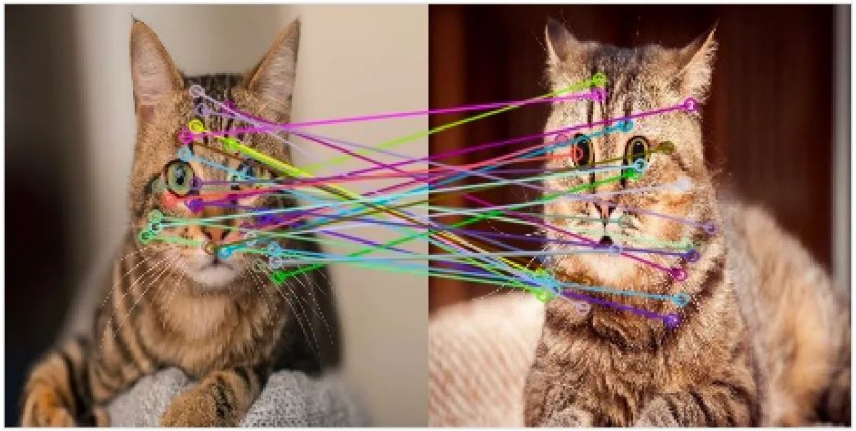
CS-GY 6643
Computer Vision

Lecture 4: Feature detection and matching



Prof. Erdem Varol

Today’s menu

Announcements

Feature detectors

Feature descriptors

Matching

Coding lab

Announcements

HW1 grades and solutions are out

If you have questions/disputes about your HW grades, please contact the TA's within 48 hours. After that we assume you are happy with the grade.

TA: Rishabh Raj

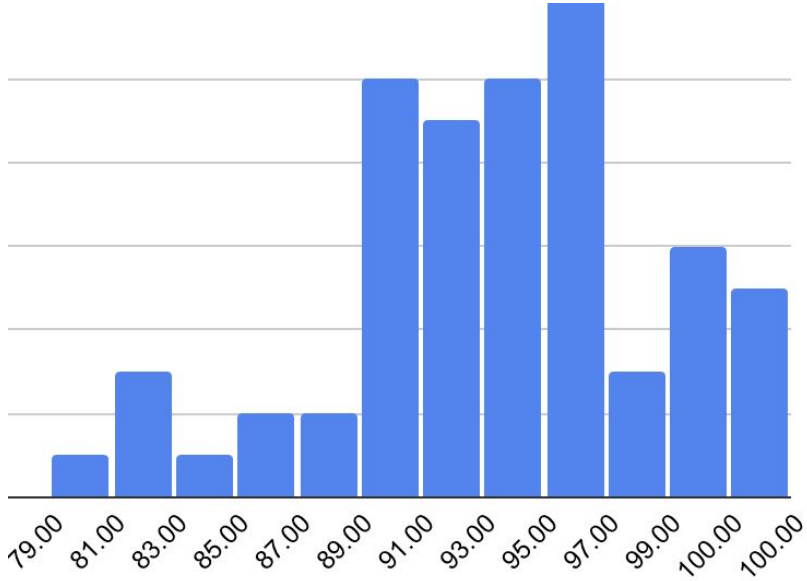
Email: rr4574@nyu.edu

Office Hours: Tuesdays 1-2pm on Zoom.

TA: Malhar Patel

Email: mkp6112@nyu.edu

Office Hours: Mondays 1-2pm on Zoom.



Final project teams are set - check your teams:

https://docs.google.com/spreadsheets/d/1RkjD30T0BsH8VO79zaxYJUE6GJFWTgVNu7nE4SJ1UYw/edit?usp=sharing

	A	B	C	D	E	F	G	H	I	J	K
1	**Project area**	**Vacancy**	**Project tentative title**	**Team member 1 name**	**Team member 1 email**	**Team member 2**	**Team member 2 email**	**Team member 3 name**	**Team member 3 email**	**Team member 4 name**	**Team member 4 email**
2	Object tracking	0	Cell tracking in C. elegans microscopy vid	Maren Eberle	mie8014@nyu.edu	Nalini Ramanatha	nar8991@nyu.edu	Ryan Fleishman	rmf9265@nyu.edu	Neel Gandhi	njg9191@nyu.edu
3	Object tracking	0	Ball Tracking in Sports	Pratham Shah	ps4896@nyu.edu	Ohm Patel	odp2008@nyu.edu	Varad Naik	vvn7114@nyu.edu	Jugal Pumbhadia	jp6988@nyu.edu
4	Image Reconstruction	0	TBA	Jaydeep Mulani	jgm9413@nyu.edu	Aditya Azad	aa10878@nyu.edu	Sidhved Warik	sw6071@nyu.edu	Vedangi Kirtane	vrk3359@nyu.edu
5	Image Alignment	0	Multimodal registration of in-vivo vs. ex-viv	Tianxiao He	th3129@nyu.edu	Yurong Liu	yl6624@nyu.edu	Jialin Li	jl10897@nyu.edu	Remo Shen	ys6345@nyu.edu
6	Superresolution	0	TBA	Shambhavi Seth	ss17936@nyu.edu	Aryan Prasad	ap7949@nyu.edu	Roman Vakhrushev	rv1057@nyu.edu	Ishan Yadav	iy2159@nyu.edu
7	Object tracking	0	Cell detection and tracking	Jizheng Dong	jd4587@nyu.edu	Yifei Zhuang	yz9865@nyu.edu	Bowen Gong	bg1941@nyu.edu	Lucy Shi	ls4900@nyu.edu
8	Image Colorization	1	TBA	Nika Emami	ne2213@nyu.edu	Mehran Ali Banka	mab10251@nyu.edu	Ansal Arimbassery	aa11164@nyu.edu	ASSIGN ME TEAMMATES	
9	Object Detection	0	Character Detection in Videos	Wei-Lun Hung	wh2528@nyu.edu	Yu-Yuan Chang	yc6549@nyu.edu	Guan-Cherng Lin	gl2547@nyu.edu	Da Hye Kim	dk3343@nyu.edu
10	Object tracking	0	Biospecimen tracking	Gordon Lu	gl1589@nyu.edu	Suemy Inagaki	si2324@nyu.edu	Felipe Oliveira	fd2264@nyu.edu	Chhatrapathi Sivaji Lakk	cl7203@nyu.edu
11	Semantic Segmentation	0	TBA	Daoyu Li	dl5312@nyu.edu	Ziming Song	zs2815@nyu.edu	Xuning Chang	xc1626@nyu.edu	Allison Lee	al6897@nyu.edu
12	Object tracking	0	Haol	Aravindsrinivas Krishnar	ak11115@nyu.edu	Priyangshu Pal	pp2833@nyu.edu	Charan K Raja	ck3740@nyu.edu	Bharat Somasundaram	bs4852@nyu.edu
13	Object detection	3	TBA	KuanYu Chen	kc5405@nyu.edu	ASSIGN ME TEAMMATES		ASSIGN ME TEAMMATES		ASSIGN ME TEAMMATES	
14	Object Detection	0	Gesture Recognition System	Pranav Mohril	pm3727@nyu.edu	Akshat Shaha	as16655@nyu.edu	Sumedh Parvatikar	sp7479@nyu.edu	Ashutosh Kumar	ak10514@nyu.edu
15	Object detection	0	Impact and Collisions	Willem Mirkovich	wjm9297@nyu.edu	Samkit Shah	ss17651@nyu.edu	Haya Diwan	hd2371@nyu.edu	Rohit Prakash	rp3963@nyu.edu
16	Image Diffusion	2	Diffusion Model from Scratch + Use Case	Aakar Mutha	am13480@nyu.edu	Mrityunjay Bhanja	mb9348@nyu.edu	ASSIGN ME TEAMMATES		ASSIGN ME TEAMMATES	
17	Image Diffusion	0	TBA	Songhao Li	sl10500@nyu.edu	Yuyang Chen	yc7065@nyu.edu	Haoliang Zhang	hz2095@nyu.edu	Victor Pou	vmp2063@nyu.edu
18	Object detection	0	Human Pose detection	Shubham Goel	sg4599@nyu.edu	Mohammed Bash	mb9885@nyu.edu	Chirag Mahajan	cm6591@nyu.edu	Nirbhaya Reddy G	ng3033@nyu.edu
19	Image Reconstruction	1	TBA	Joshua Alfred Jayapal	jj3811@nyu.edu	Adel Del Valle	ad7082@nyu.edu	Adith Santhosh	at6115@nyu.edu	ASSIGN ME TEAMMATES	

Final project proposal due Oct 10

20% of your final project grade (4 out of 20 pts)

Proposal template here:

https://www.overleaf.com/read/rhdrqnmsyzpv#c1b706

(Please copy this template in your own account)

Must complete as a team - one person may submit on brightspace.

Computer Vision CS-GY 6643 - Final Project - Project Title

Jane Doe, jd1234@nyu.edu,John Doe, jd1235@nyu.edu, Bob Doe, bd1234@nyu.edu, Mary Doe, md1234@nyu.edu

1 Introduction and background

Introduce the problem statement, importance, background on how others have tried to tackle it. Cite references (Bommasani et al. 2021; Han et al. 2021). If we were to solve this problem, what can we gain scientifically and methodologically? Why has this problem been difficult to solve before and why do you think now is the time to solve it (e.g. before we didn't have the right type of sensors or computing resources, now we do). Briefly introduce how your proposed approach is going to overcome the previous obstacles.

2 Datasets

Describe the datasets that will be used for this study. Generate some figures to illustrate how they look like, what kind of noise we expect to see, class imbalance, missing and incomplete features etc.

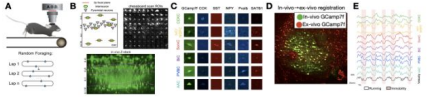


Figure and caption

3 Methods

Introduce the mathematical formulation of the problem (if applicable) or describe the model architecture using diagrams.

4 Evaluation metrics

Which metrics will you use to measure the success of the method. Are there literature evidence to suggest these are appropriate metrics?

5 Baseline methods

Which methods will you show as baselines?

6 Preliminary Results

Show how a prototype of the proposed method can work on the datasets either with real data or using a toy example. Also, show that you are able to run a baseline method here.

7 Final project plan

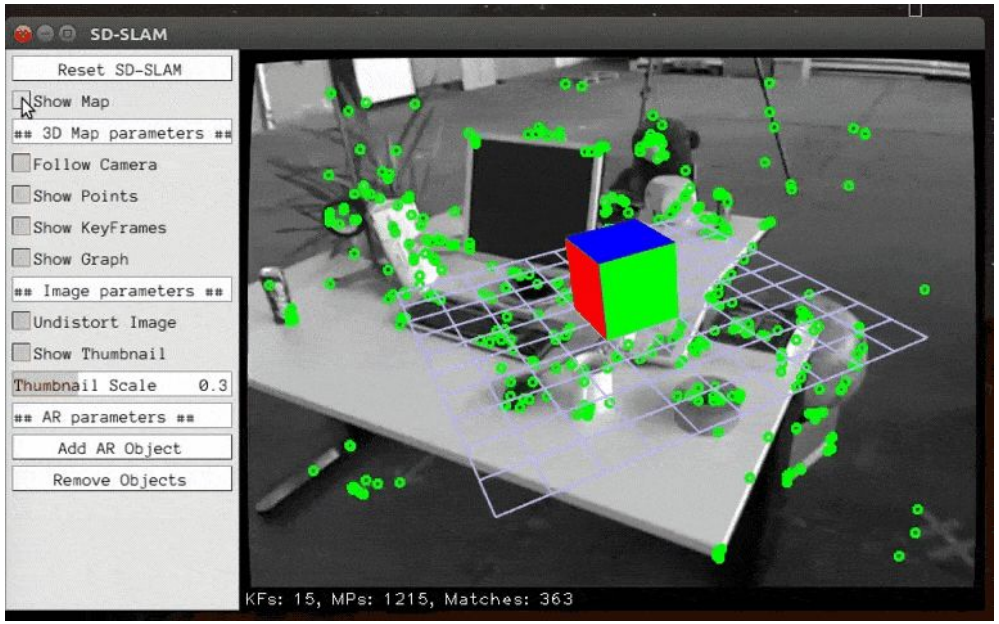
What did you learn from your preliminary experiments? How will this affect how you will cast your model for the final project? It is important to demonstrate here that based on your preliminary results, your final project is something you can execute in the the next two months.

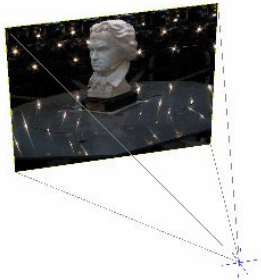
8 Author contributions

You must explicitly state who in the team is going to execute which parts of the project.

Lecture

Suppose we want to match two images (or series of images) , how would we go about doing it?





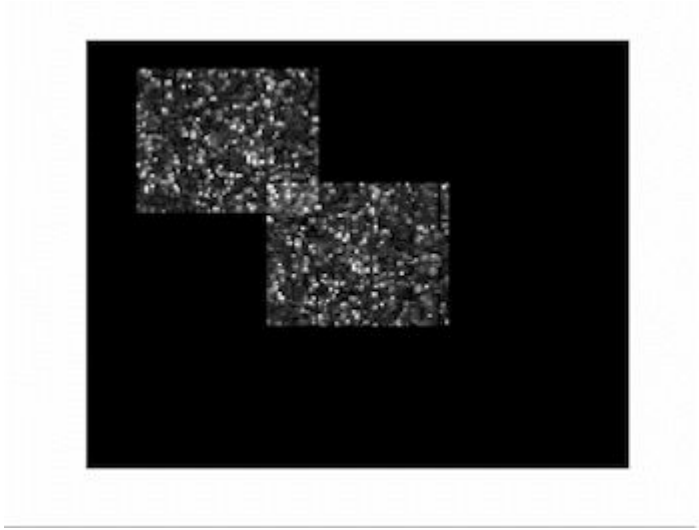
Specific example

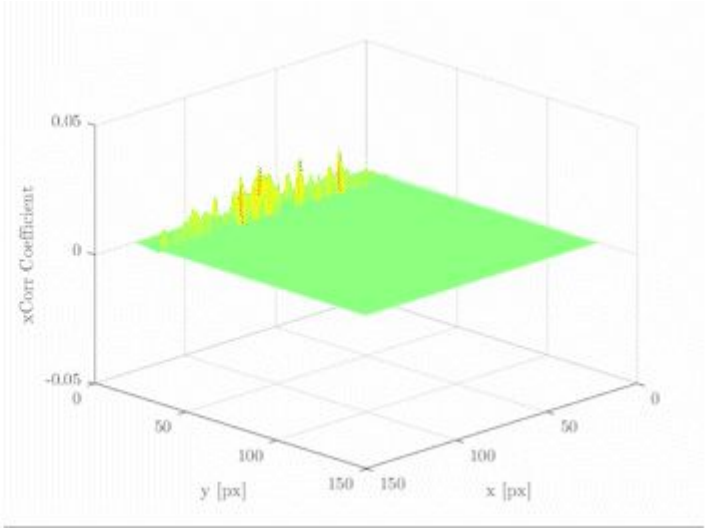
Given two images: how do you align them?

One possible solution

```
for y in range(-ySearch,ySearch+1):
    for x in range(-xSearch,xSearch+1):
        #Touches all HxW pixels!
        check_alignment_with_images()
```

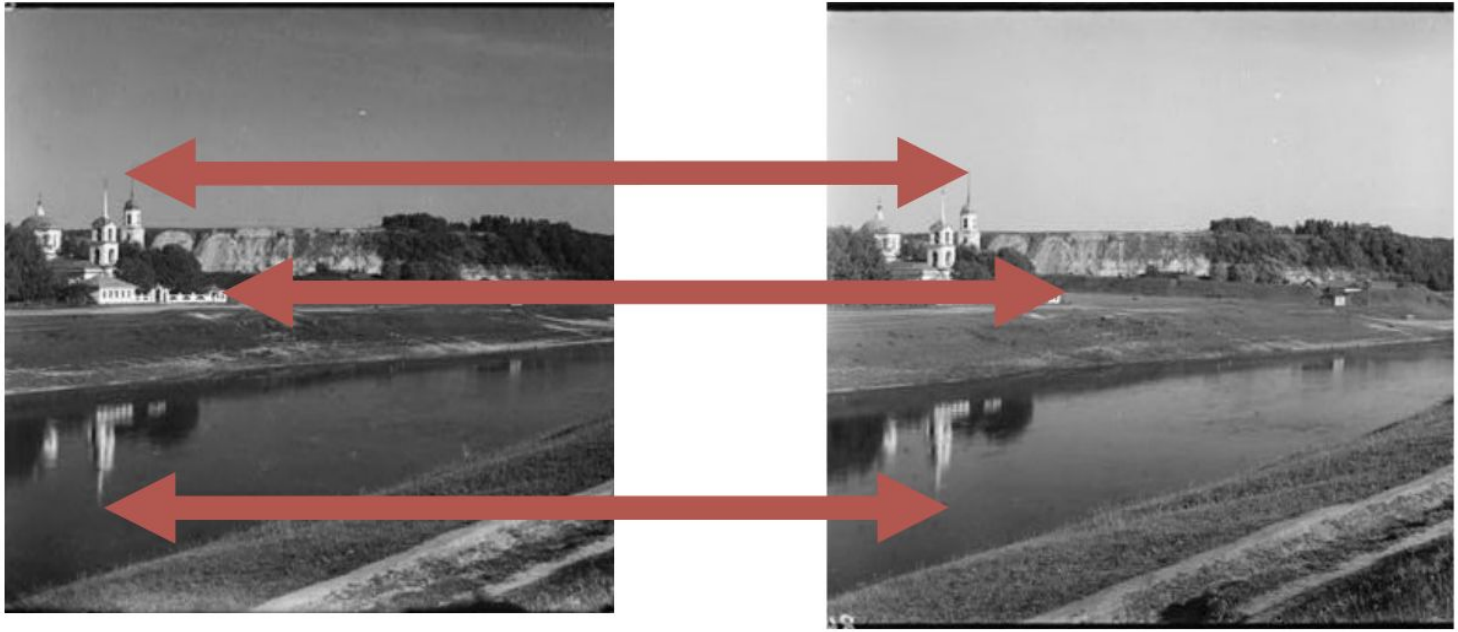
Recall this can be done using cross-correlation + thresholding (from previous lecture)





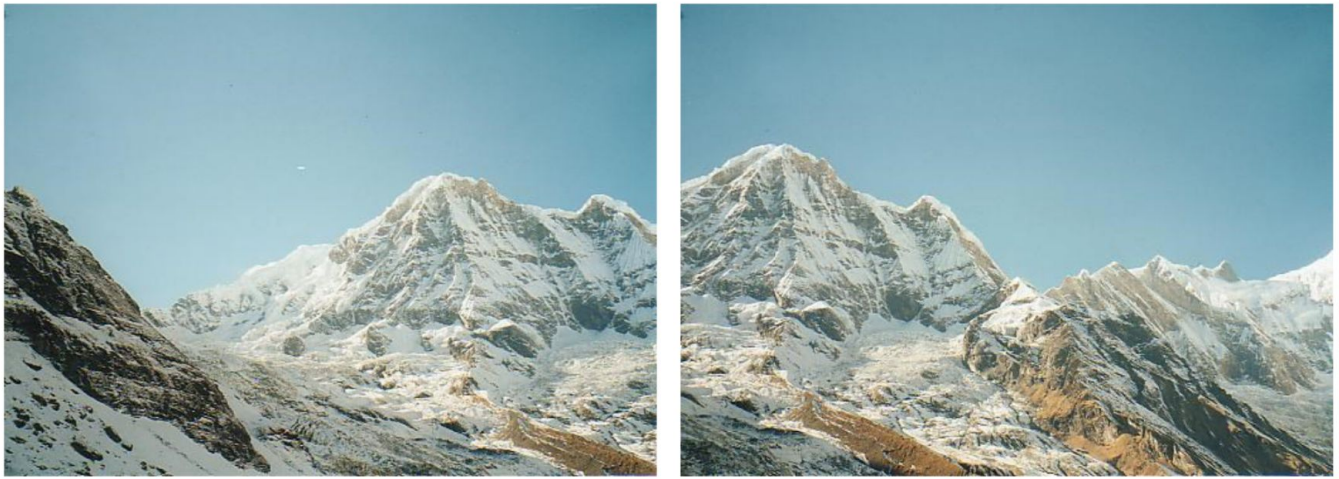
A bit more efficient - but relies on a key assumption…

We can easily find the translation here:



A harder example

Given these images: how do you align them?



