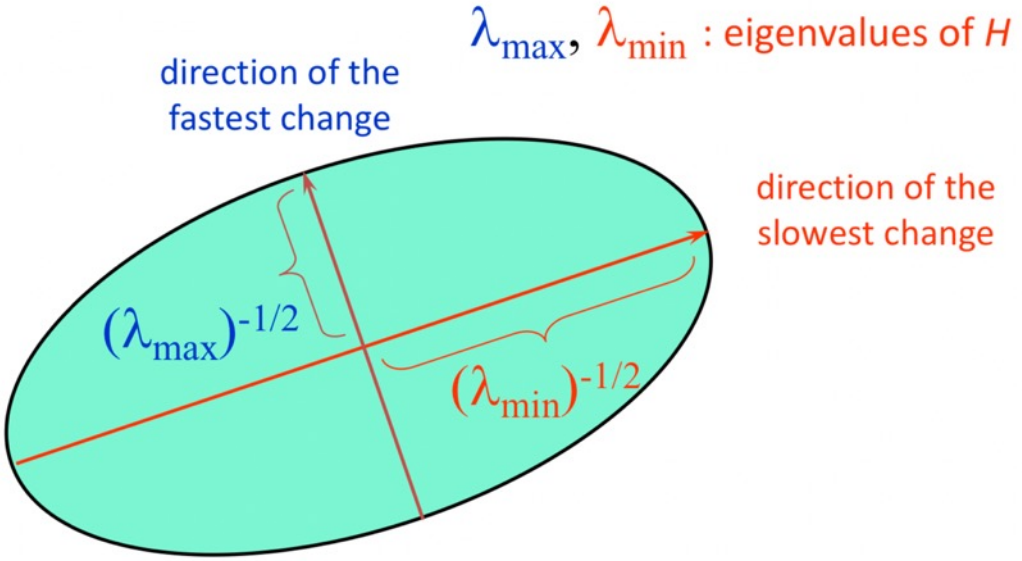
So what do the eigenvalues and eigenvectors look like in practice?

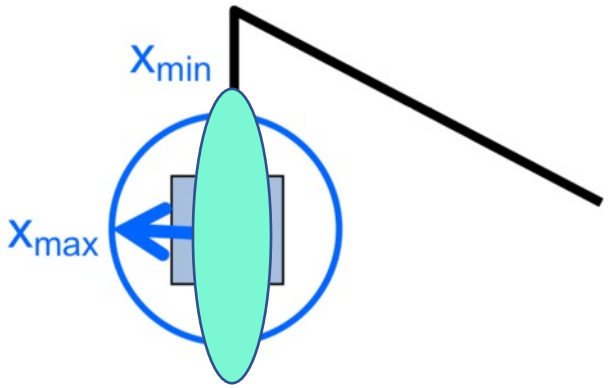




Corners correspond to an ellipse for which both eigenvalues are large.

So what do the eigenvalues and eigenvectors look like in practice?



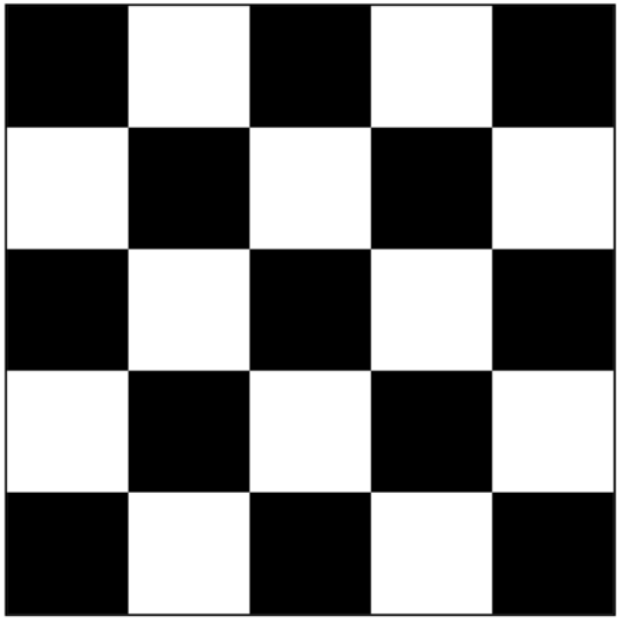


Corners correspond to an ellipse for which both eigenvalues are large.

So now what? How do we actually compute the corners?

Scoring the features by the eigenvalues

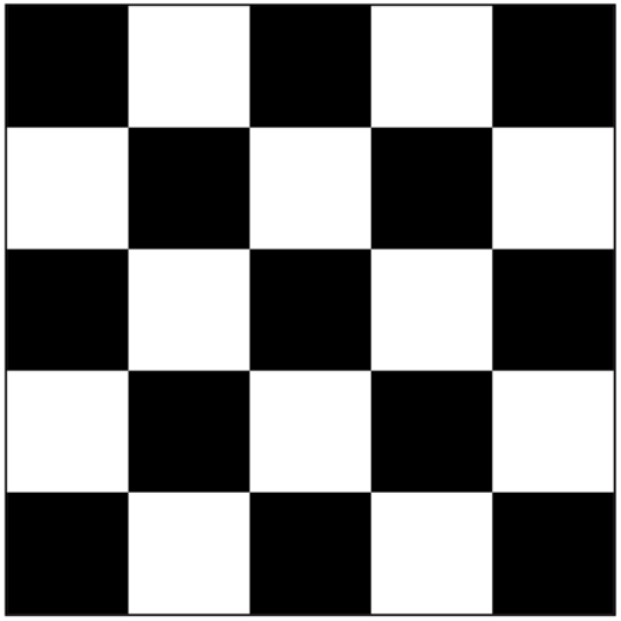
Let's look at a simple example:



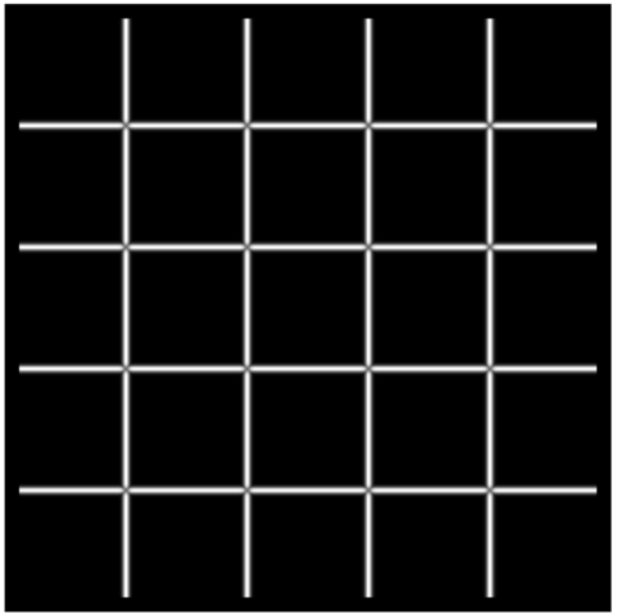
I

Scoring the features by the eigenvalues

Let's look at a simple example:



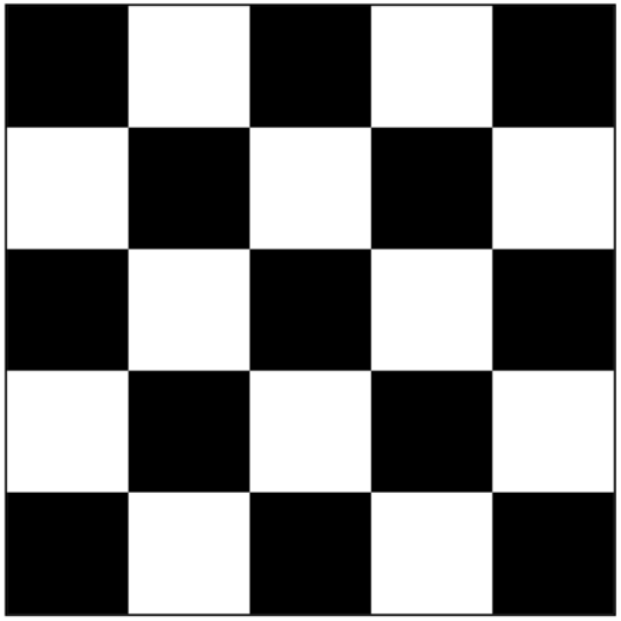
I



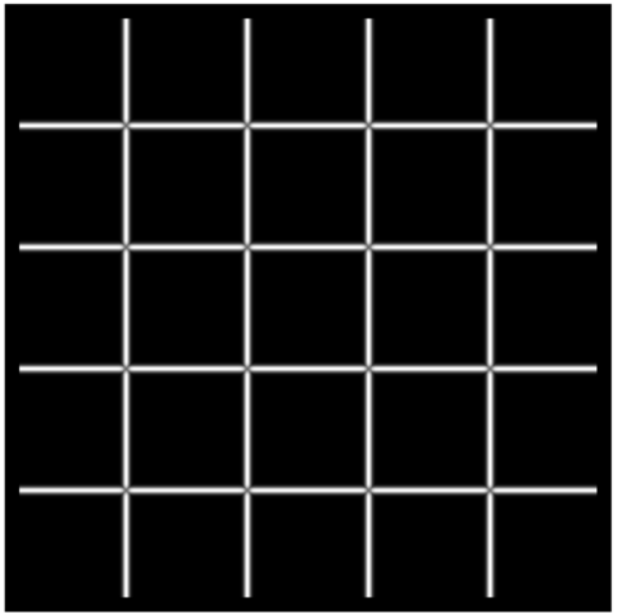
$\lambda_{\max}$

Scoring the features by the eigenvalues

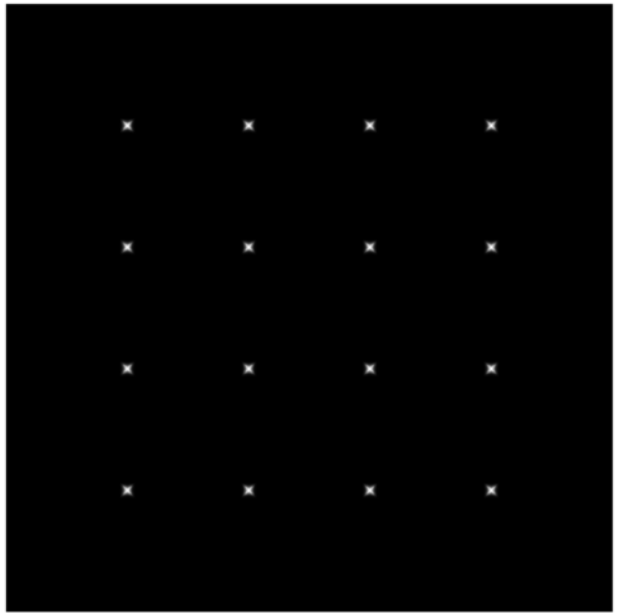
Let's look at a simple example:



I



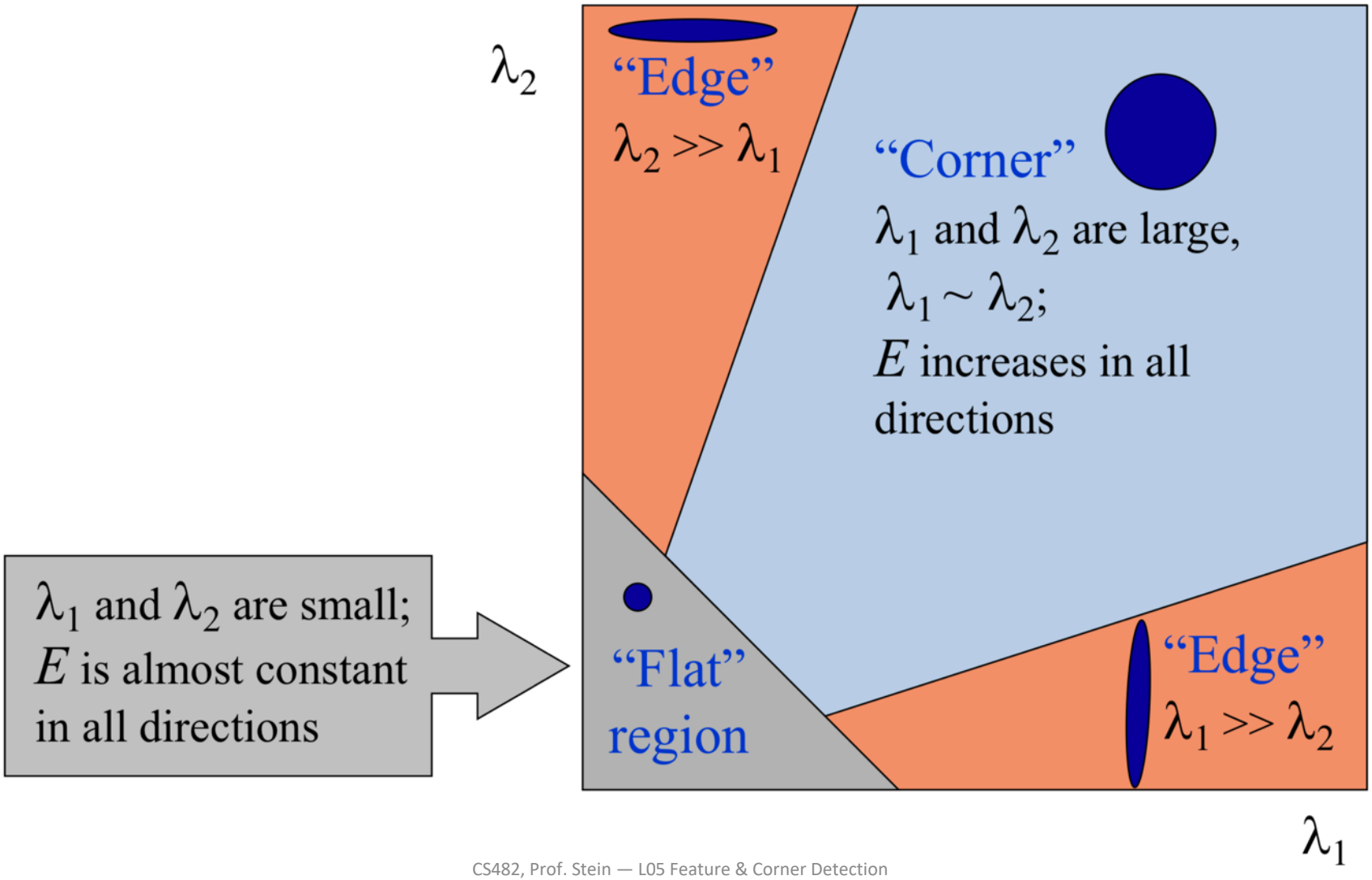
$\lambda_{\max}$



$\lambda_{\min}$

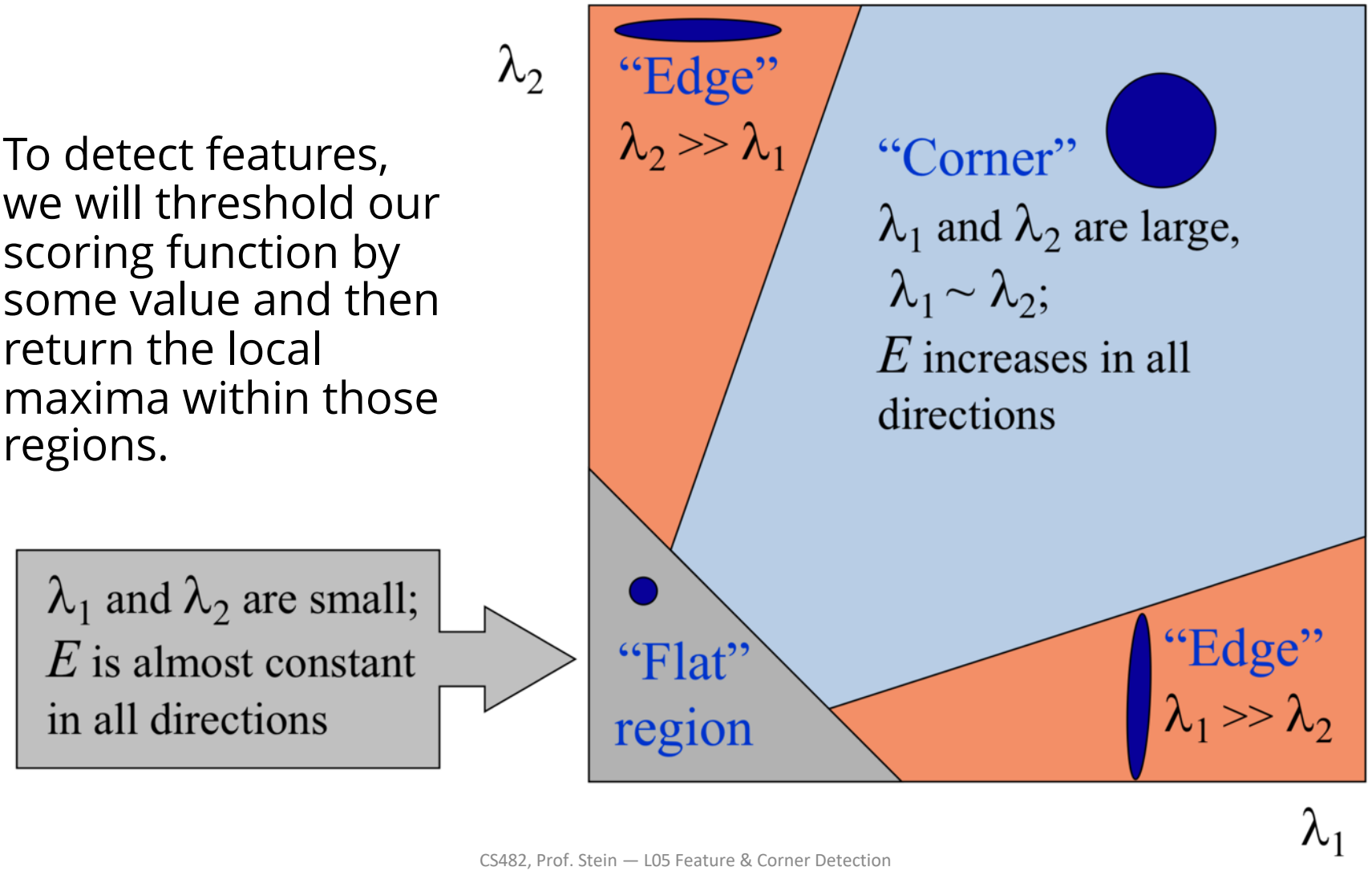
Corners exist where the minimal eigenvalue is high.

We can classify points in the image by their eigenvalues (a scoring function)



We can classify points in the image by their eigenvalues (a scoring function)

To detect features, we will threshold our scoring function by some value and then return the local maxima within those regions.



Some scoring functions are easier to compute than others

It is perhaps more common to use the determinant and the trace to define f:

$$f = \frac{\lambda_1 \lambda_2}{\lambda_1 + \lambda_2}$$

$$= \frac{determinant(H)}{trace(H)}$$

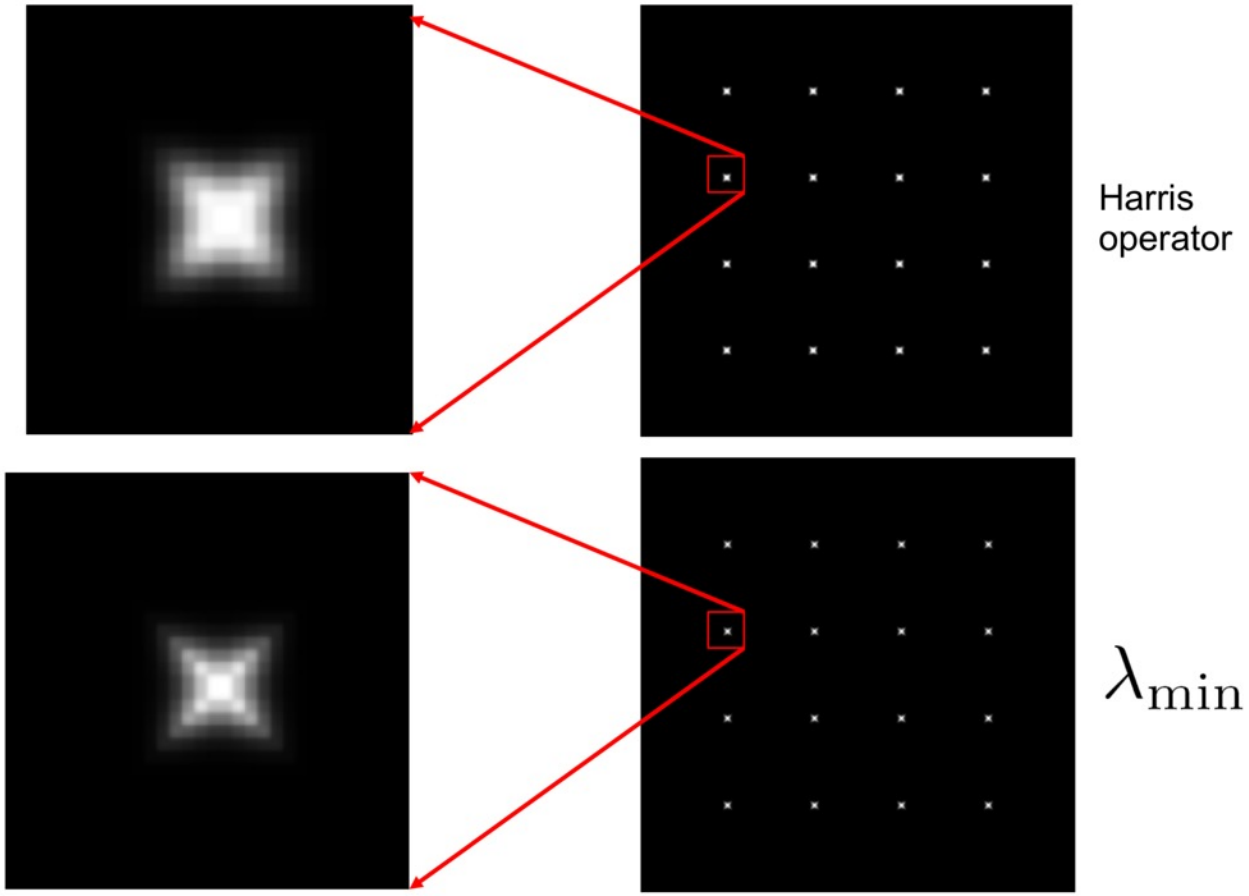
$$\det \begin{bmatrix} A & B \\ B & C \end{bmatrix} = AC - B^2$$

$$\mathrm{tr} \begin{bmatrix} A & B \\ B & C \end{bmatrix} = A + C$$

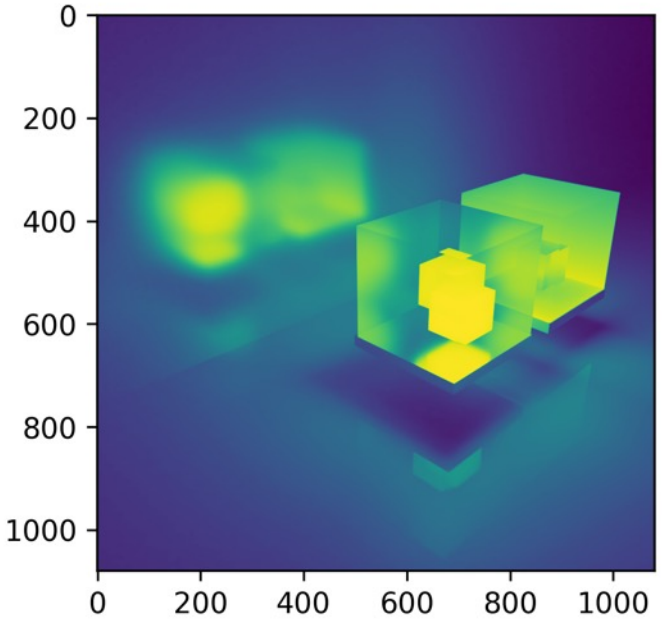
This is the *Harris Operator*

- It is faster to compute (doesn't require a square root)
- Very similar to the *mimimum-eigenvalue* scoring function in practice.

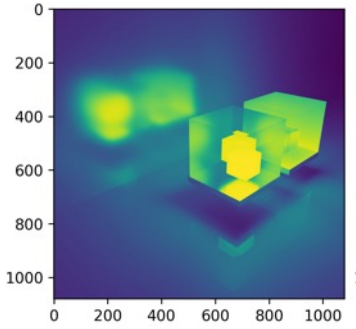
Some scoring functions are easier to compute than others