These aren't off by a small 2D translation but instead by a 3D rotation + translation of the camera.

Another possible solution

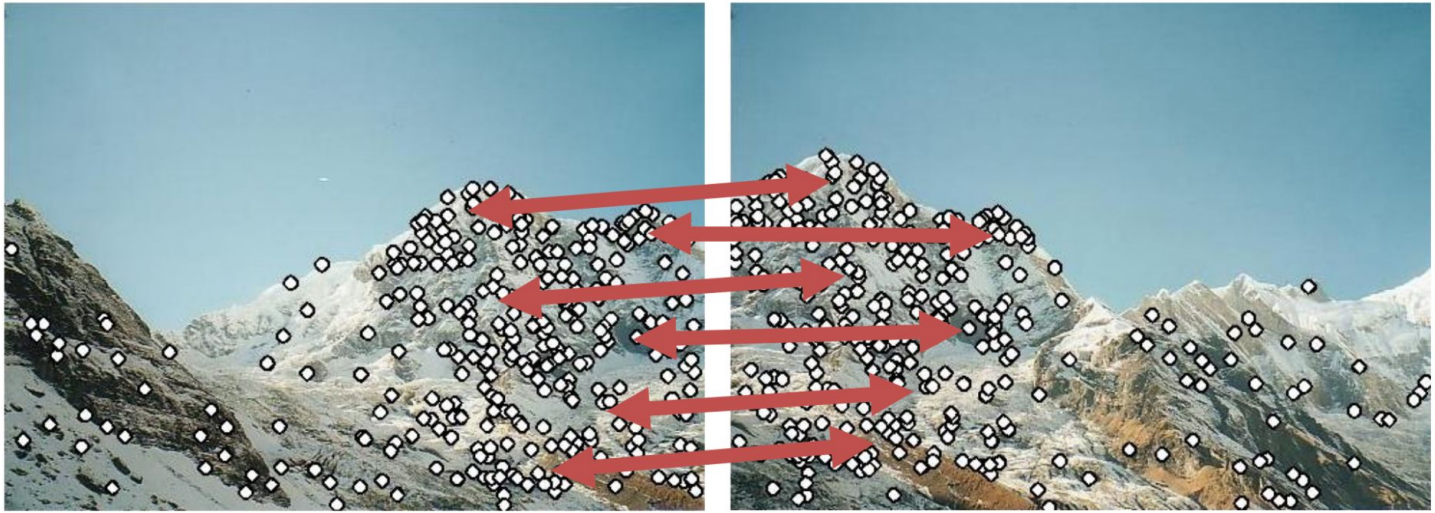
```
for y in yRange:
    for x in xRange:
        for z in zRange:
            for xRot in xRotVals:
                for yRot in yRotVals:
                    for zRot in zRotVals:
                        #touches all HxW pixels!
                        check_alignment_with_images()
```

Very inefficient and not at all how us humans “match” objects visually

Alternative human solution

Given these images: how would **you** align them?

Finding and matching process broken into steps



1) Detect “important” points
2) Describe them
3) Match points with similar descriptions

Finding and matching in algorithmic steps

Detect, and describe important points in image 1
Detect, and describe important points in image 2

for i in “important” in points in image 1
    for j in “important” in points in image 2
        if is_close(description of “important” point i in image 1,description of “important” point j in image 2)
            matched_points<-[matched_points (i,j)]

Note: the search space is reduced to set of “important” points rather than all pixels and rotations

How can we reduce the search space of matching points?

¡ Filtering !

Recap of previous lecture

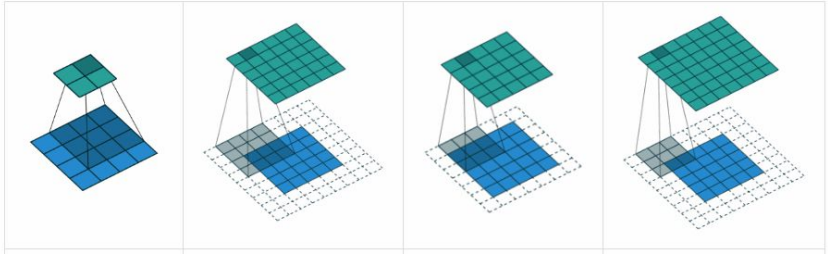
Filters help enhance or suppress image details (e.g. smoothing, sharpening)

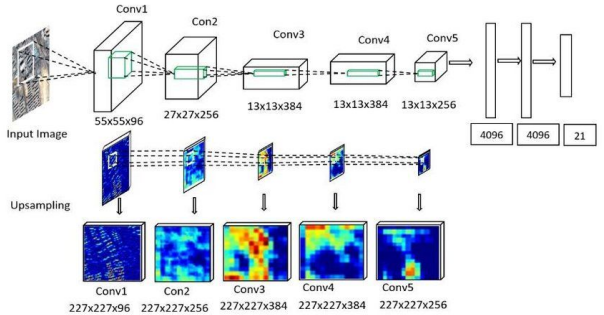
Convolutions are the mathematical operators behind almost all filtering

Filters can be stacked to yield higher level details

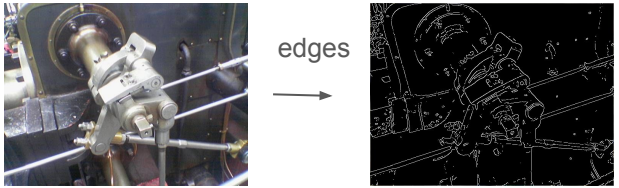
Edge detectors are a combination of gradient filters e.g. canny edge detector

Edges are low-level image features





Low level → High level

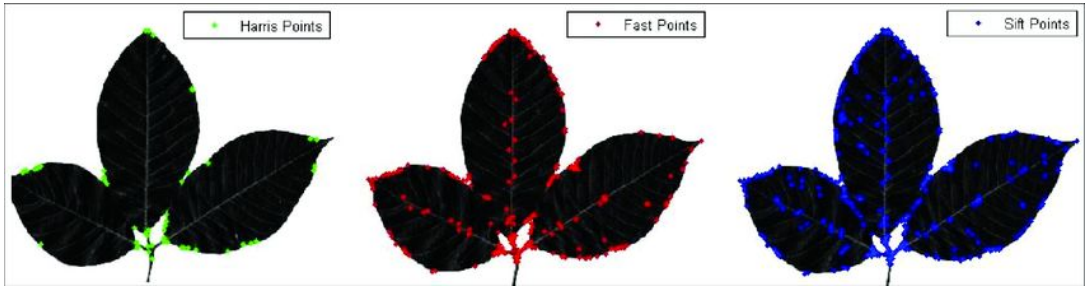


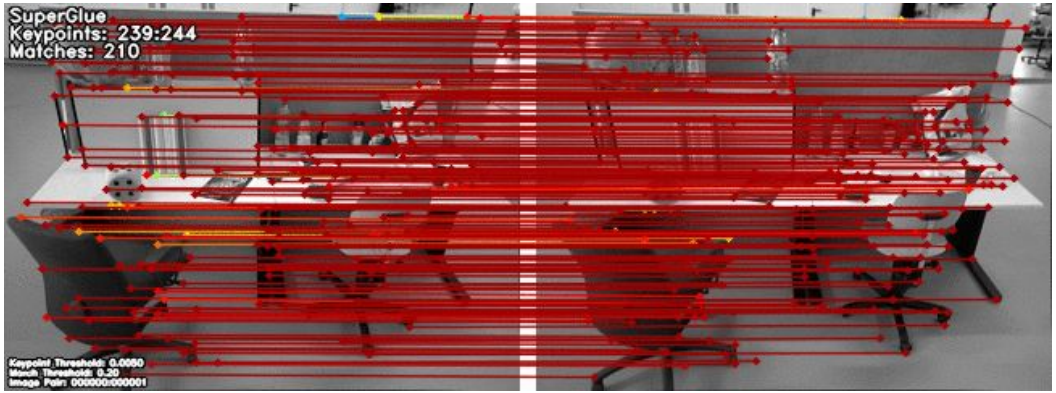
Today we will learn

How to reduce important parts of images to keypoints

How to describe keypoints using perturbation invariant features

How to match keypoints across images



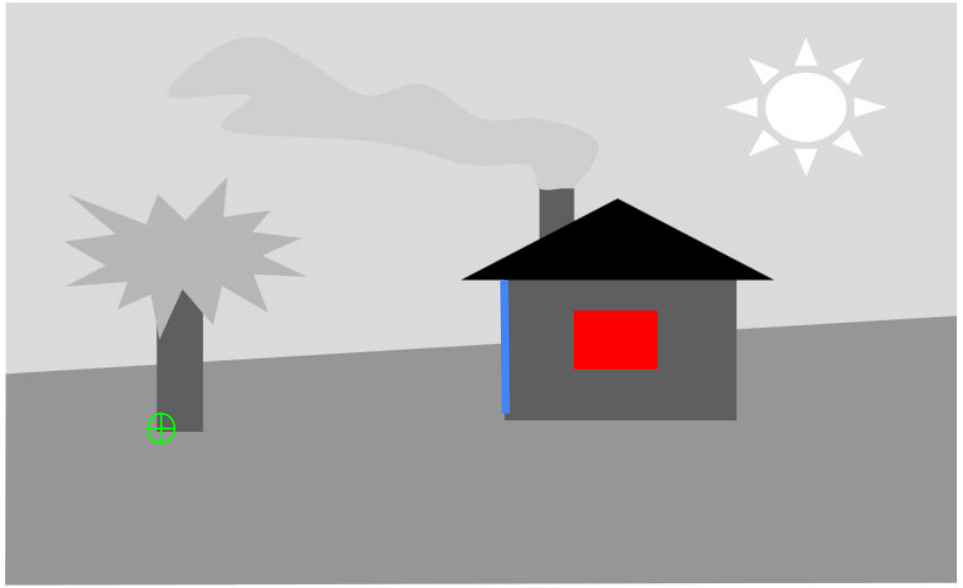


Recall the piecewise flat model of images

There is sparse amount of detail in each image quantified by flats, edges and corners.

We don’t know where these are a priori.

Solution: Detectors!

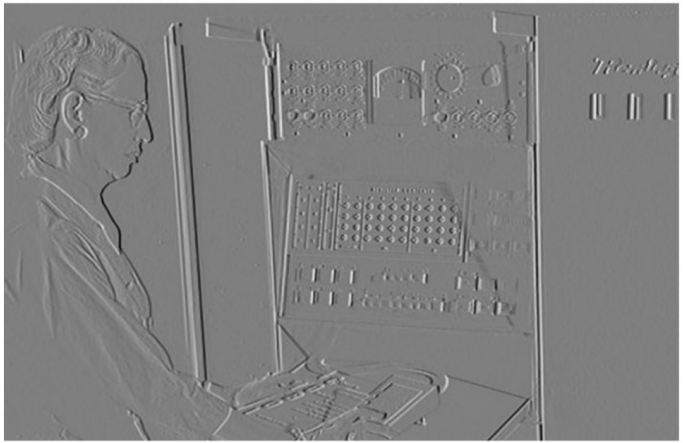


First level filter - gradients

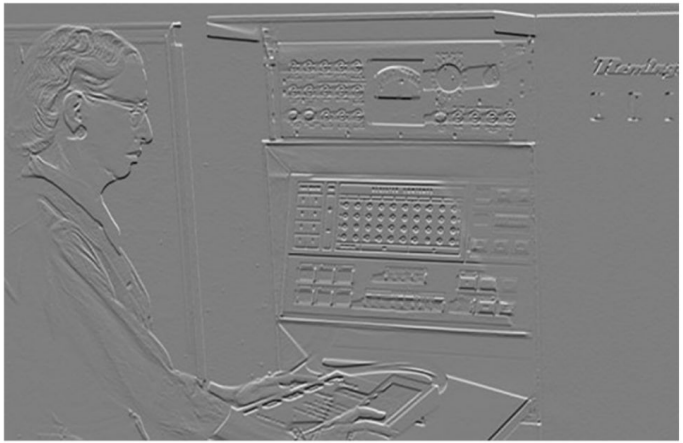
Gradient: direction of maximum change.

What's the relationship to edge direction?

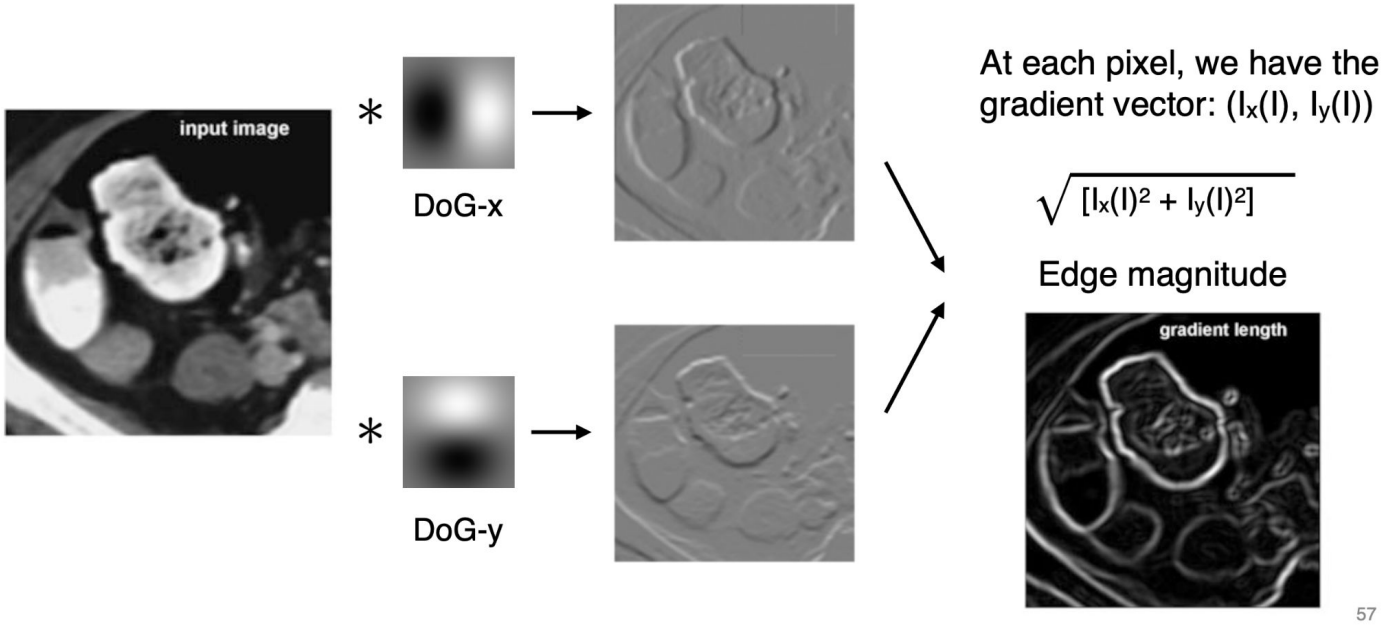
Ix



Iy



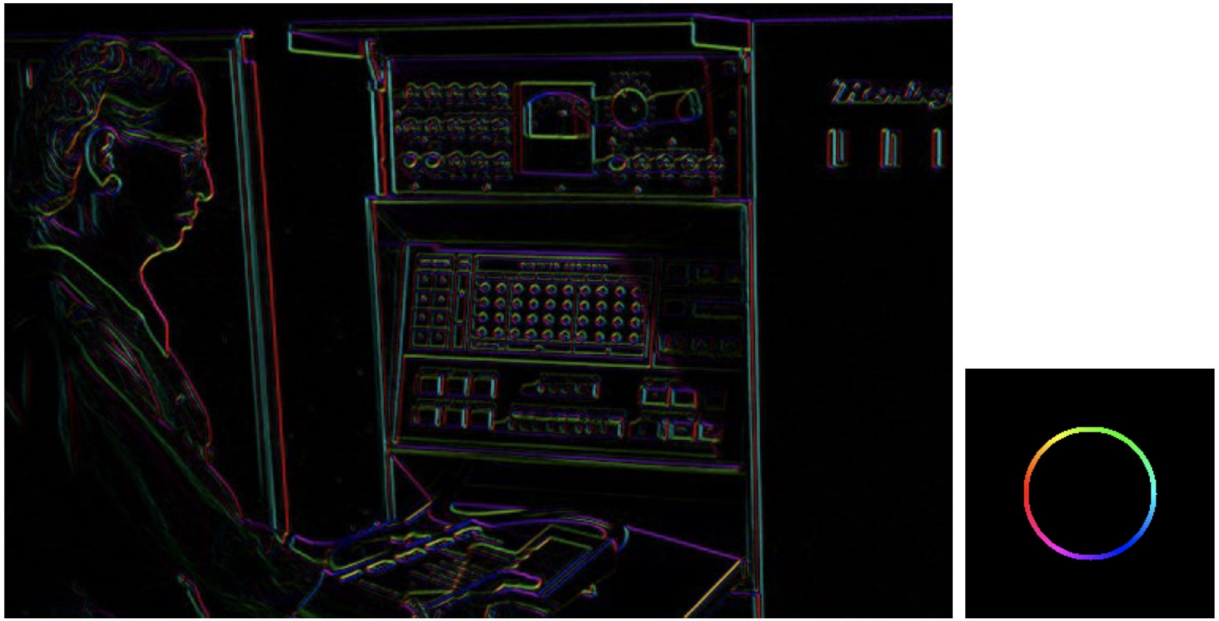
Magnitude of gradients at each pixel → threshold → edge detection



57

We can describe edges by their orientation:

atan2(Iy,Ix): orientation



If we want to match two images, are edges helpful?