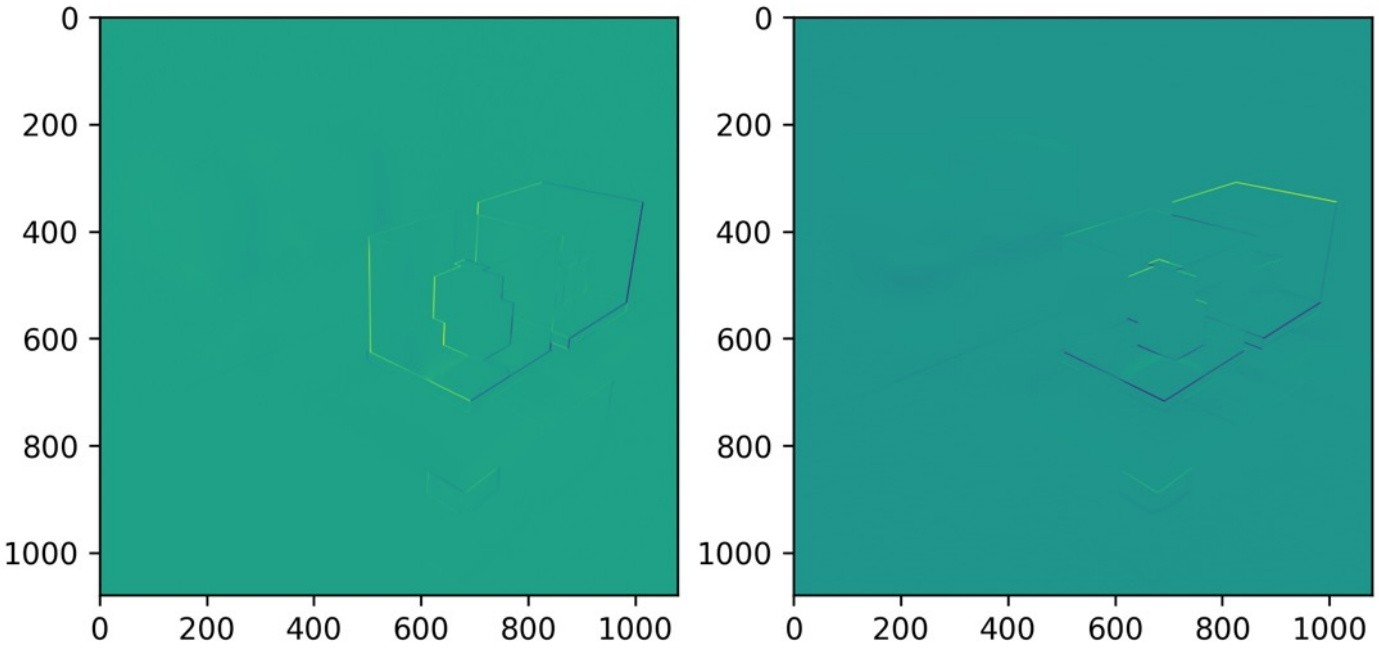
Overview of Harris Corner Detection



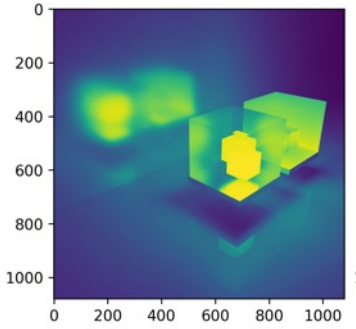
Overview of Harris Corner Detection



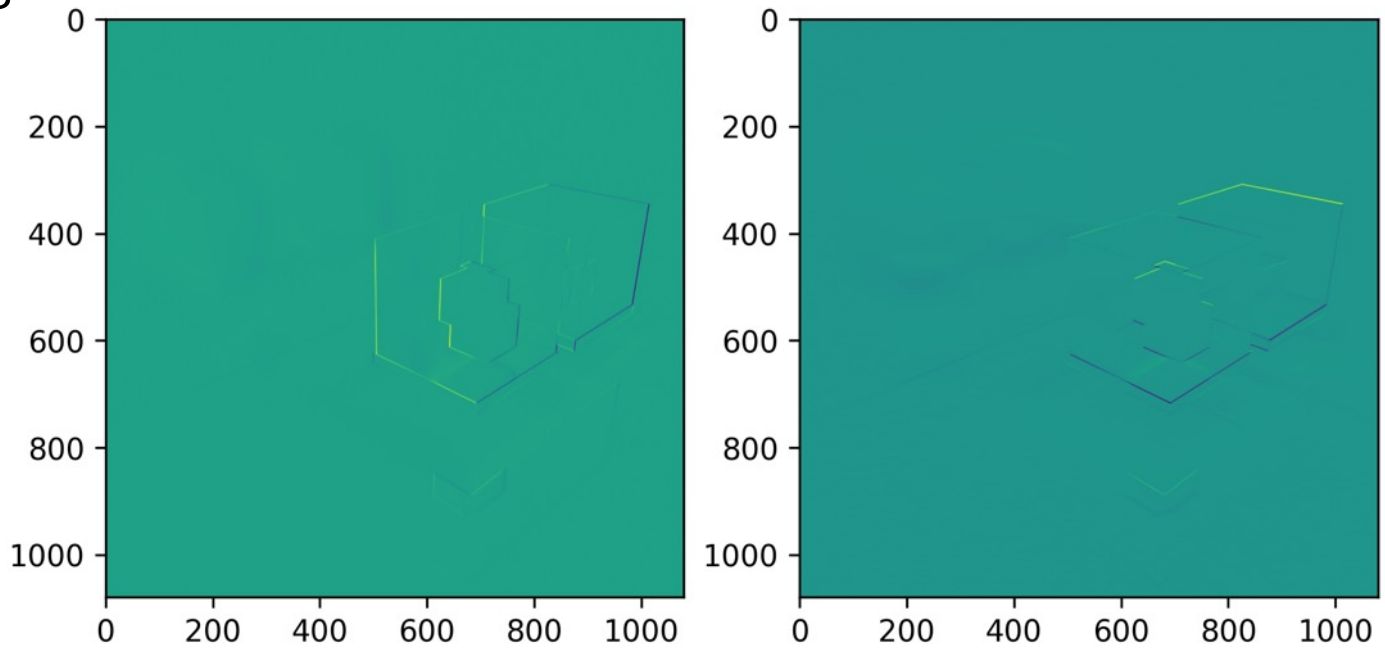
1. Compute the gradient at each point in the image



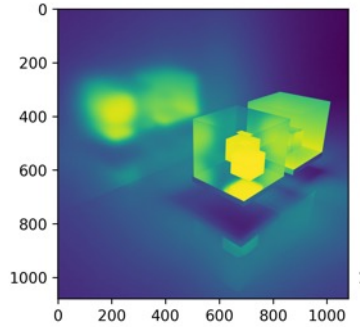
Overview of Harris Corner Detection



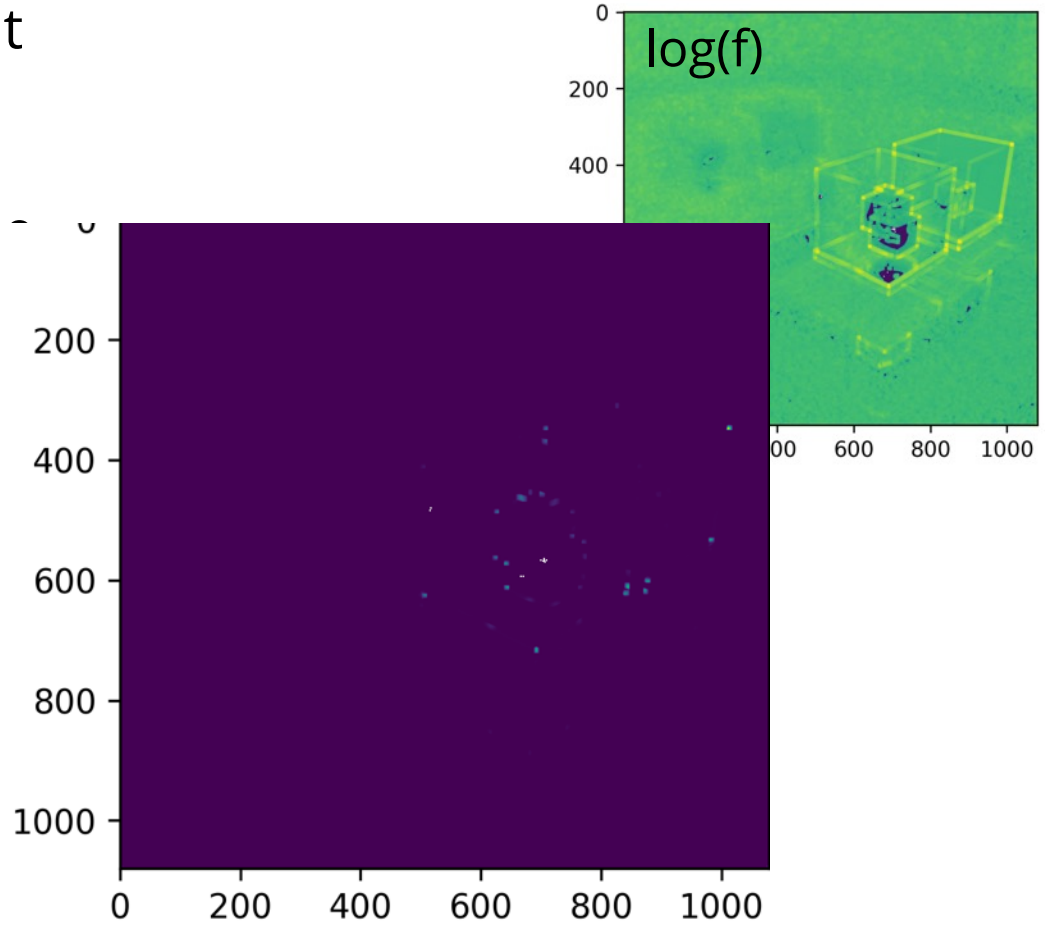
1. Compute the gradient at each point in the image
2. Compute the H matrix from the elements of the gradient