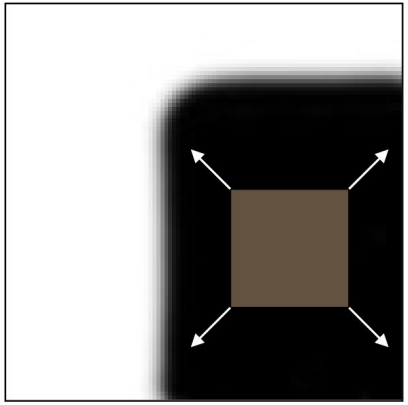
Sometimes…

Often, edge images are not sparse enough.

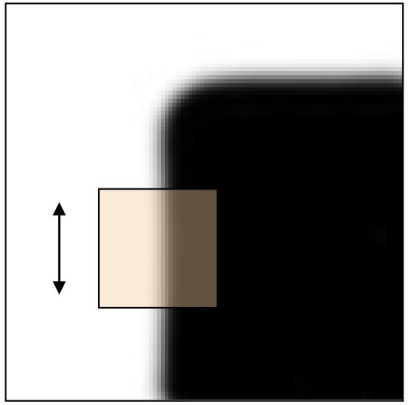
Too many pixel combinations to consider.

```
for i in edge points in image 1
        for j in edge points in image 2
                if is_close(description of edge point i in image
1,description of edge point j in image 2)
                        matched_points<-[matched_points (i,j)]
```

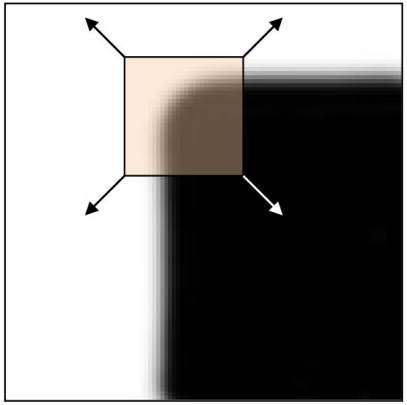
Edges are denser image features than corners



"flat" region:
no change in
all directions



"edge":
no change
along the edge
direction



"corner":
significant
change in all
directions

What we want out of image feature detectors

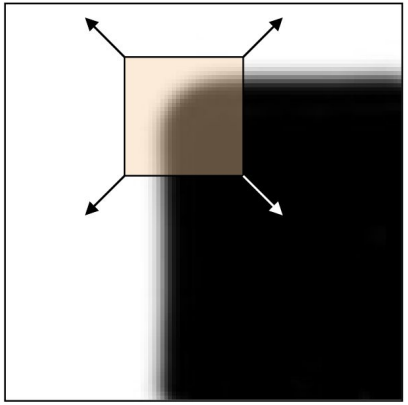
- Repeatable: should find same things even with distortion
- Saliency: each feature should be distinctive
- Compactness: shouldn’t just be all the pixels
- Locality: should only depend on local image data

Harris corner detector

Let's start by building a corner detector from first principles

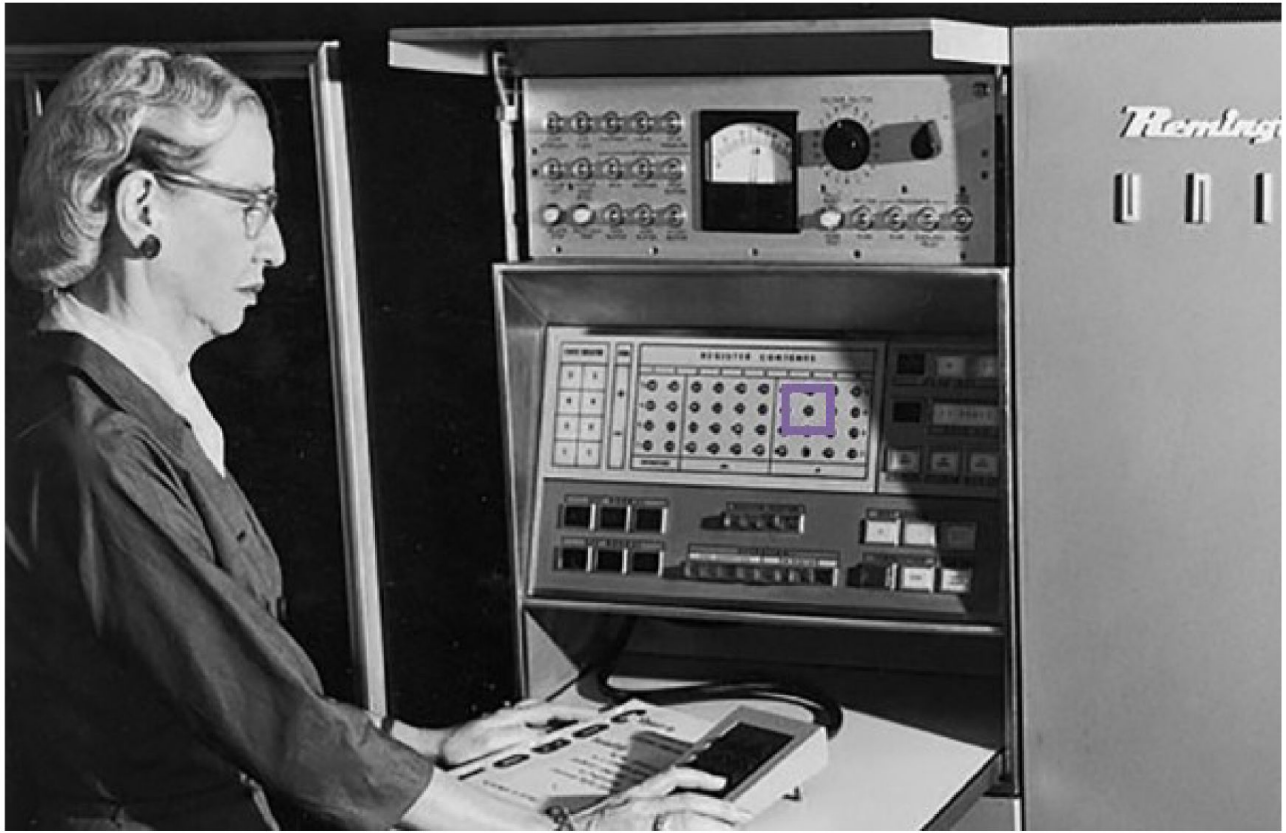
"flat" region:
no change in
all directions

"edge":
no change
along the edge
direction

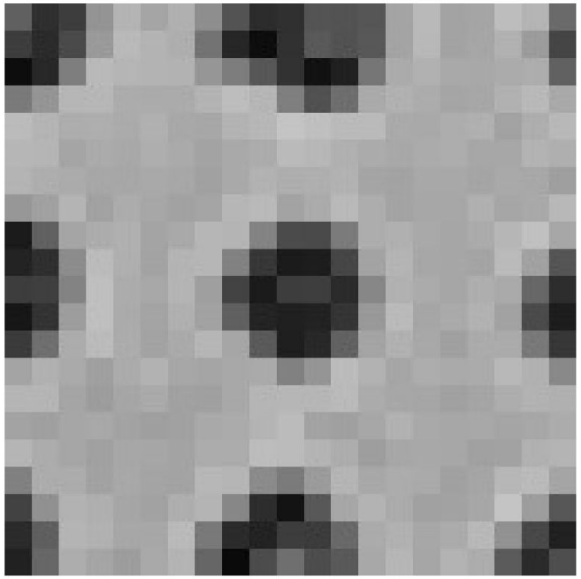


"corner":
significant
change in all
directions

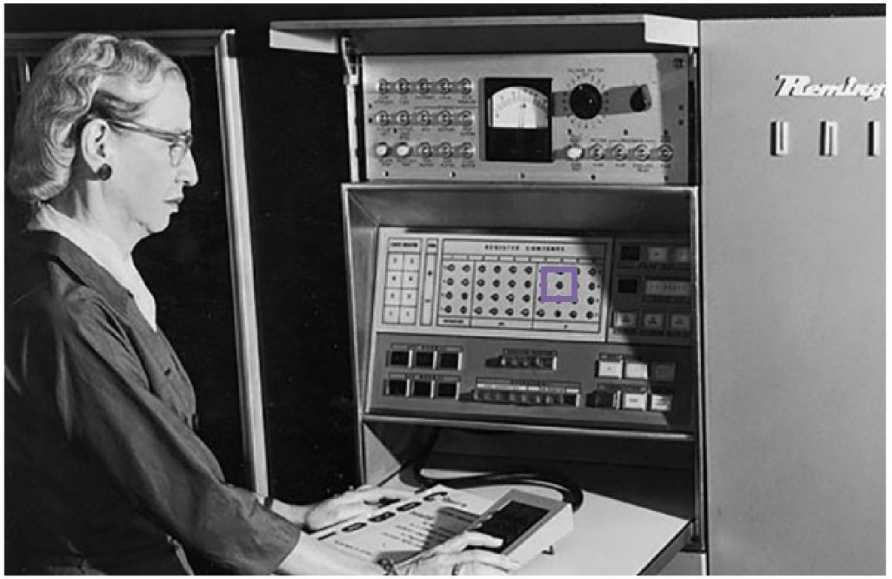
Is this a corner?



Zoom-In at x,y



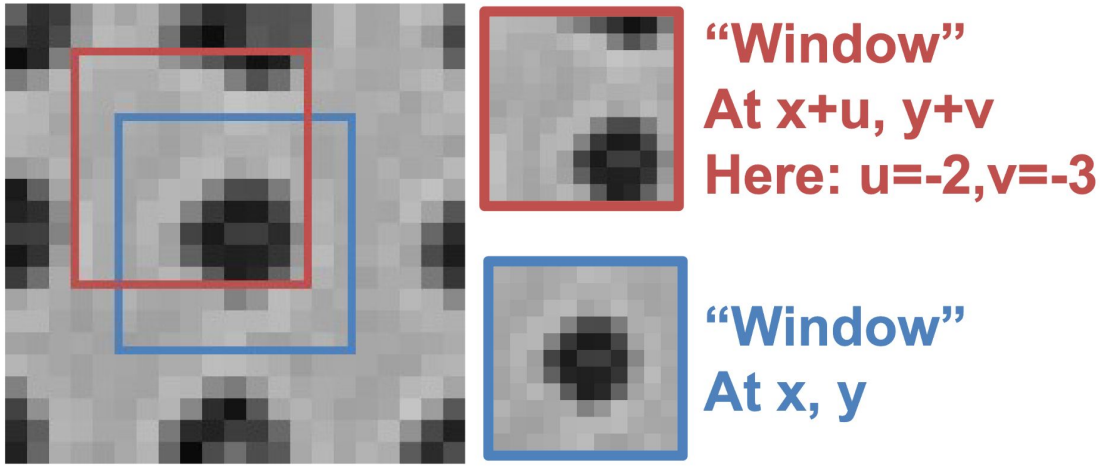
Original Image



Intuition: we want to find pixels where changing in any direction causes large error

Zoom-In at x,y

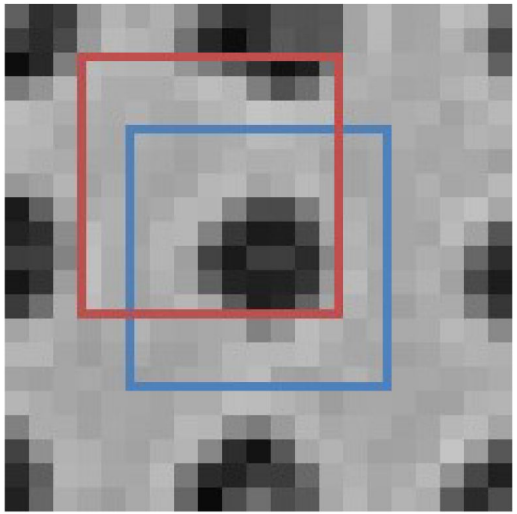
Window without and with Offset



How might we measure similarity?

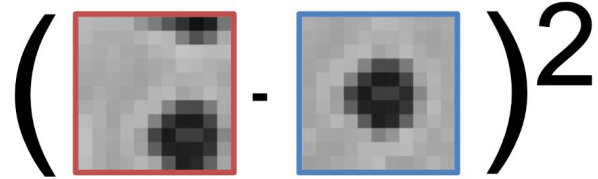
For each pixel (x,y), compute error at arbitrary local shifts (u,v)

Zoom-In at x,y

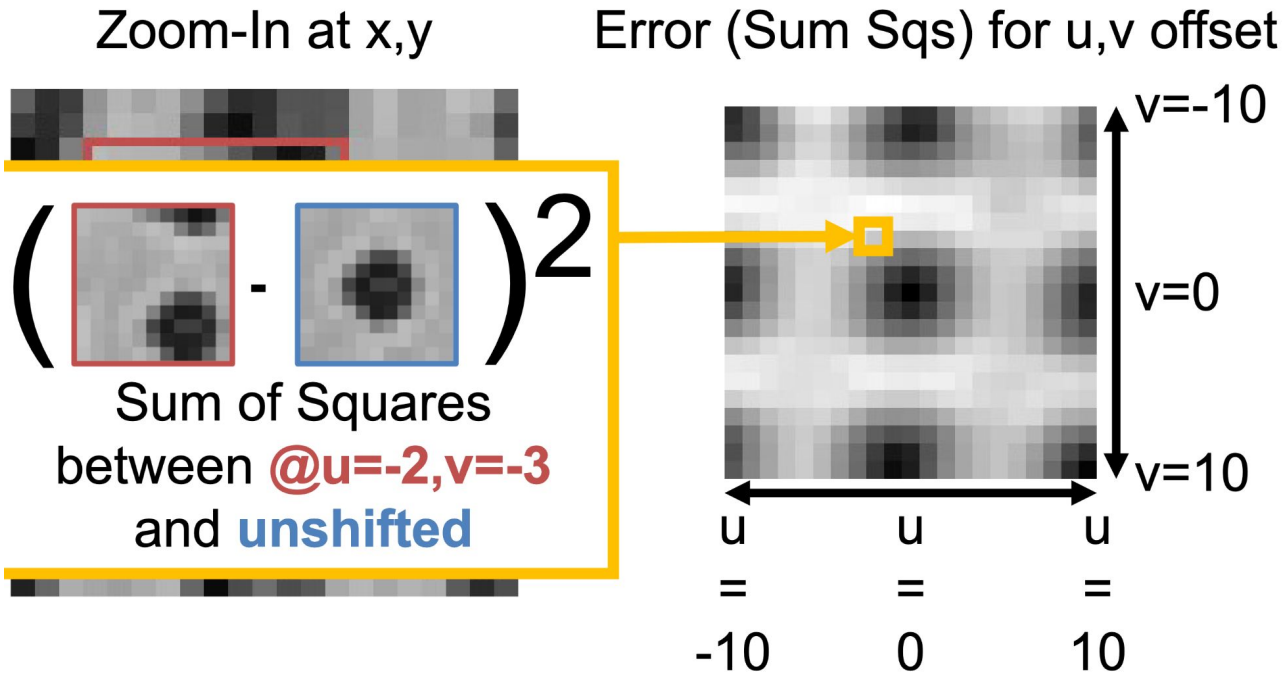


Error (Sum Sqs) for u,v offset

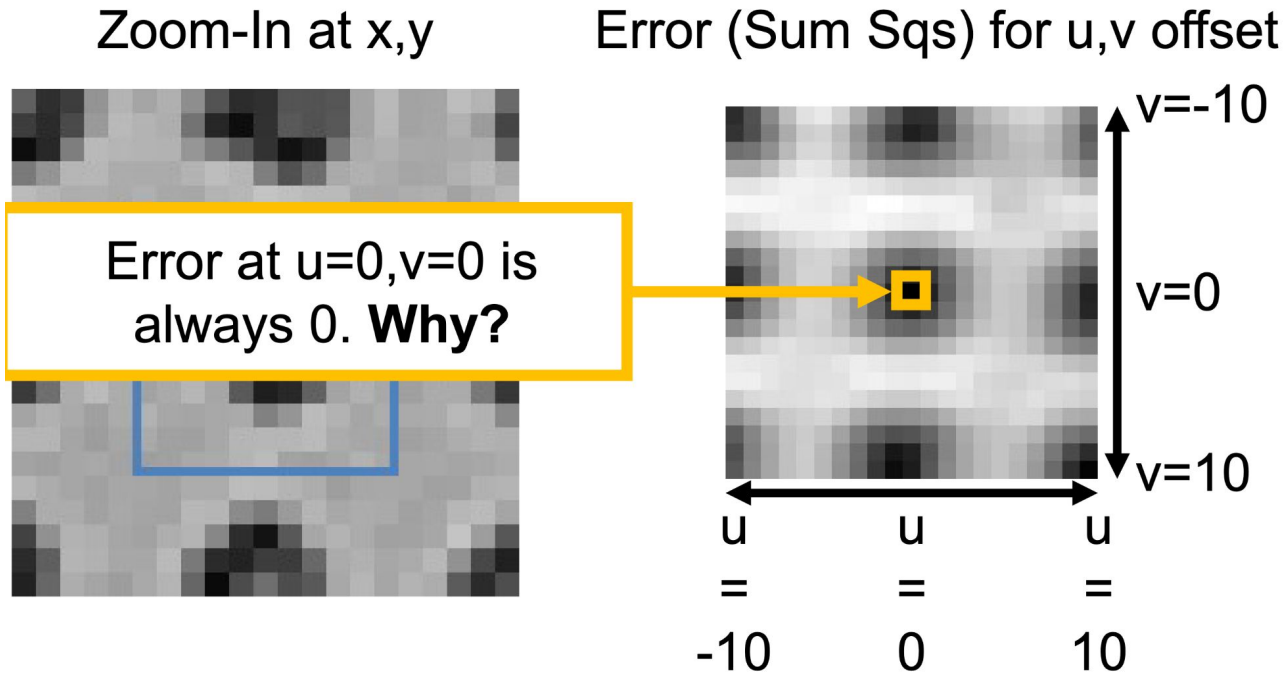
$$E(u,v) = \sum_{(x,y)\in W} (I[x+u, y+v] - I[x,y])^2$$



For each pixel (x,y), compute error at arbitrary local shifts (u,v)

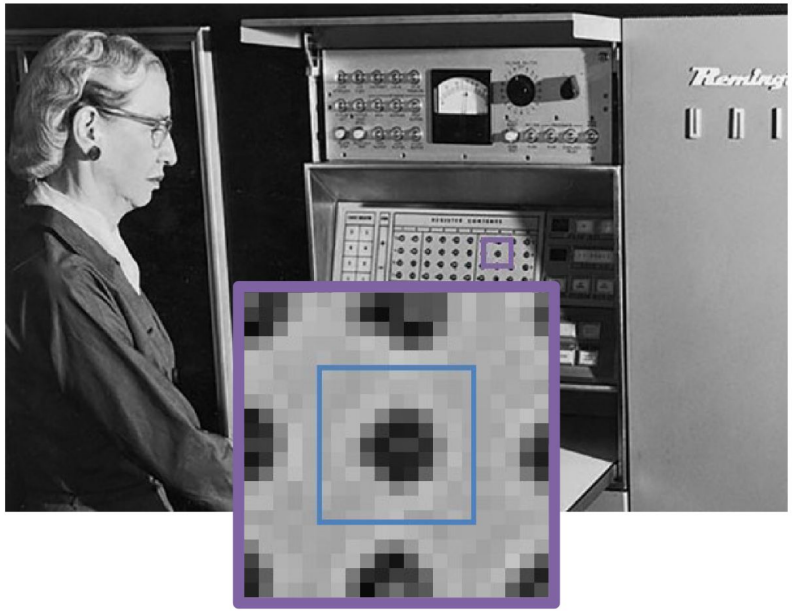


For each pixel (x,y), compute error at arbitrary local shifts (u,v)



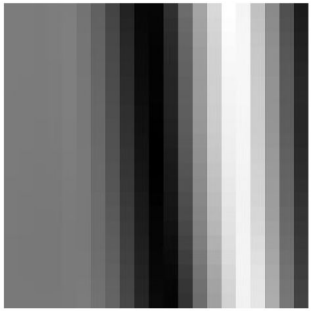
For the query patch in purple, what is the local shift error look like?

Original Image and Zoom-In

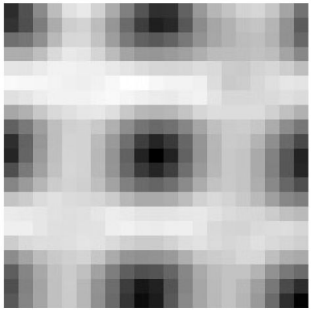


Error Options

A

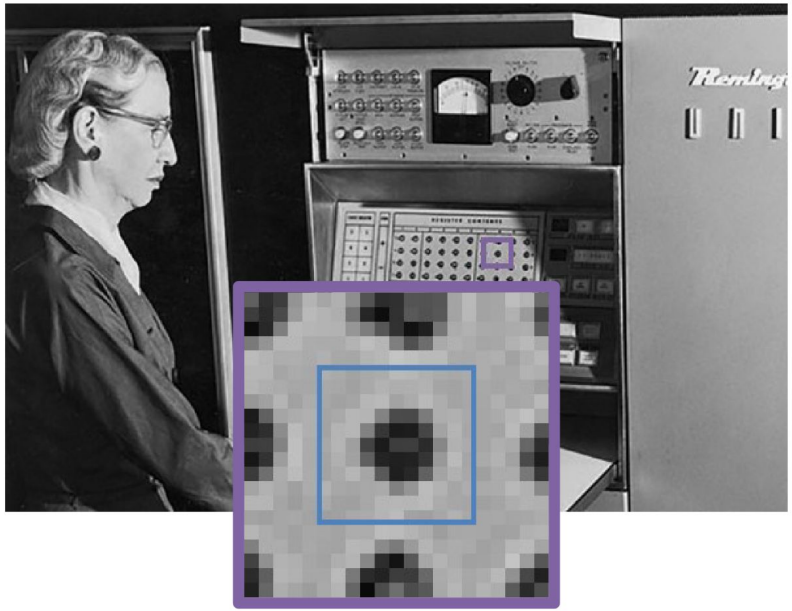


B

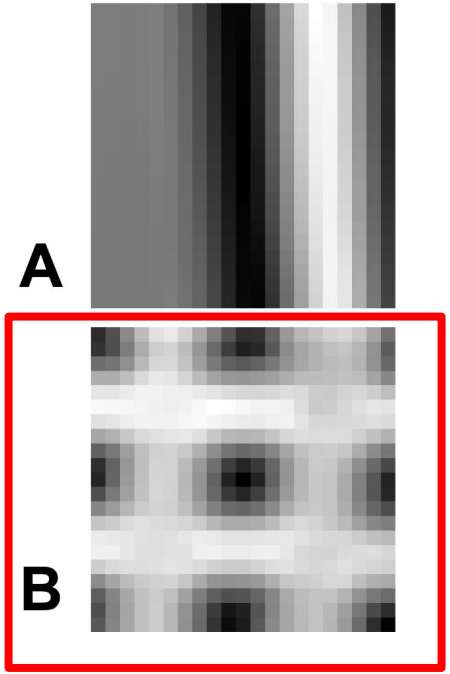


For the query patch in purple, what is the local shift error look like?

Original Image and Zoom-In

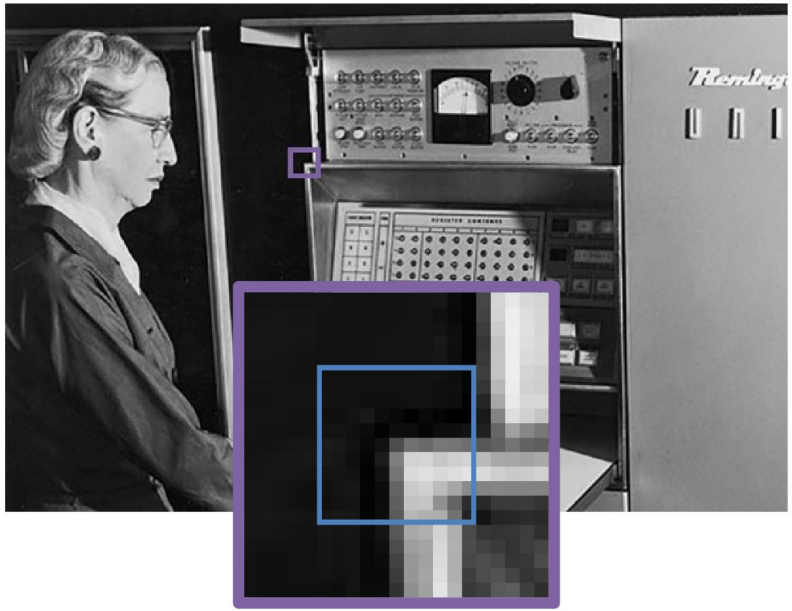


Error Options



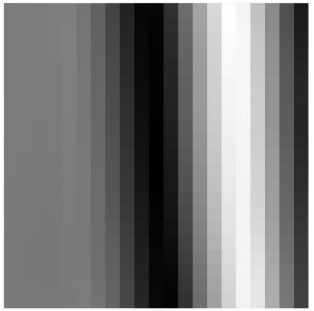
For the query patch in purple, what is the local shift error look like?

Original Image and Zoom-In

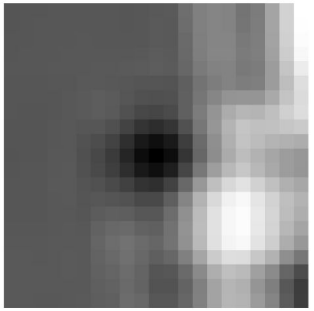


Error Options

A

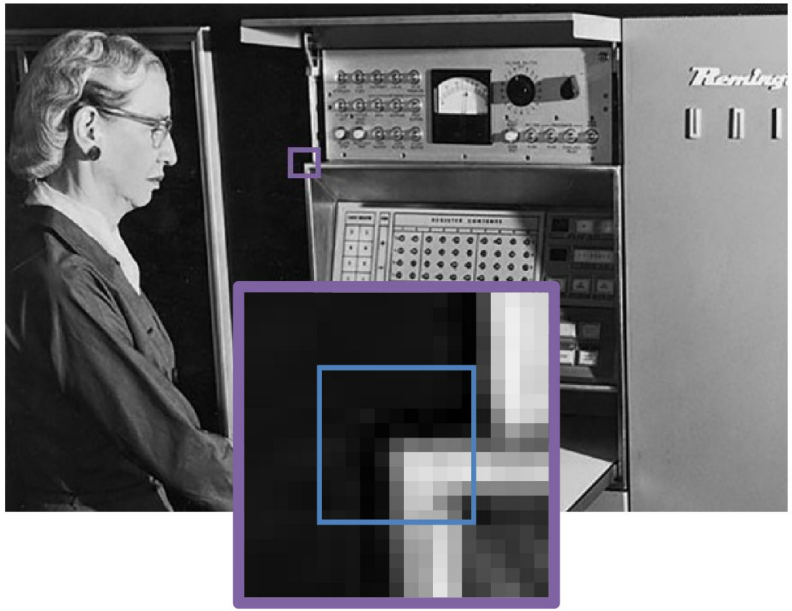


B

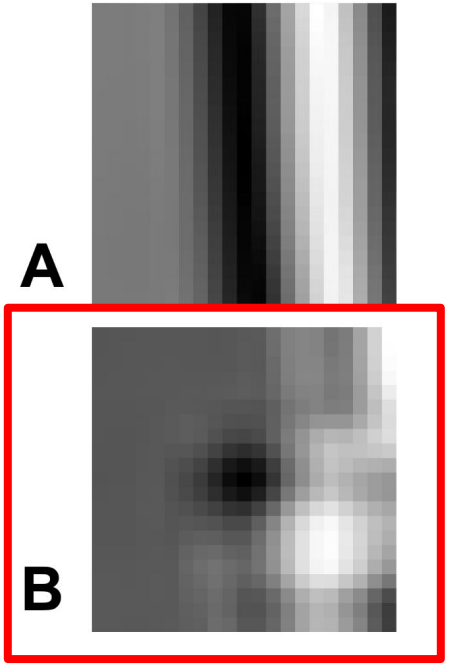


For the query patch in purple, what is the local shift error look like?

Original Image and Zoom-In

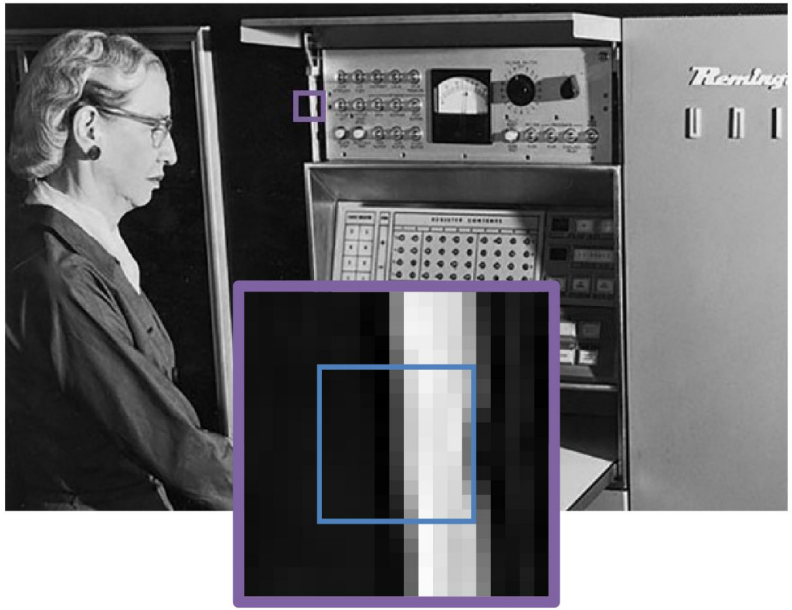


Error Options



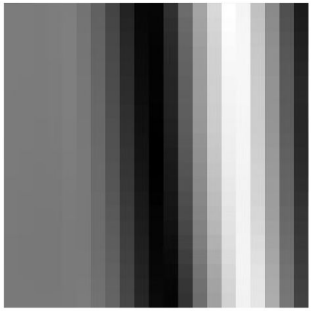
For the query patch in purple, what is the local shift error look like?

Original Image and Zoom-In

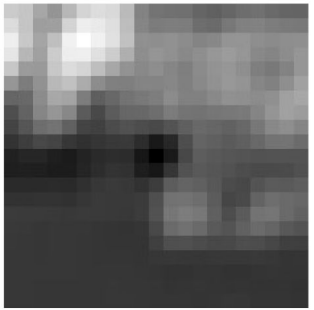


Error Options

A

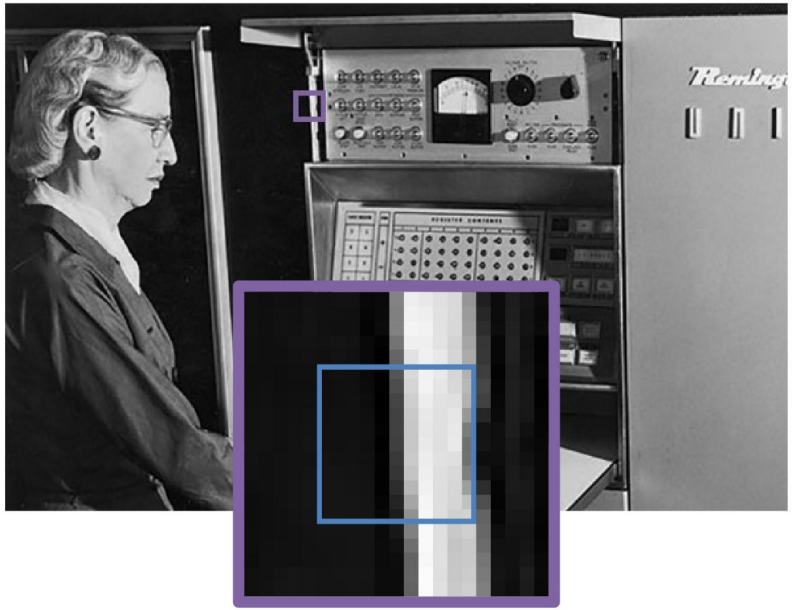


B

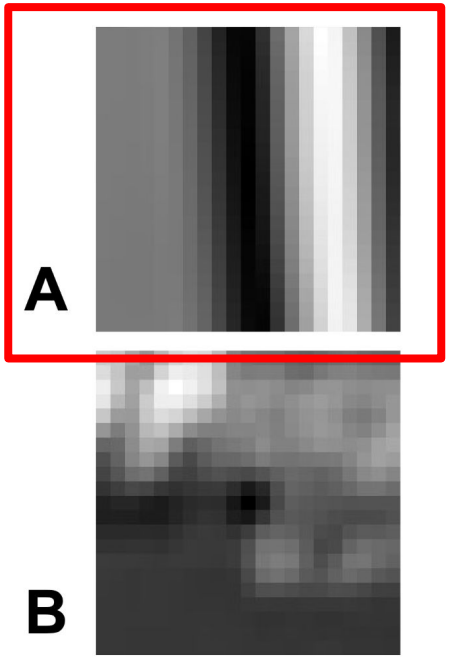


For the query patch in purple, what is the local shift error look like?

Original Image and Zoom-In

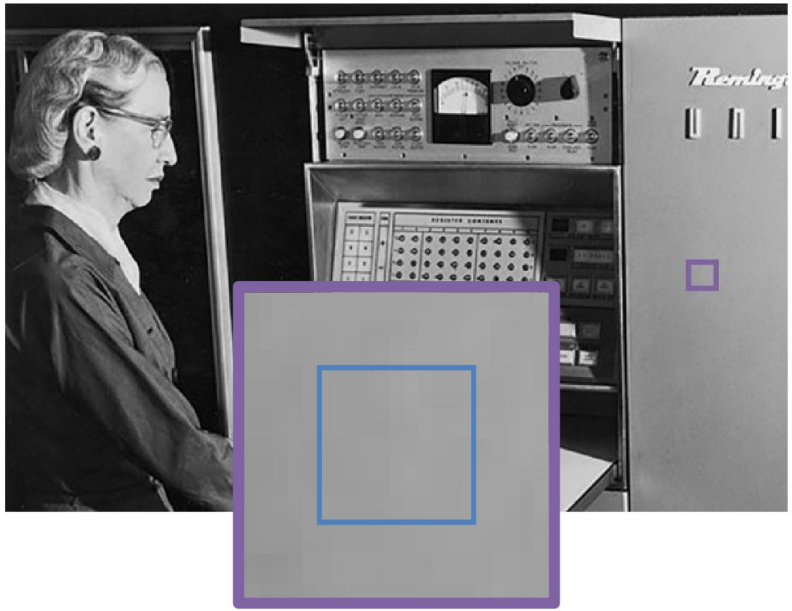


Error Options

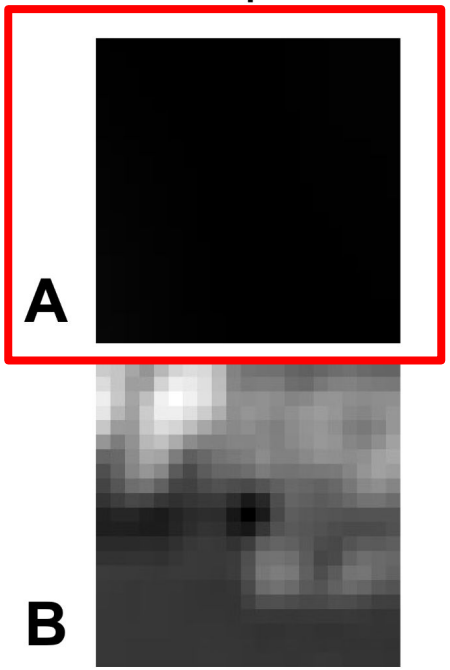


For the query patch in purple, what is the local shift error look like?

Original Image and Zoom-In



Error Options



Corners seem to have a distinct pattern that this filter yields - can we compute it efficiently?

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

$$\left(\text{  } - \text{  } \right)^2$$

Shifting windows around is expensive!
We'll find a trick to approximate this.

Note: only need to get the gist

Recall Taylor Series – way of *linearizing* a function:

$$f(x+d) \approx f(x) + \frac{\partial f}{\partial x} d$$

Do the same with images, treating them as function of x, y

$$I(x+u, y+v) \approx I(x,y) + I_x u + I_y v$$

For brevity: Ix = Ix at point (x,y), Iy = Iy at point (x,y)

Taylor series expansion for I at every single point in window

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

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

Cancel

$$= \sum_{(x,y)\in W} \left(I_x u + I_y v\right)^2$$

Expand

$$= \sum_{(x,y)\in W} I_x^2 u^2 + 2 I_x I_y uv + I_y^2 v^2$$

For brevity: Ix = Ix at point (x,y), Iy = Iy at point (x,y)

By linearizing image, we can approximate E(u,v) with quadratic function of u and v

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

$$= [u,v]\boldsymbol{M}[u,v]^T$$

$$\boldsymbol{M} = \begin{bmatrix} \sum_{x,y\in W} I_x^2 & \sum_{x,y\in W} I_x I_y \\ \sum_{x,y\in W} I_x I_y & \sum_{x,y\in W} I_y^2 \end{bmatrix}$$

M only depends on the underlying image gradients

U,v only depends on the neighborhood.