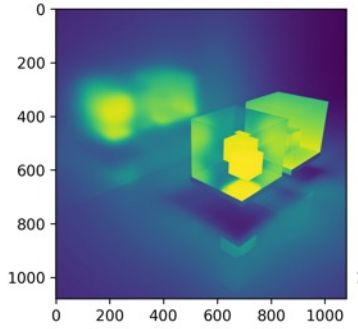
Overview of Harris Corner Detection



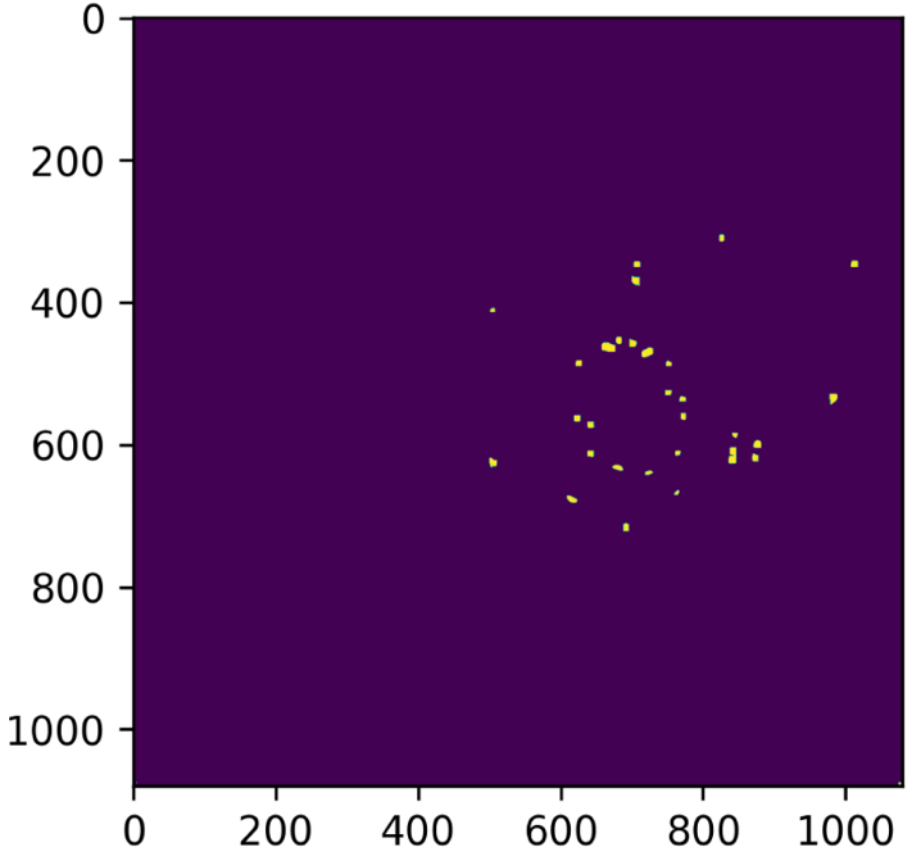
1. Compute the gradient at each point in the image
2. Compute the H matrix from the elements of th- gradient
3. Compute the scoring function (perhaps by computing the eigenvalues)



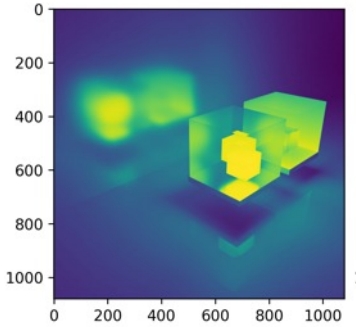
Overview of Harris Corner Detection



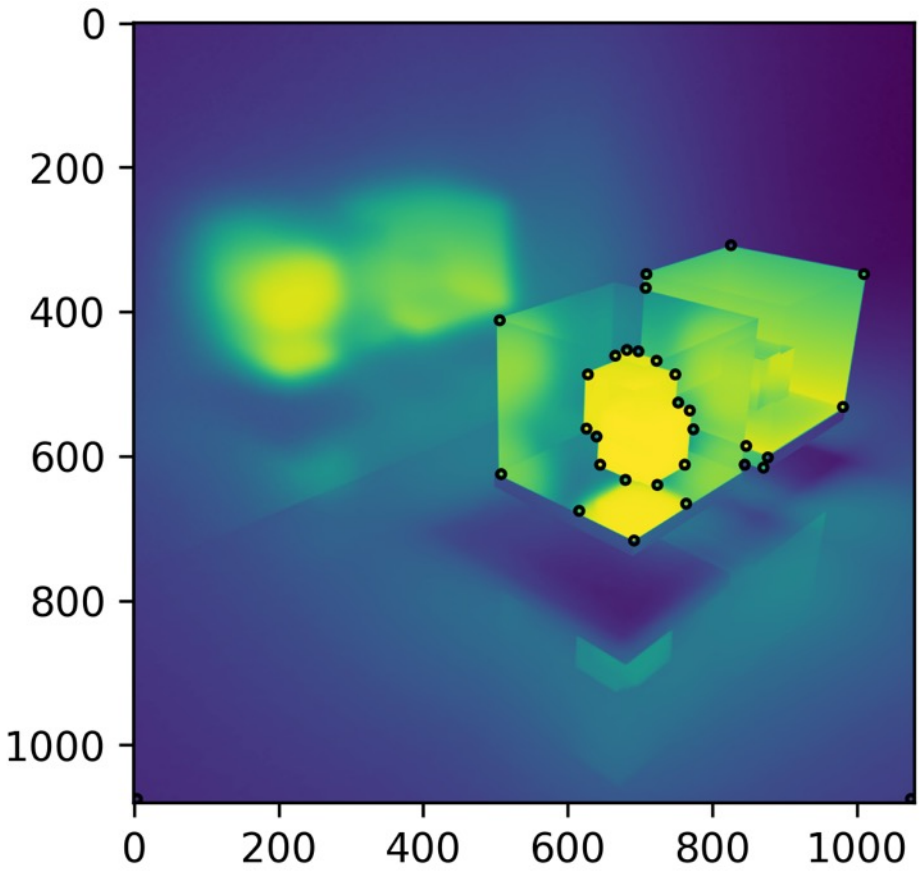
1. Compute the gradient at each point in the image
2. Compute the H matrix from the elements of the gradient
3. Compute the scoring function (perhaps by computing the eigenvalues)
4. Find points with a large response (threshold f)