M is called the second moment matrix

Pretend gradients are *either* vertical or horizontal at a pixel (so Ix Iy = 0)

$$\boldsymbol{M} = \begin{bmatrix} \sum_{x,y \in W} I_x^2 & \sum_{x,y \in W} I_x I_y \\ \sum_{x,y \in W} I_x I_y & \sum_{x,y \in W} I_y^2 \end{bmatrix} \approx \begin{bmatrix} a & 0 \\ 0 & b \end{bmatrix}$$

a,b both small: flat  $\begin{bmatrix} 0.1 & 0 \\ 0 & 0.1 \end{bmatrix}$

One big, other small: edge  $\begin{bmatrix} 50 & 0 \\ 0 & 0.1 \end{bmatrix}$ or $\begin{bmatrix} 0.1 & 0 \\ 0 & 50 \end{bmatrix}$

a,b both big: corner  $\begin{bmatrix} 50 & 0 \\ 0 & 50 \end{bmatrix}$

Pretend gradients are *either* vertical or horizontal at a pixel (so Ix Iy = 0)

$$\boldsymbol{M} = \begin{bmatrix} \sum_{x,y \in W} I_x^2 & \sum_{x,y \in W} I_x I_y \\ \sum_{x,y \in W} I_x I_y & \sum_{x,y \in W} I_y^2 \end{bmatrix} \approx ? \begin{bmatrix} a & 0 \\ 0 & b \end{bmatrix}$$

a,b both small: flat

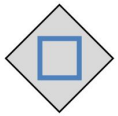


Image might be rotated by rotation θ!

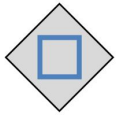
One big, other small: edge

a,b both big: corner

Pretend gradients are *either* vertical or horizontal at a pixel (so Ix Iy = 0)

$$\boldsymbol{M} = \begin{bmatrix} \sum_{x,y \in W} I_x^2 & \sum_{x,y \in W} I_x I_y \\ \sum_{x,y \in W} I_x I_y & \sum_{x,y \in W} I_y^2 \end{bmatrix} = \boldsymbol{V^{-1}} \begin{bmatrix} a & 0 \\ 0 & b \end{bmatrix} \boldsymbol{V}$$

a,b both small: flat



If image rotated by rotation θ / matrix **V**

One big, other small: edge

M will look like

$$\boldsymbol{V^{-1}} \begin{bmatrix} a & 0 \\ 0 & b \end{bmatrix} \boldsymbol{V}$$

a,b both big: corner

Can calculate M at pixel, by summing nearby gradients, but need access to a and b.

$$\boldsymbol{M} = \begin{bmatrix} \sum_{x,y \in W} I_x^2 & \sum_{x,y \in W} I_x I_y \\ \sum_{x,y \in W} I_x I_y & \sum_{x,y \in W} I_y^2 \end{bmatrix} = \boldsymbol{V^{-1}} \begin{bmatrix} a & 0 \\ 0 & b \end{bmatrix} \boldsymbol{V}$$

Given **M**, can decompose it into eigenvectors **V** and eigenvalues λ_1, λ_2 with $\mathbf{M} = \boldsymbol{V^{-1}} \begin{bmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{bmatrix} \boldsymbol{V}$.

Really slow. Why?

Can calculate M at pixel, by summing nearby gradients, but need access to a and b.

$$\boldsymbol{M} = \begin{bmatrix} \sum_{x,y \in W} I_x^2 & \sum_{x,y \in W} I_x I_y \\ \sum_{x,y \in W} I_x I_y & \sum_{x,y \in W} I_y^2 \end{bmatrix} = \boldsymbol{V^{-1}} \begin{bmatrix} a & 0 \\ 0 & b \end{bmatrix} \boldsymbol{V}$$

Instead: compute quantity R from **M**

$$R = \det(\boldsymbol{M}) - \alpha \operatorname{trace}(\boldsymbol{M})^2 = \lambda_1 \lambda_2 - \alpha (\lambda_1 + \lambda_2)^2$$

Easy fast formula for 2x2

Fast – sum the diagonal

Empirical value, usually 0.04-0.06

Harris corner detector (1988)

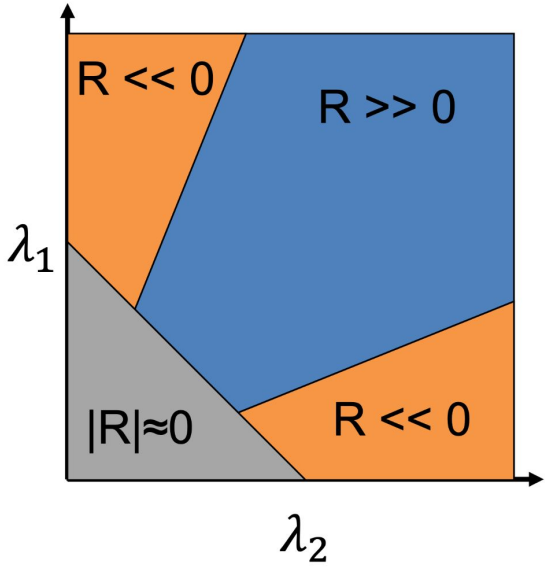
R tells us whether we're at a corner, edge, or flat

$$R = \det(\boldsymbol{M}) - \alpha \operatorname{trace}(\boldsymbol{M})^2 = \lambda_1\lambda_2 - \alpha\,(\lambda_1 + \lambda_2)^2$$

flat $\lambda_1, \lambda_2 \approx 0$

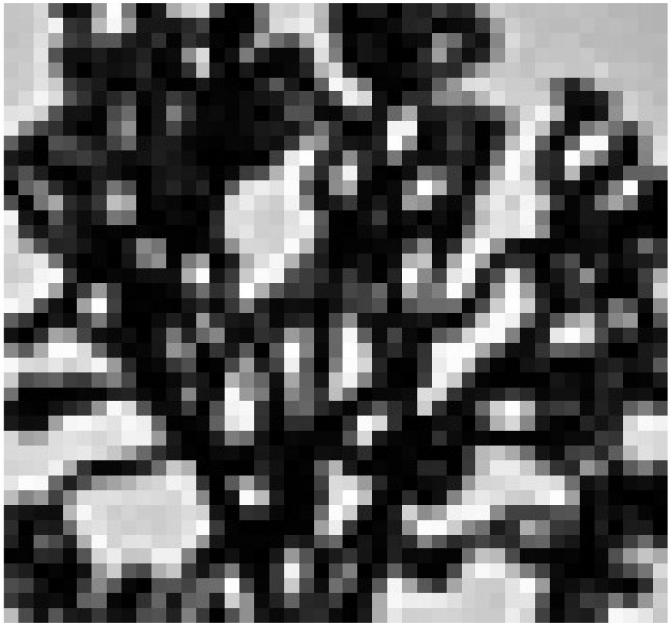
edge $\lambda_1 \gg \lambda_2 \gg 0$ $\lambda_2 \gg \lambda_1 \gg 0$

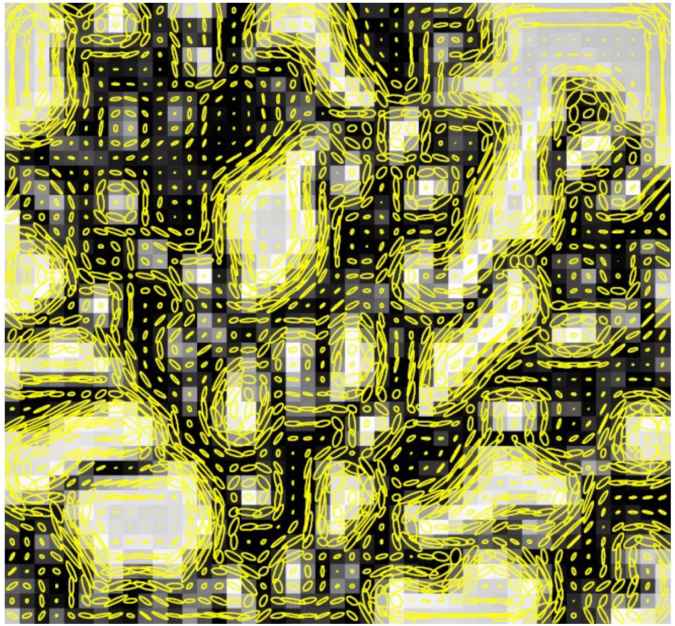
corner $\lambda_1 \approx \lambda_2 \gg 0$

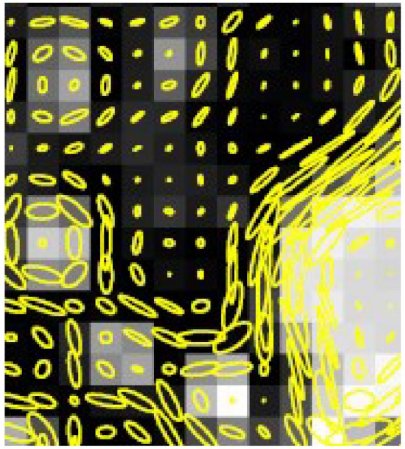


Remake of standard diagram from S. Lazebnik from original Harris paper.

Intuition on u,v and M

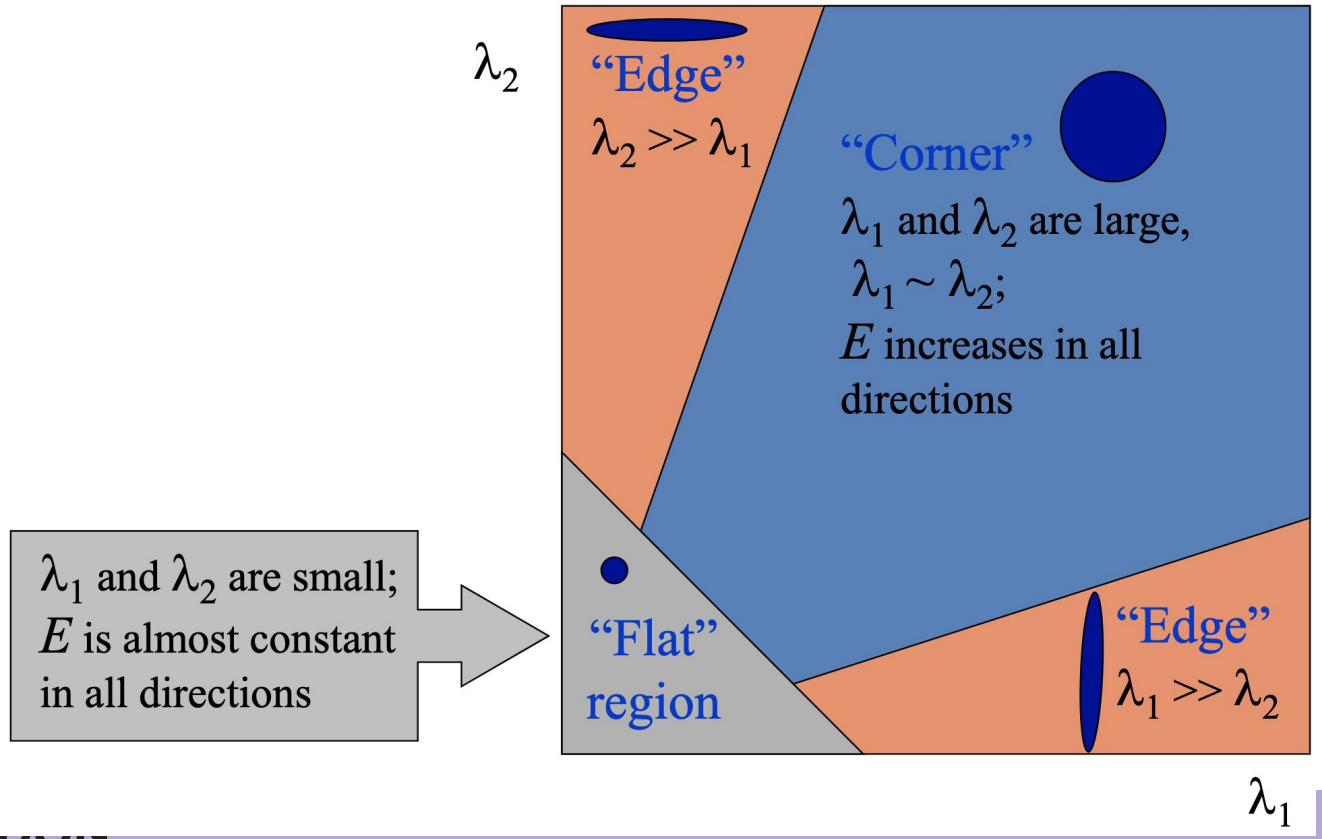






Technical note: M is often best *visualized* by first taking inverse, so long edge of ellipse goes along edge

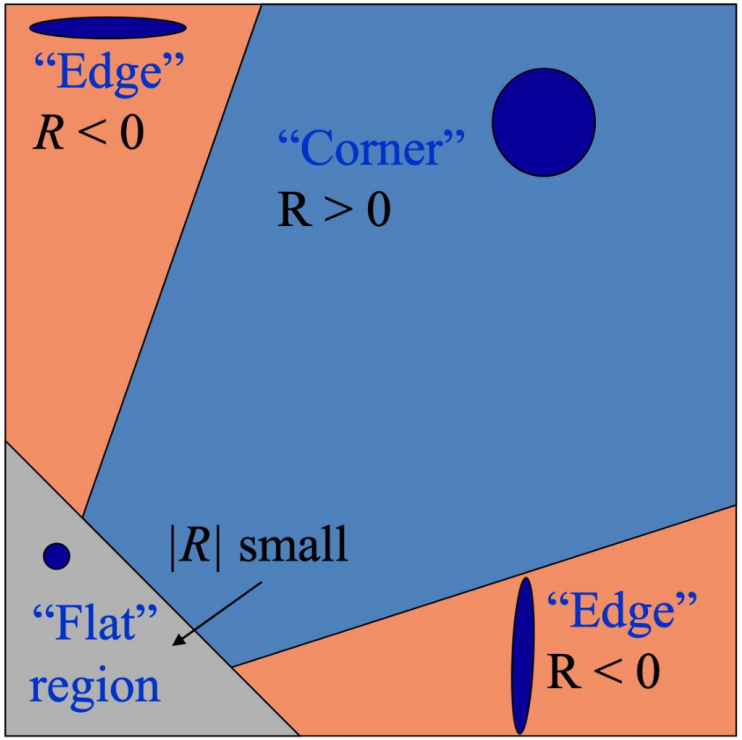
Intuition on the eigenvalues of M



Intuition on the Harris metric R

$$R = \det(\boldsymbol{M}) - \alpha\, trace(\boldsymbol{M})^2$$
$$= \lambda_1\lambda_2 - \alpha(\lambda_1 + \lambda_2)^2$$

α: constant (0.04 to 0.06)



Takeaways for Harris corner detector

- Need to be able to take derivatives of image
- Need to be able to compute the entries of **M** at every pixel.
- Should know that some properties of **M** indicate whether a pixel is a corner or not.

$$\boldsymbol{M} = \begin{bmatrix} \sum_{x,y \in W} I_x^2 & \sum_{x,y \in W} I_x I_y \\ \sum_{x,y \in W} I_x I_y & \sum_{x,y \in W} I_y^2 \end{bmatrix}$$

Harris corner detector in practice

1. Compute partial derivatives Ix, Iy per pixel
2. Compute **M** at each pixel, using Gaussian weighting w

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

C.Harris and M.Stephens. "A Combined Corner and Edge Detector." *Proceedings of the 4th Alvey Vision Conference*: pages 147—151, 1988.

Harris corner detector in practice

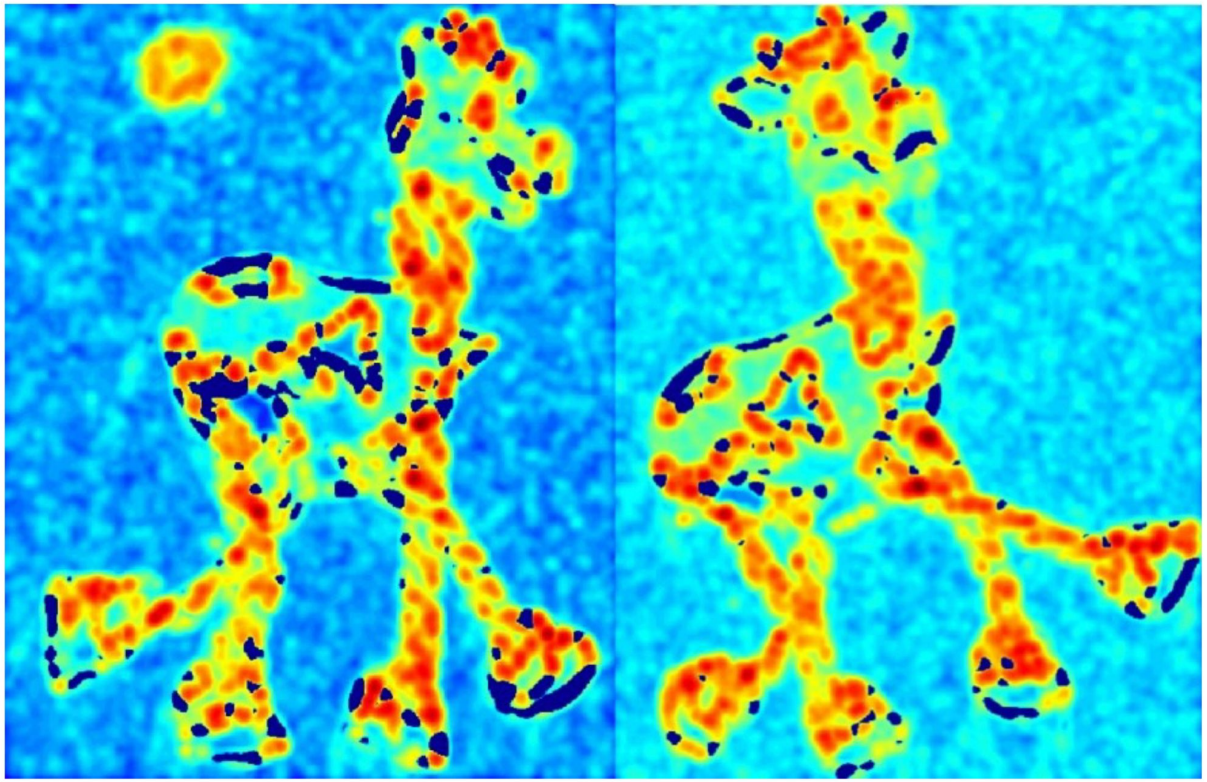
1. Compute partial derivatives Ix, Iy per pixel
2. Compute **M** at each pixel, using Gaussian weighting w
3. Compute response function R

$$R = \det(\boldsymbol{M}) - \alpha\, trace(\boldsymbol{M})^2$$
$$= \lambda_1\lambda_2 - \alpha(\lambda_1 + \lambda_2)^2$$

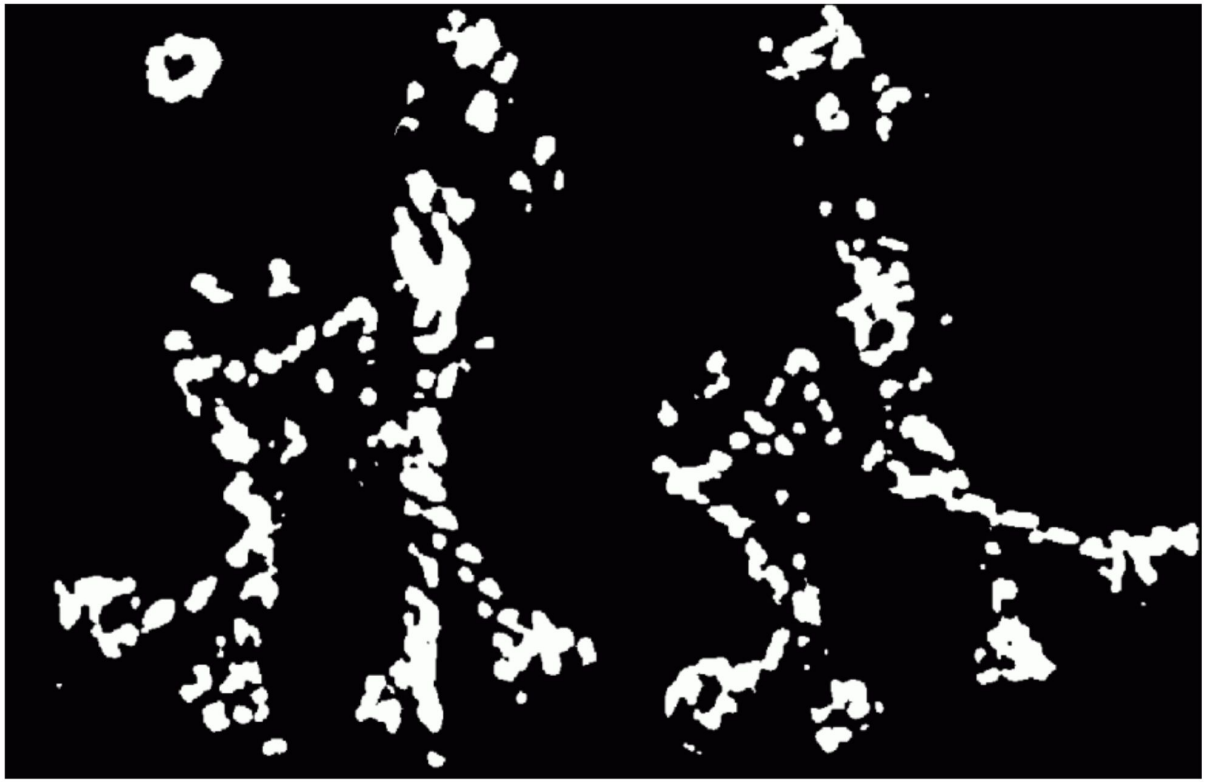
C.Harris and M.Stephens. "A Combined Corner and Edge Detector."
Proceedings of the 4th Alvey Vision Conference: pages 147—151, 1988.

Harris corner detector steps

Harris corner detector steps - Compute R value from Ix,Iy



Harris corner detector steps - R thresholded



Harris corner detector steps - R local maxima

Harris detector in code: Simple convolution operations and pointwise multiplications

```
def corner_response(image, k, Sx=Sx, Sy=Sy, G=G):
    # compute first derivatives
    dx = cv2.filter2D(image, ddepth=-1, kernel=Sx)
    dy = cv2.filter2D(image, ddepth=-1, kernel=Sy)

    # Gaussian Filter (blur)
    A = cv2.filter2D(dx*dx, ddepth=-1, kernel=G)
    B = cv2.filter2D(dy*dy, ddepth=-1, kernel=G)
    C = cv2.filter2D(dx*dy, ddepth=-1, kernel=G)

    # compute corner response at all pixels
    return (A*B - (C*C)) - k*(A + B)*(A + B)

# hyperparameters
k = 0.05
thresh = 0.5

# thresholded corner responses
strong_corners = corner_response(image, k) > thresh
```

https://medium.com/@itberrios6/harris-corner-and-edge-detector-4169312aa2f8

Recall what we want out of image feature detectors

- Repeatable: should find same things even with distortion
- Saliency: each feature should be distinctive
- Compactness: shouldn’t just be all the pixels
- Locality: should only depend on local image data

Harris detector properties

- **Invariant** to some things: image is transformed and corners remain the same
- **Covariant/equivariant** with some things: image is transformed and corners transform with it.

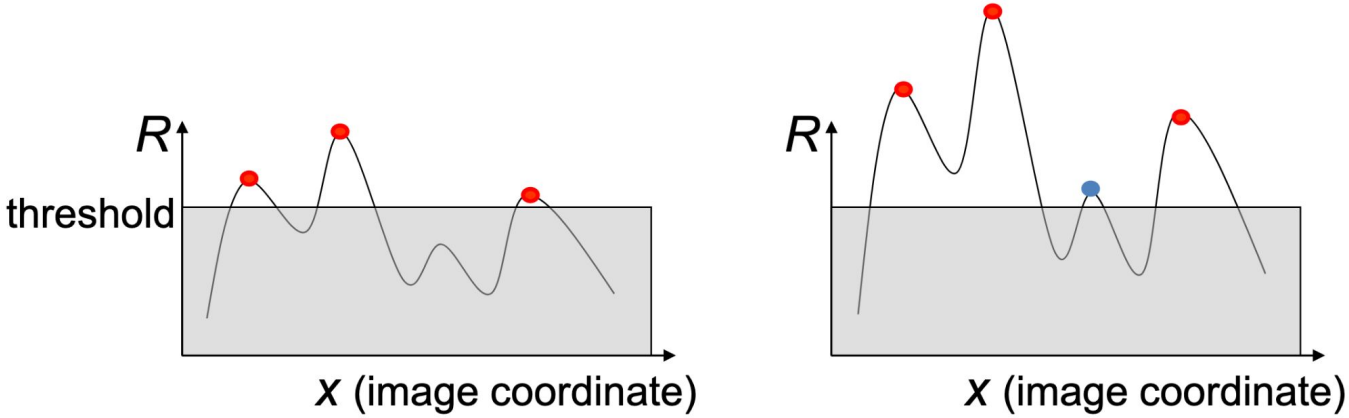
How about under different lighting and geometry of image pairs

Affine Intensity Change

$$I_{new} = aI_{old} + b$$

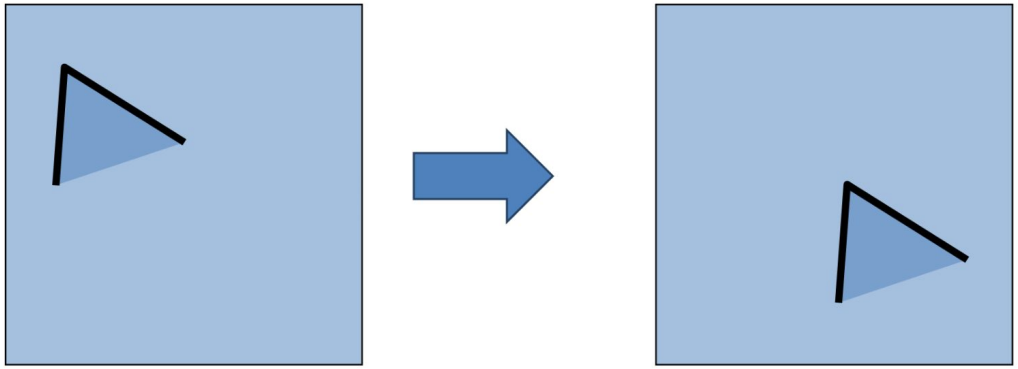
M only depends on derivatives, so *b* is irrelevant

But *a* scales derivatives and there's a threshold



Partially invariant to affine intensity changes

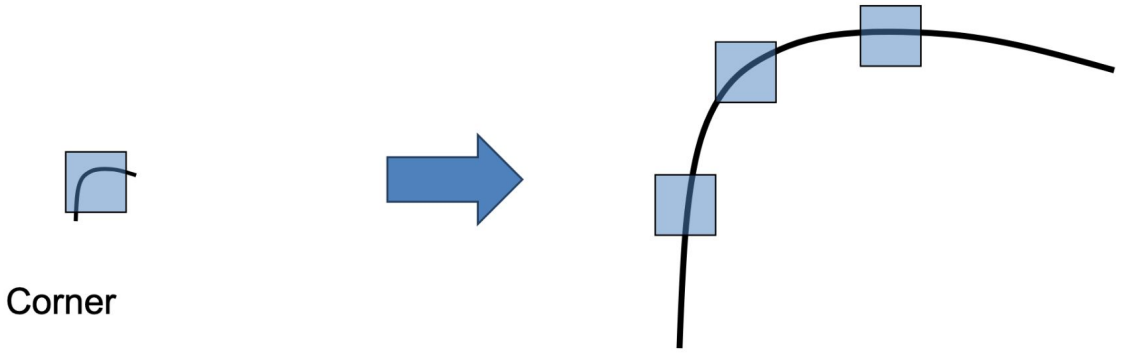
Image Translation



All done with convolution. Convolution is translation equivariant.

Equivariant with translation

Image Scaling



One pixel can become many pixels and vice-versa.

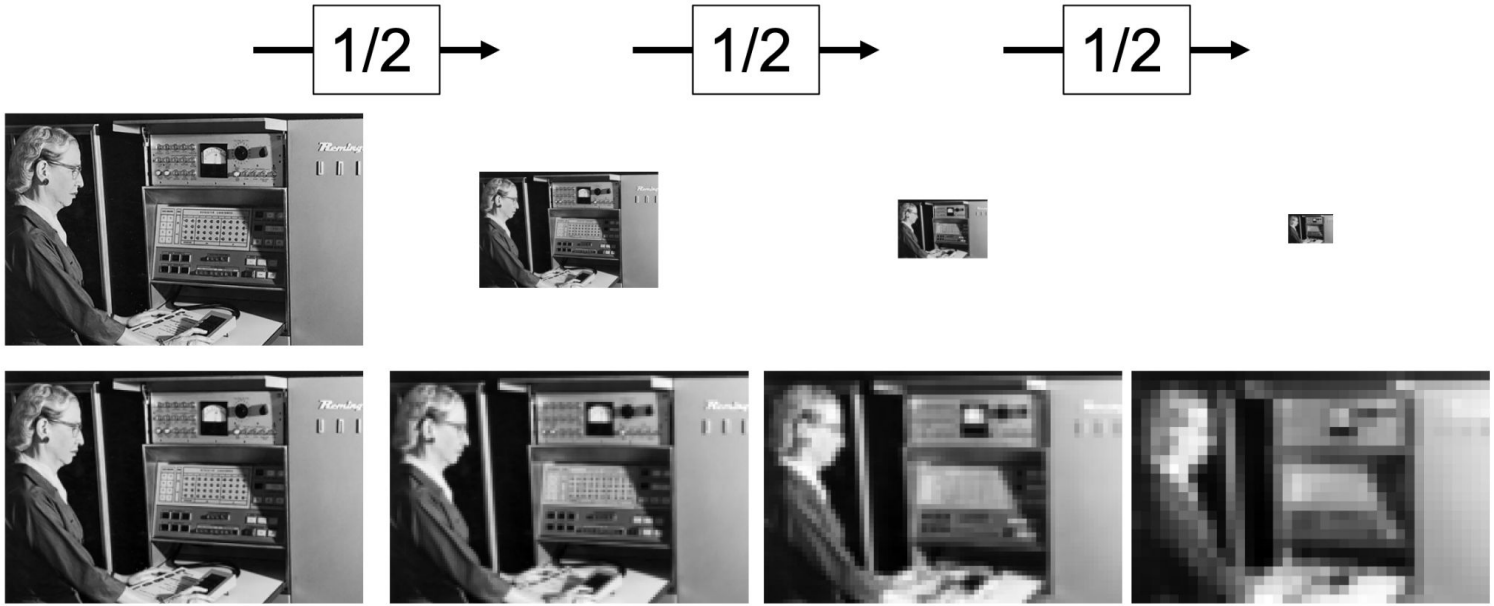
Not equivariant with scaling

How do we fix this?

Scale invariant feature detectors

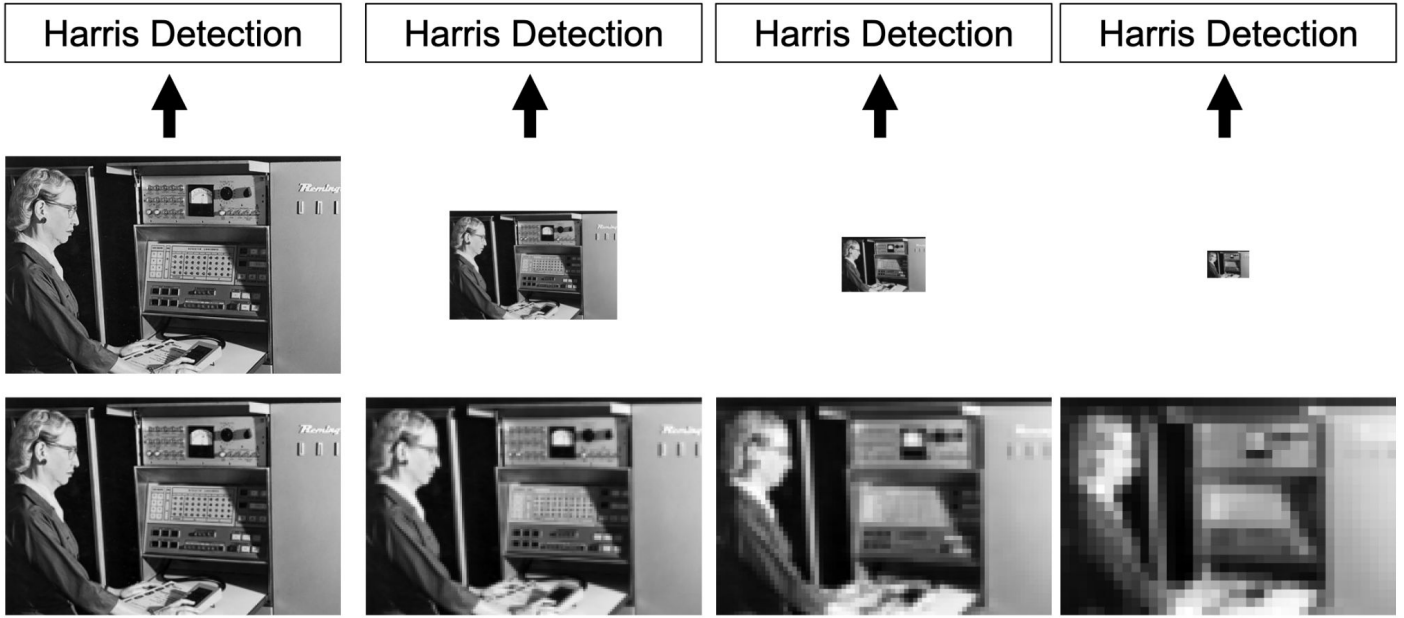
Key Idea: Scale Space

Left to right: each image is half-sized
Upsampled with big pixels below



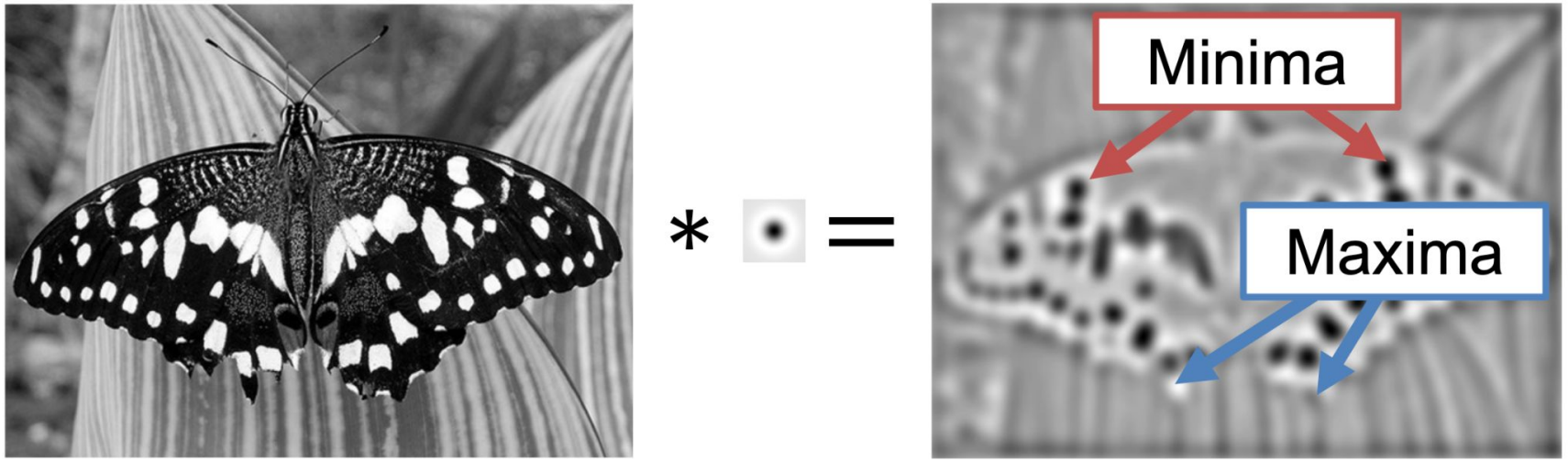
Solution to Scales

Try them all!