Overview of Harris Corner Detection

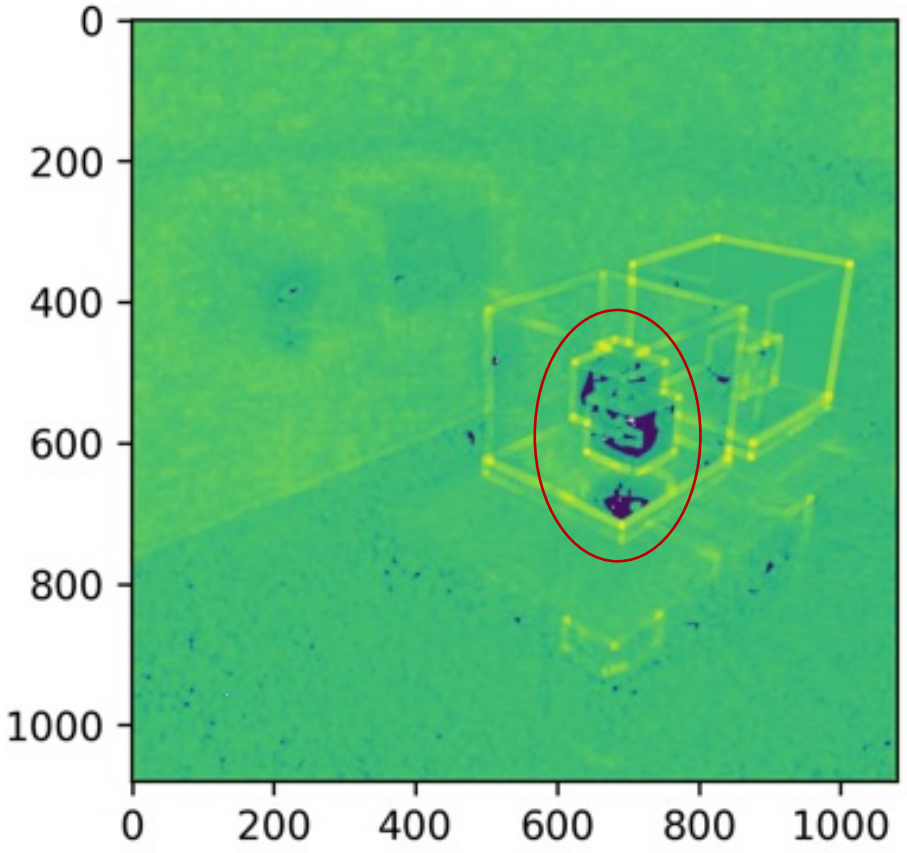


1. Compute the gradient at each point in the image
2. Compute the H matrix from the elements of the gradient
3. Compute the scoring function (perhaps by computing the eigenvalues)
4. Find points with a large response (threshold f)
5. Find local maxima of f after thresholding



Aside: you may run into numerical issues sometimes.

Sometimes the value of f may be undefined (if the denominator is zero) or your plot of log(f) may look odd (since log(0) == NaN). As long as it doesn't change where your corners come from, don't worry about it.

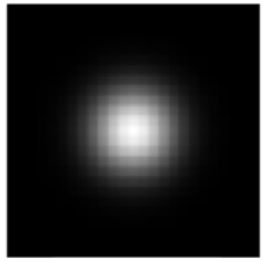


In practice, we weight the derivatives (since a sum/average doesn't work too well)

We can weight the pixels based on their distance from the center pixel.

$$H = \sum_{(x,y) \in W} \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix}$$

$$H = \sum_{(x,y) \in W} w_{x,y} \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix}$$