Blob Detection

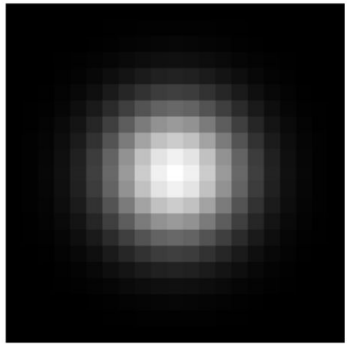
Another detector (has some nice properties)



Find maxima ***and minima*** of blob filter response in scale ***and space***

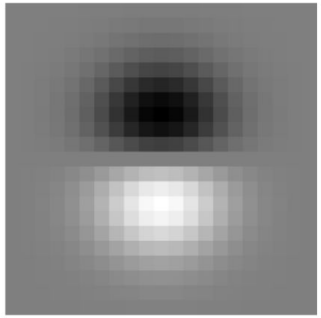
Gaussian Derivatives

Gaussian

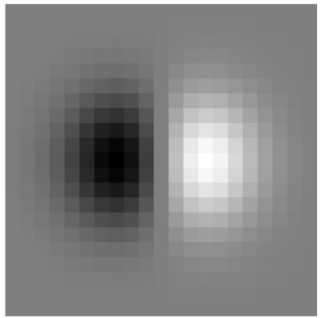


1st Deriv

$$\frac{\partial}{\partial y} g$$

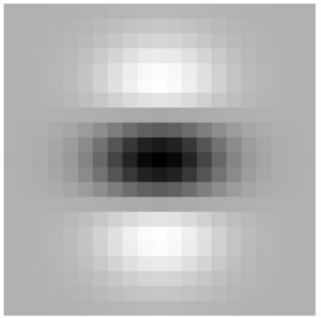


$$\frac{\partial}{\partial x} g$$

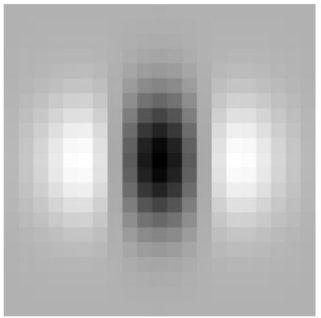


2nd Deriv

$$\frac{\partial^2}{\partial^2 y} g$$

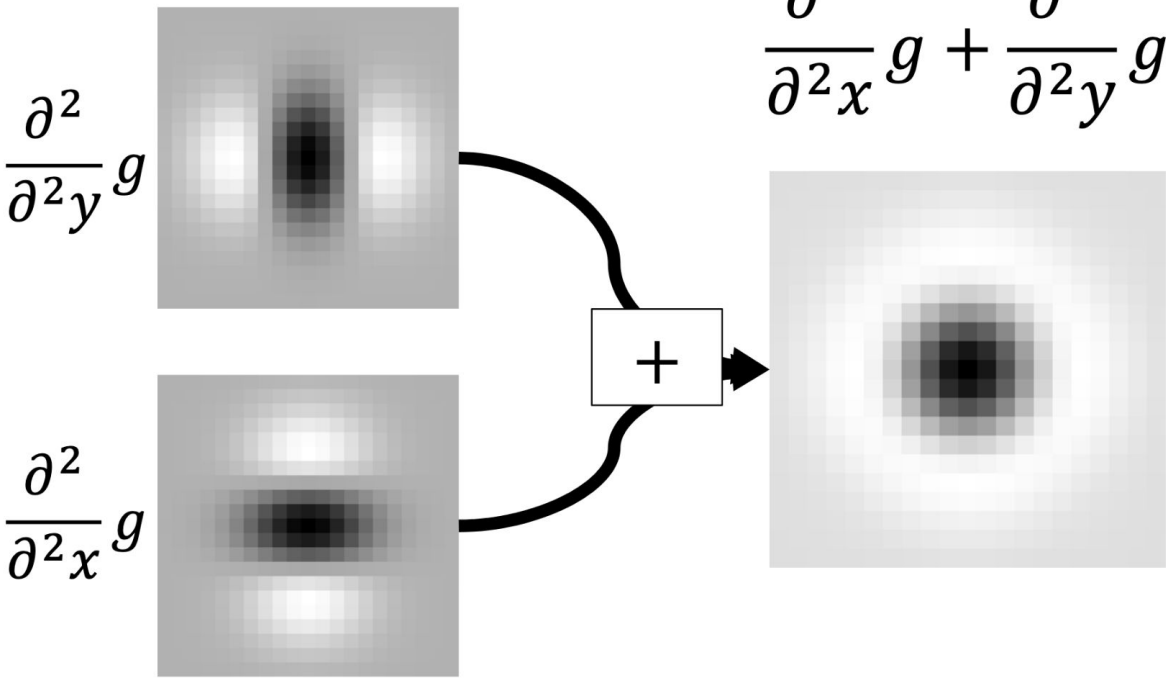


$$\frac{\partial^2}{\partial^2 x} g$$



Laplacian of Gaussian (LoG)

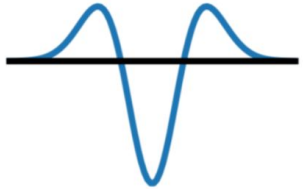
$$\frac{\partial^2}{\partial^2 x} g + \frac{\partial^2}{\partial^2 y} g$$



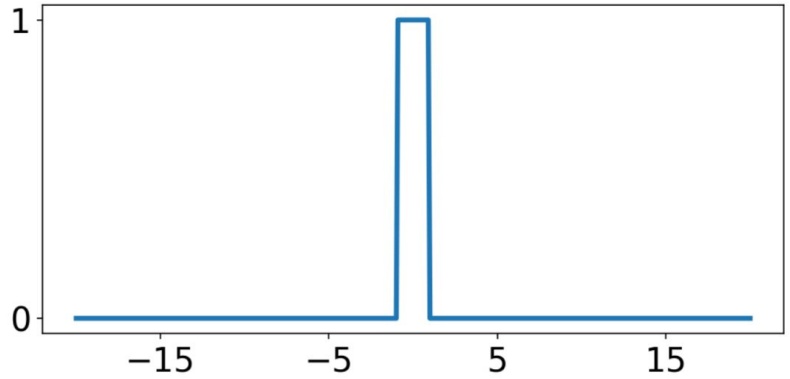
Slight detail: for technical reasons, you need to scale the Laplacian of Gaussian if you want to compare across sigmas.

$$\nabla^2_{norm} = \sigma^2 \left(\frac{\partial^2}{\partial x^2} g + \frac{\partial^2}{\partial^2 y} g \right)$$

Edge Detection with LoG

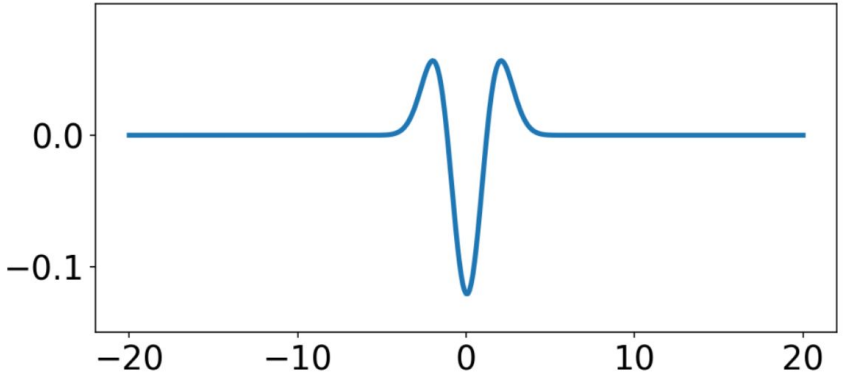


f



Edge

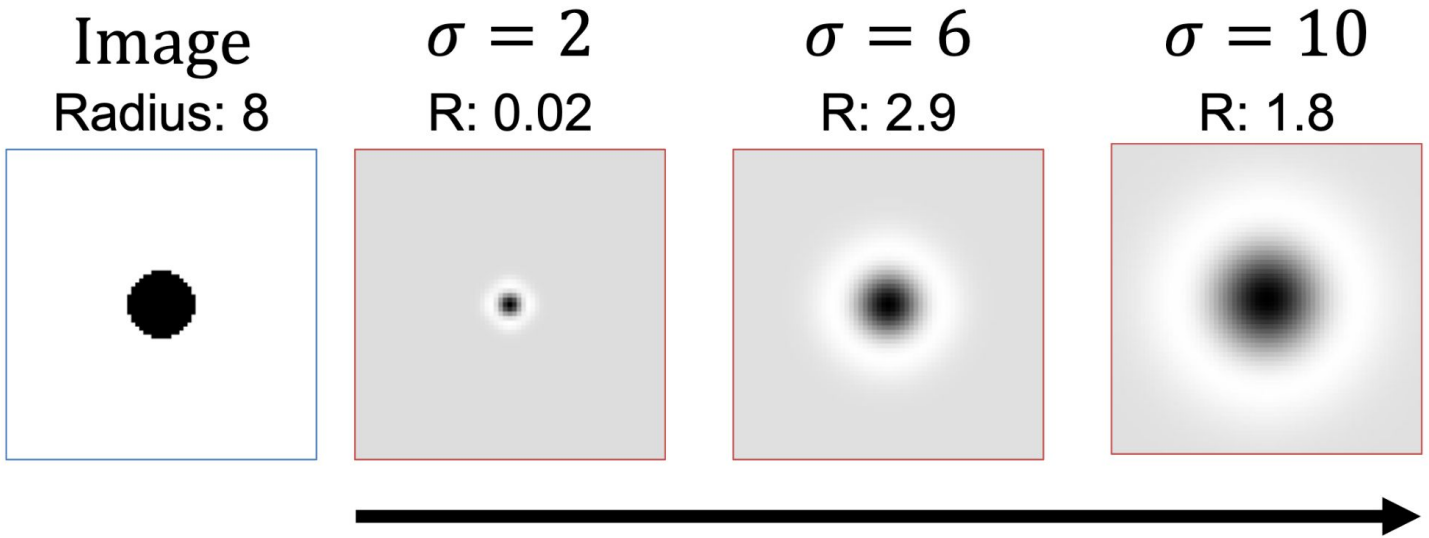
$f * \frac{\partial^2}{\partial^2 x} g$



Edge *
LoG =
Zero-crossing

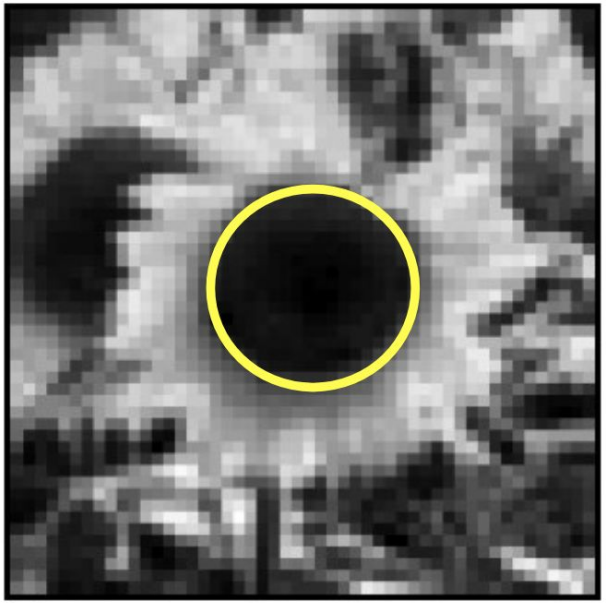
Laplacian can also be used to measure blob size - laplacian that matches the size of the blob maximizes the response

Given binary circle and Laplacian filter of scale σ, we can compute the response as a function of the scale.

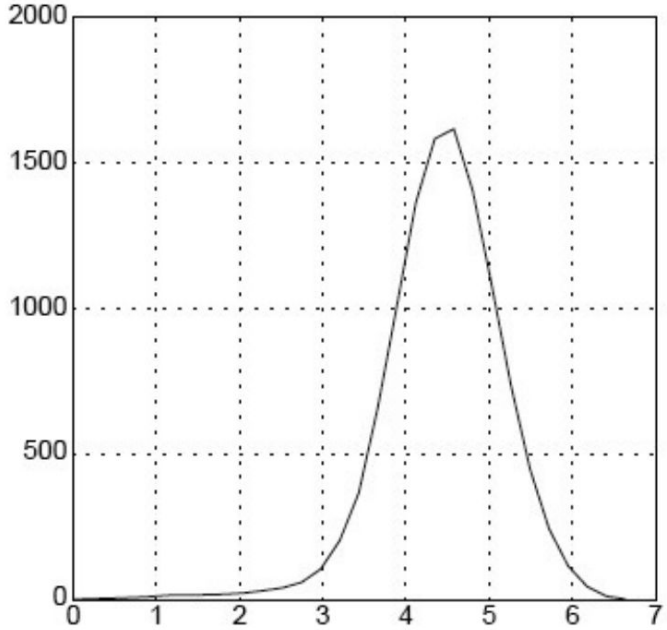


Characteristic scale of a blob is the scale that produces the maximum response

Image



Abs. Response



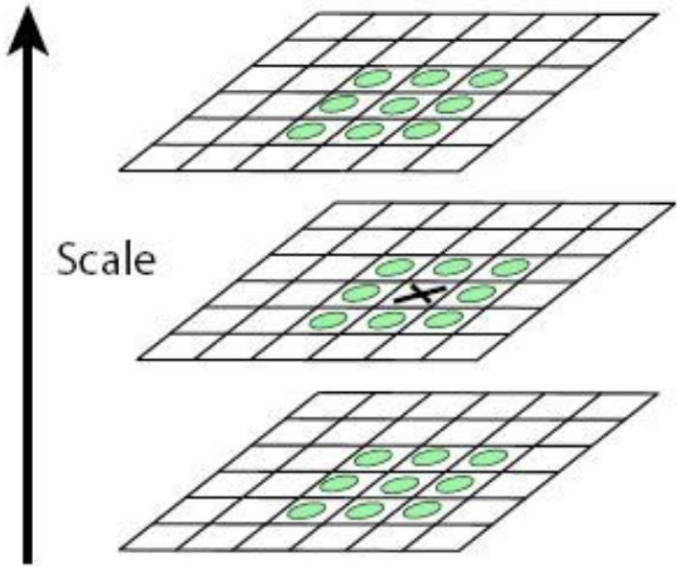
Scale-space blob detector: Example

Convolve image with scale-normalized Laplacian at several scales

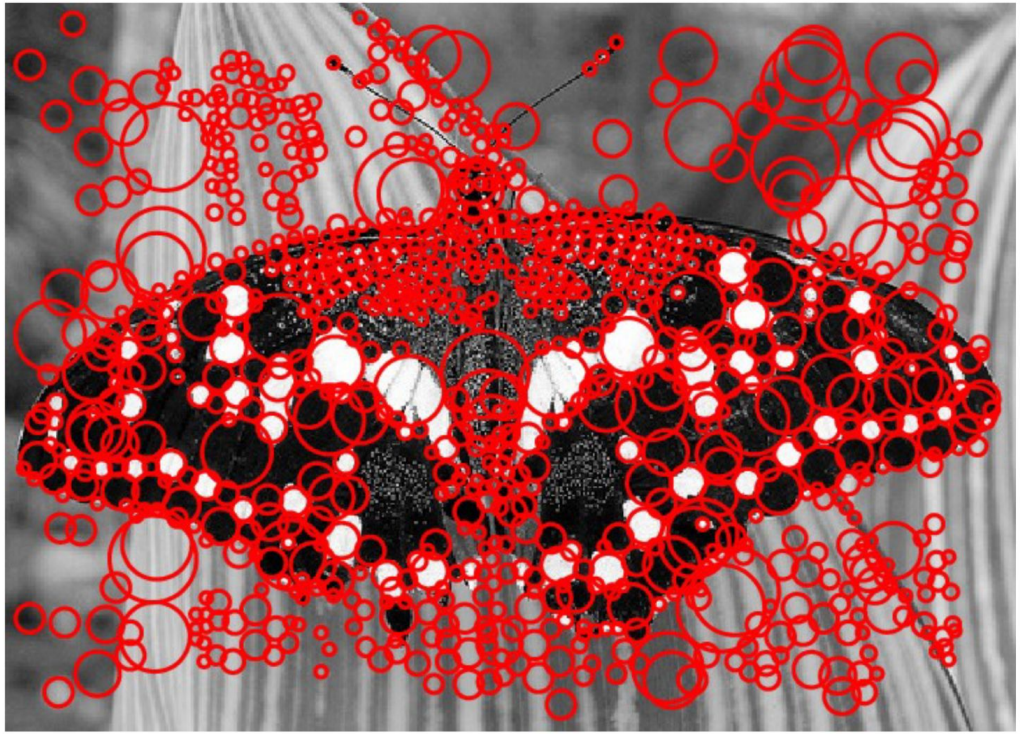
Find local maxima across pixels and scales

Suppose I[:,:,k] is image at scale k. Point i,j,k is maxima (minima if you flip sign) in image I if:

```
for y=range(i-1,i+1+1):
        for x in range(j-1,j+1+1):
                for c in range(k-1,k+1+1):
                        if y == i and x== j and c==k:
                                continue
                        #below has to be true
                        I[y,x,c] < I[i,j,k]
```



Threshold the local maxima and display the surviving pixels with their scale of maximum response