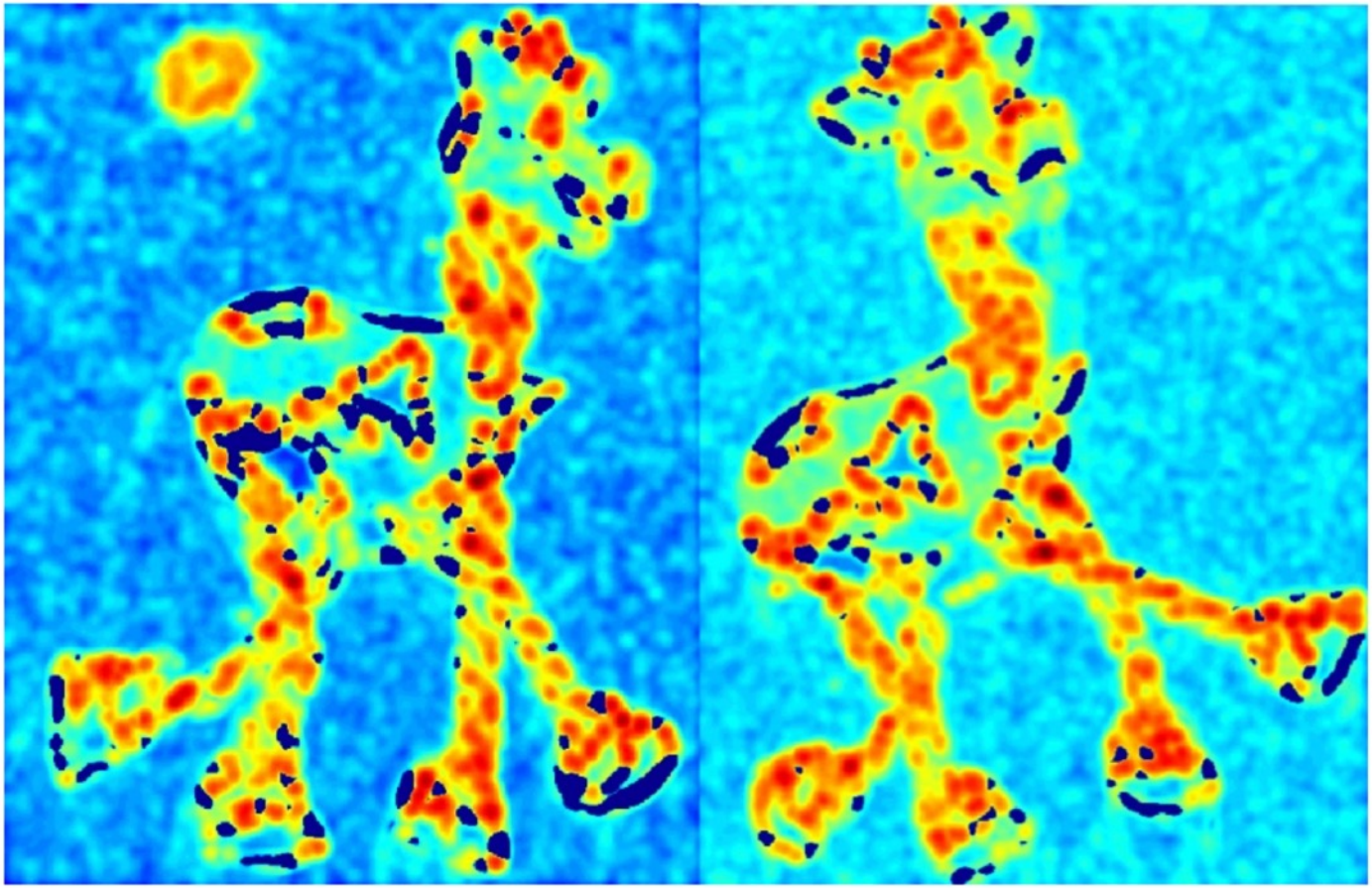


$w_{x,y}$

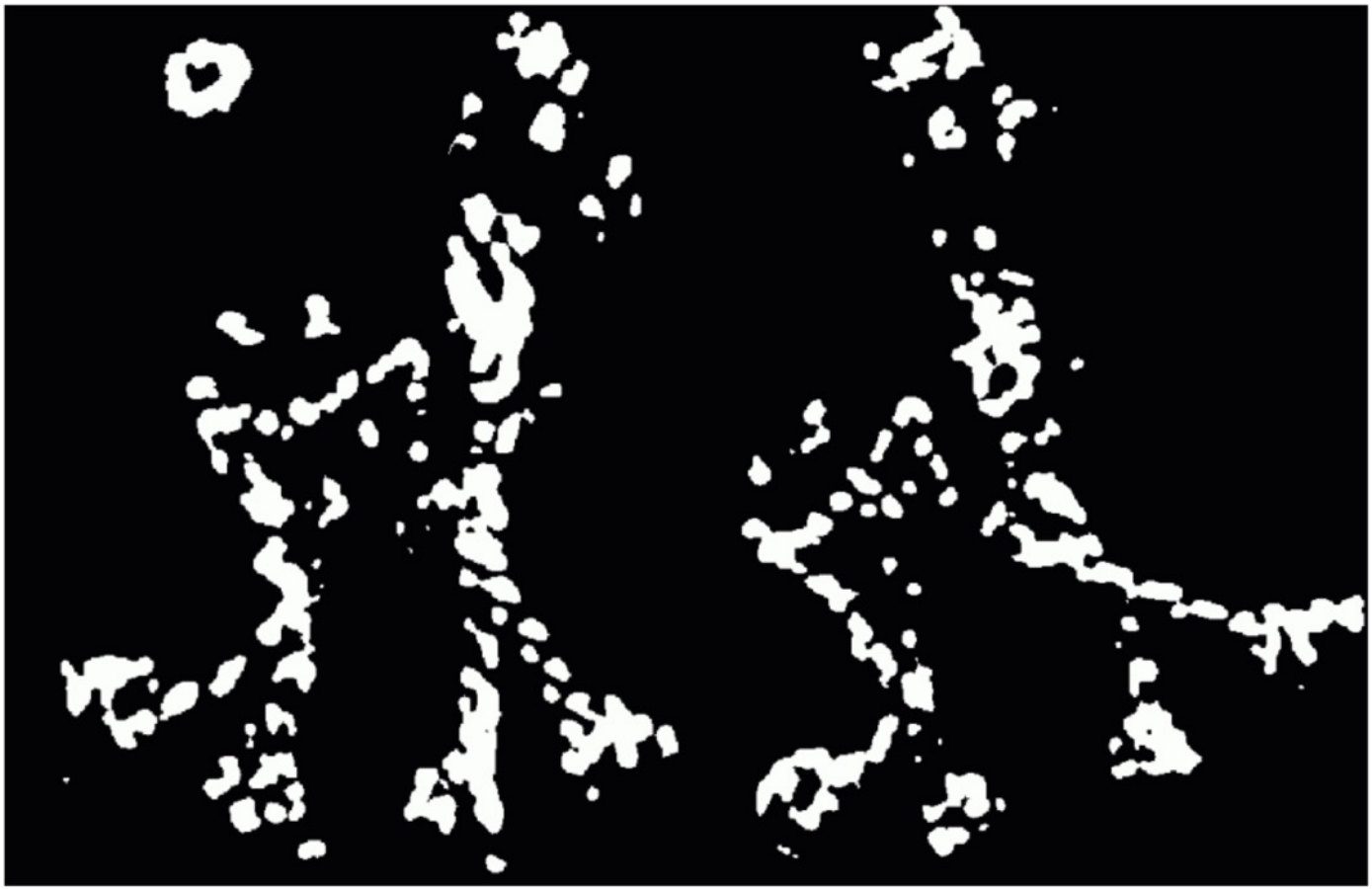
A Gaussian Kernel is a common choice.

Another Harris Detector Example
Image

Another Harris Detector Example
f value



Another Harris Detector Example
threshold f



Another Harris Detector Example
Harris Features (Corners)

Another Harris Detector Example
Harris Features (Corners)

Notice that the features are in similar locations between the two images.

L05B Harris Corners

Now that you've learned about Harris Corners, you're going to implement them using the `light_cubes_sm.png` image I have provided.

1. Compute the gradient at each point in the image (using Sobel Filters)
2. Compute the H matrix (or at least the elements of it) from the elements of the gradient
3. Compute the scoring function (perhaps by computing the eigenvalues)
4. Find points with a large response (threshold f)
5. Find local maxima of f after thresholding

Where the Harris matrix H is defined by

$$H = \begin{bmatrix} A & B \\ B & C \end{bmatrix} = \sum_{(x,y)\in W} w_{x,y} \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix}$$

and I_x is the x-derivative of the image I, I_y is the y-derivative of the image I and w is a weight matrix (start with a mean filter kernel $w_{x,y} = 1$, but in your programming assignment you will be asked to use Gaussian kernel). Use the Sobel filter to compute the image derivatives. You can use either definition of f we introduced in class:

$$f = min(\lambda_1, \lambda_2)$$
$$f = \frac{\det(H)}{\mathrm{tr}(H)}$$

I have provided you with a local-maxima-finding function for you to use. (*Note*: I have tested it, but not extensively...)

In [1]:

```
# Startup code

import numpy as np
import matplotlib.pyplot as plt
from PIL import Image
import scipy.ndimage
import scipy.ndimage.filters as filters

def load_image(filepath):
    """Loads an image into a numpy array.
    Note: image will have 3 color channels [r, g, b]."""
    img = Image.open(filepath)
    return (np.asarray(img).astype(np.float)/255)[:, :, :3]

def get_local_maxima(data, threshold):
    # See: https://stackoverflow.com/a/9113227/3672986
```