We have scale invariant feature detectors - how do we describe them? SIFT

We want scale, rotation and illumination invariant feature descriptions

Image – 40

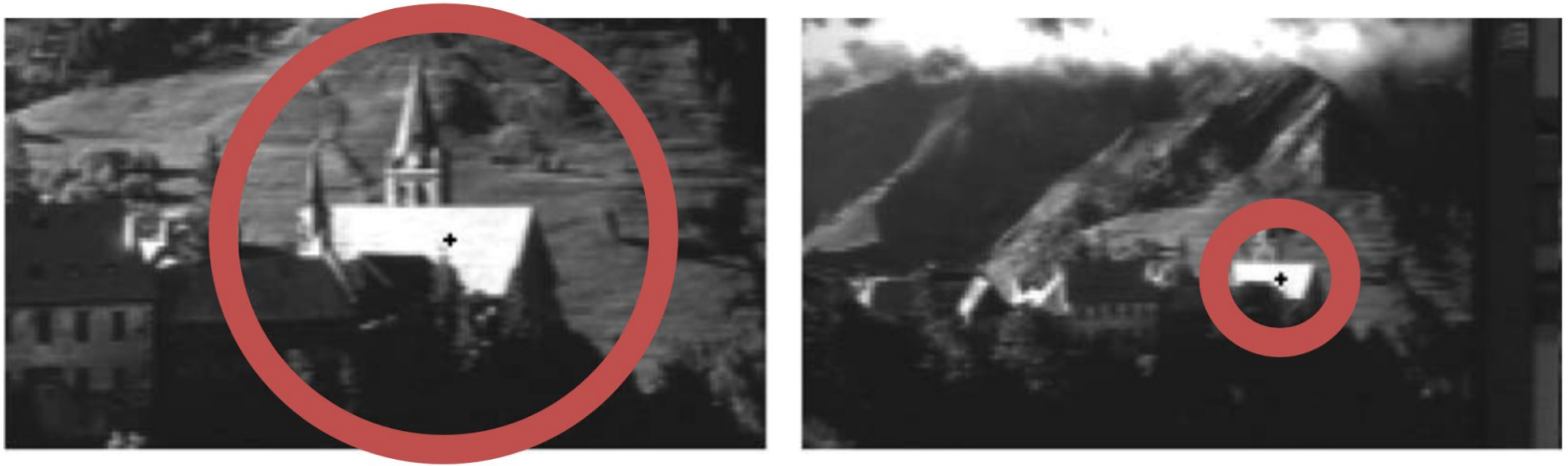
1/2 size, rot. 45
Lightened+40

Full
Image

100x100 crop
at Glasses

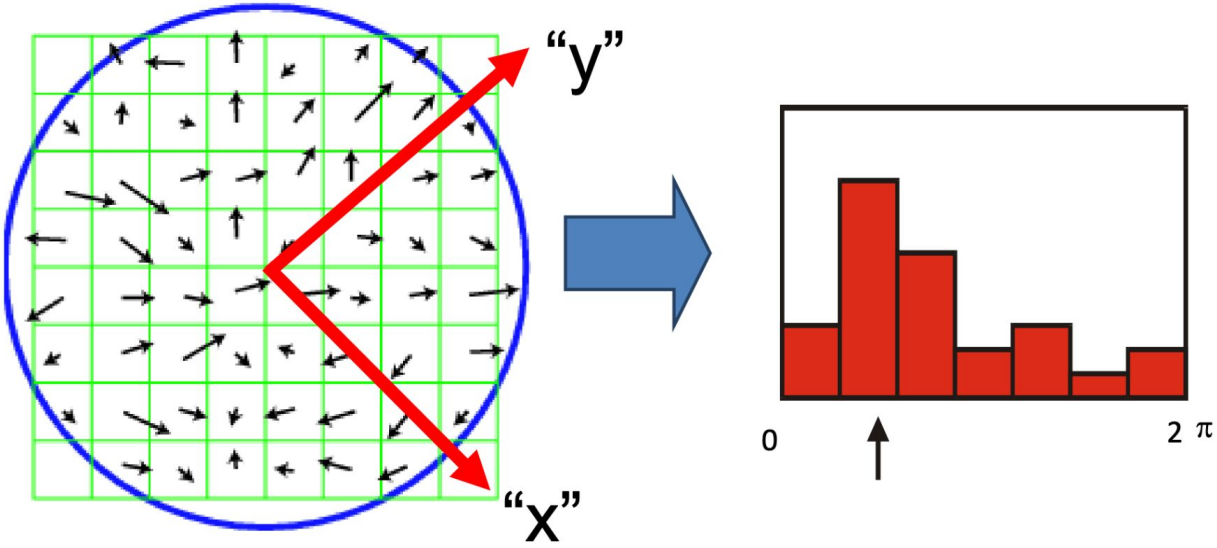
Handling Scale

Given characteristic scale (maximum Laplacian response), we can just rescale image

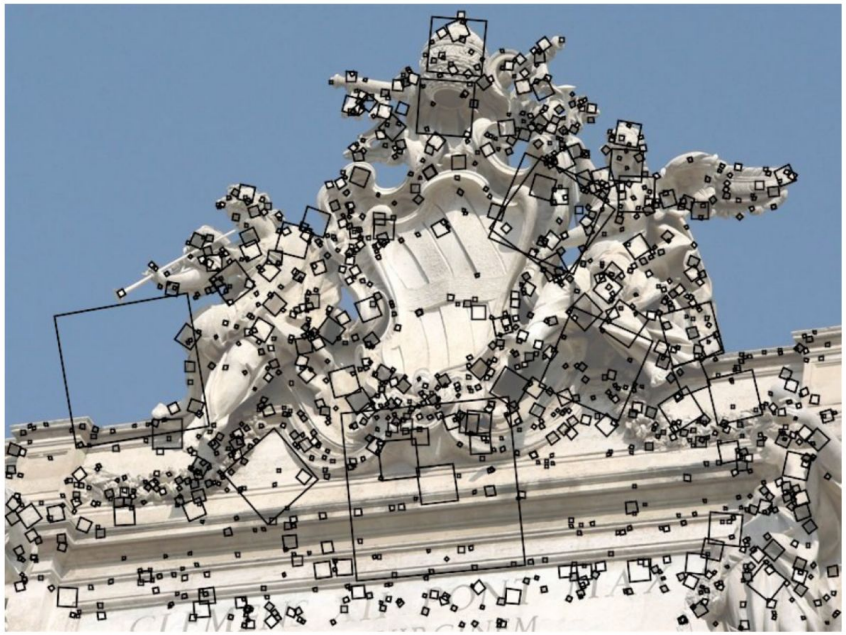


Handling Rotation

Given window, can compute “dominant orientation”
and then rotate image

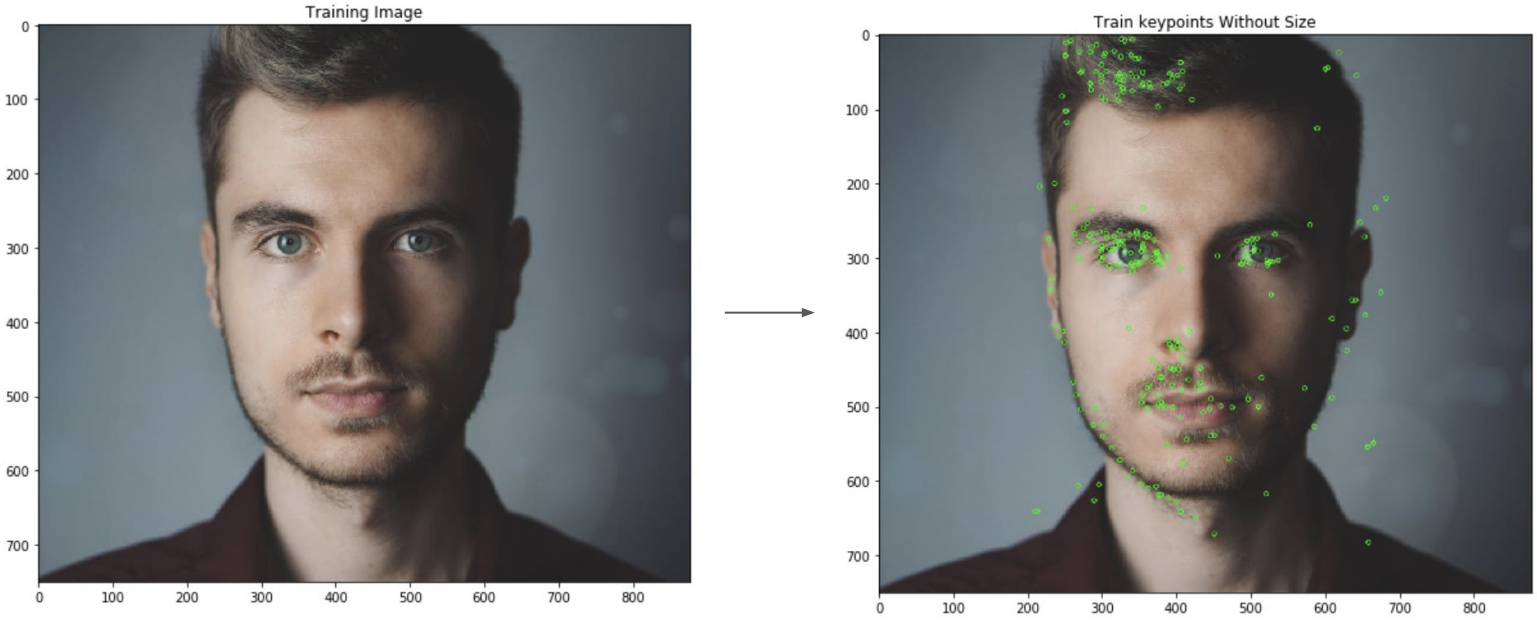


SIFT features at characteristic scales and dominant orientations



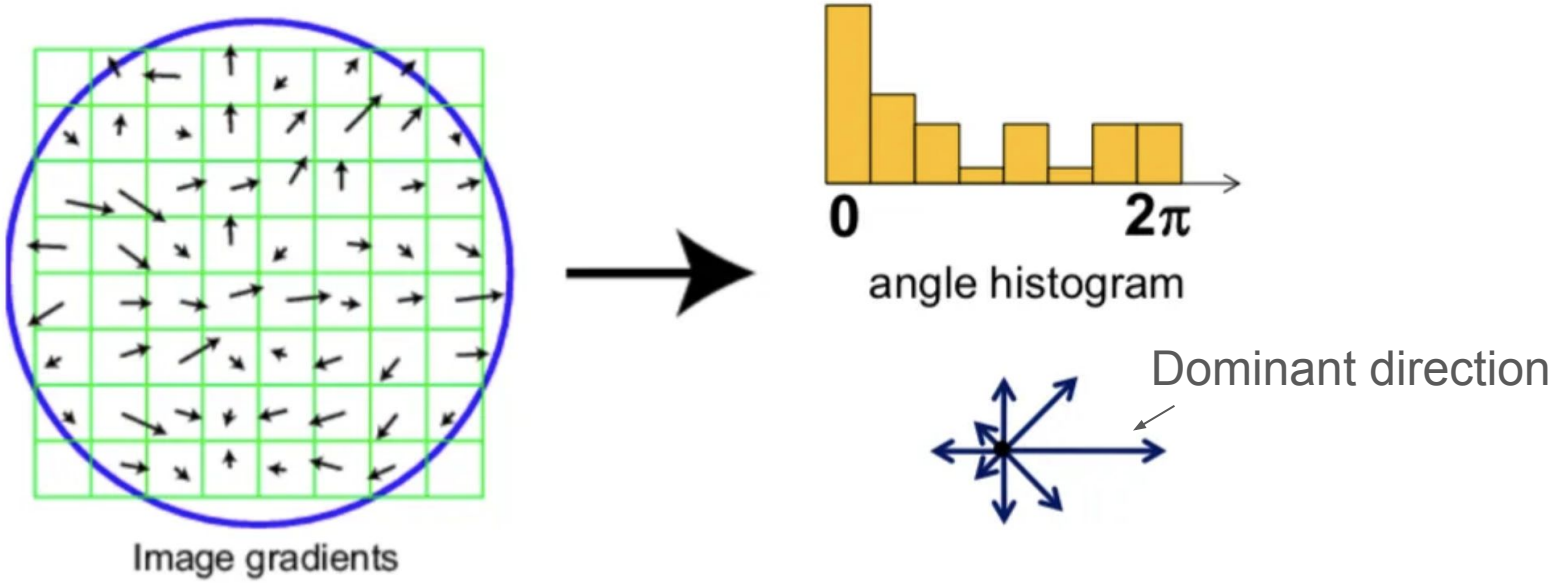
David G. Lowe. "Distinctive image features from scale-invariant keypoints." IJCV 60 (2), pp. 91-110, 2004.

SIFT: step one - grab keypoints using scale invariant blob detection and harris corner detection



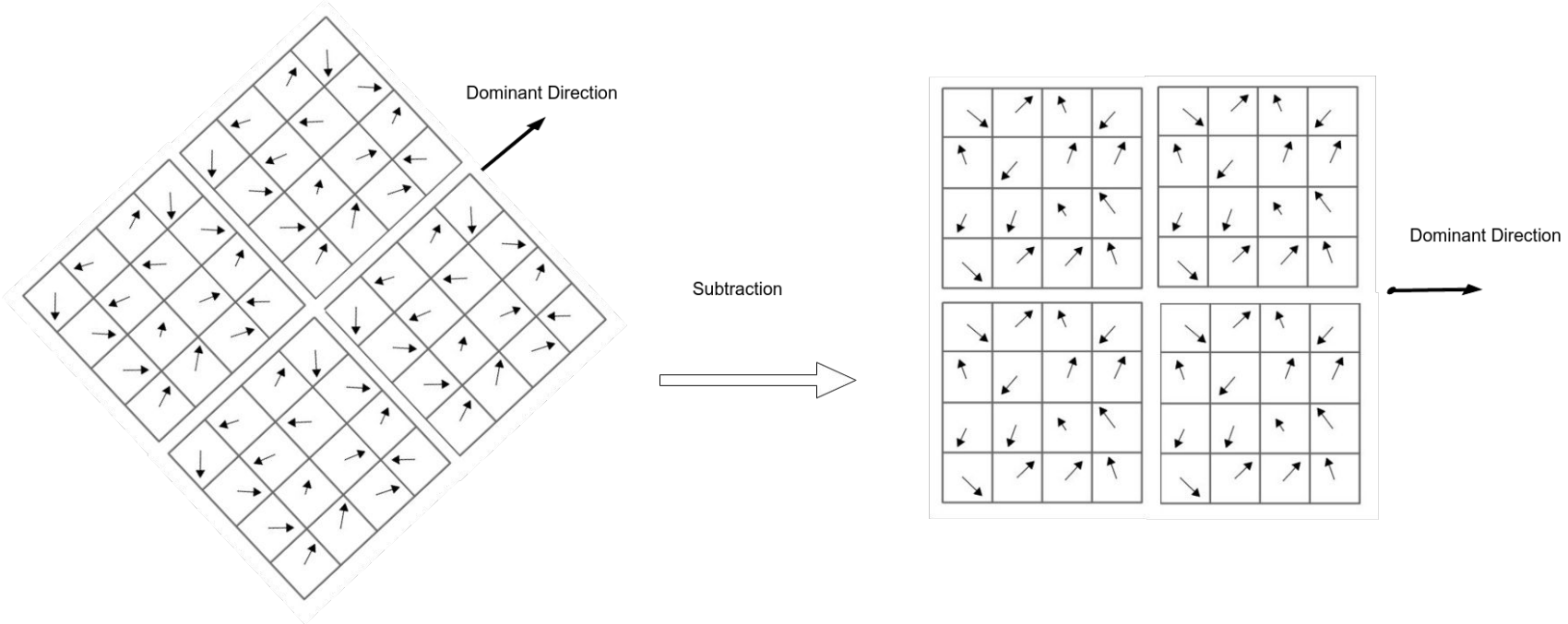
Each keypoint now has a location + scale associated with it

SIFT: step two, compute “dominant” orientation of each keypoint using local gradients.

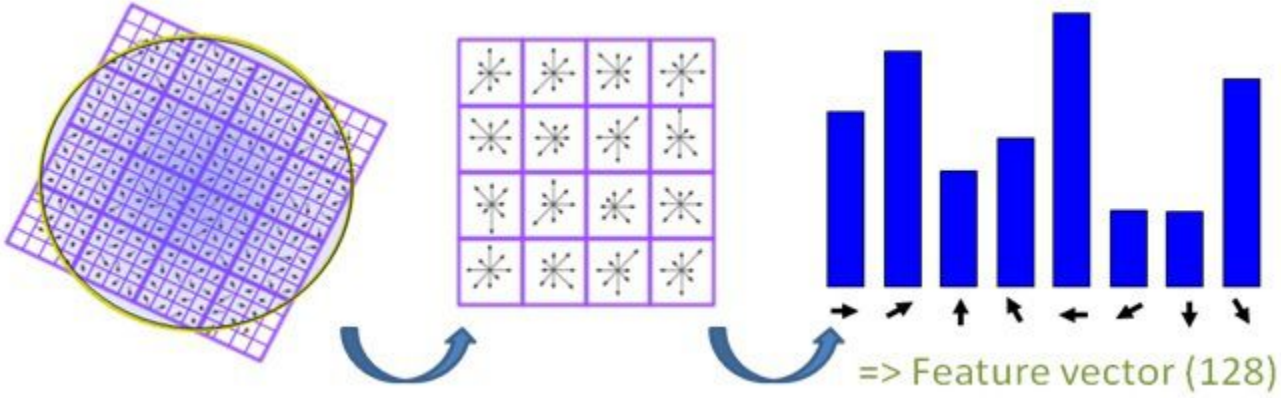


Each keypoint now has a location + scale + orientation associated with it

SIFT: step three, compute neighboring orientations subtracted by the dominant direction of the keypoint

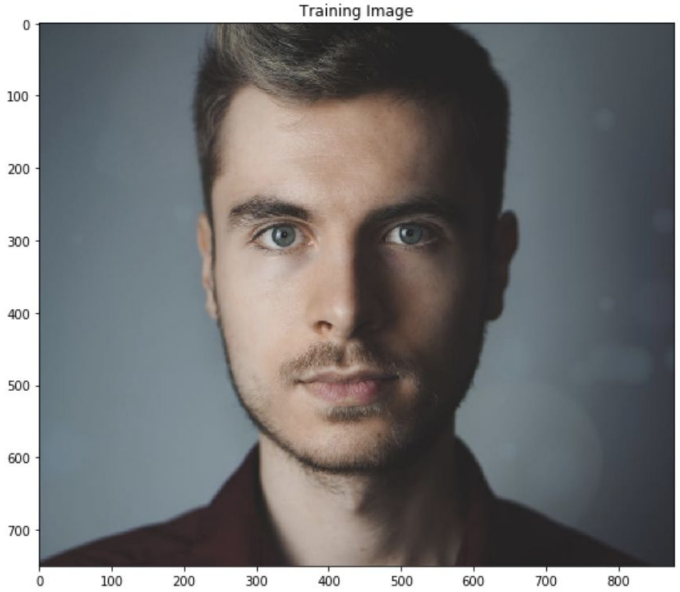


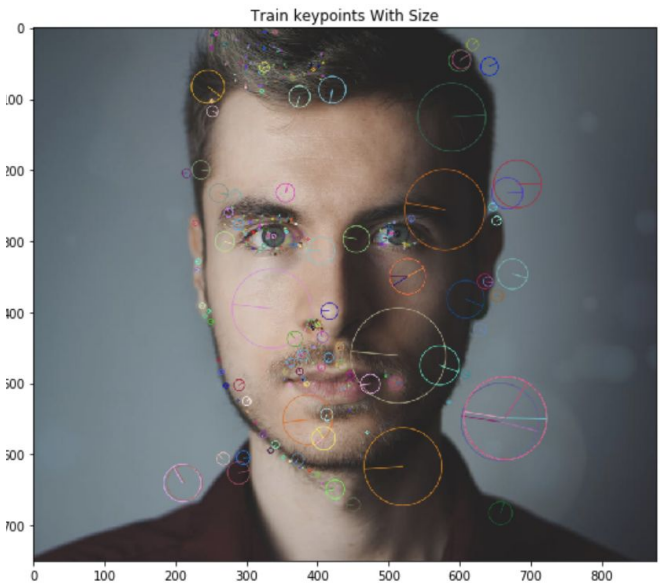
SIFT: step four, compute rotation subtracted directional histograms (8bins) for all the N x N (usually 4 x 4) neighborhoods



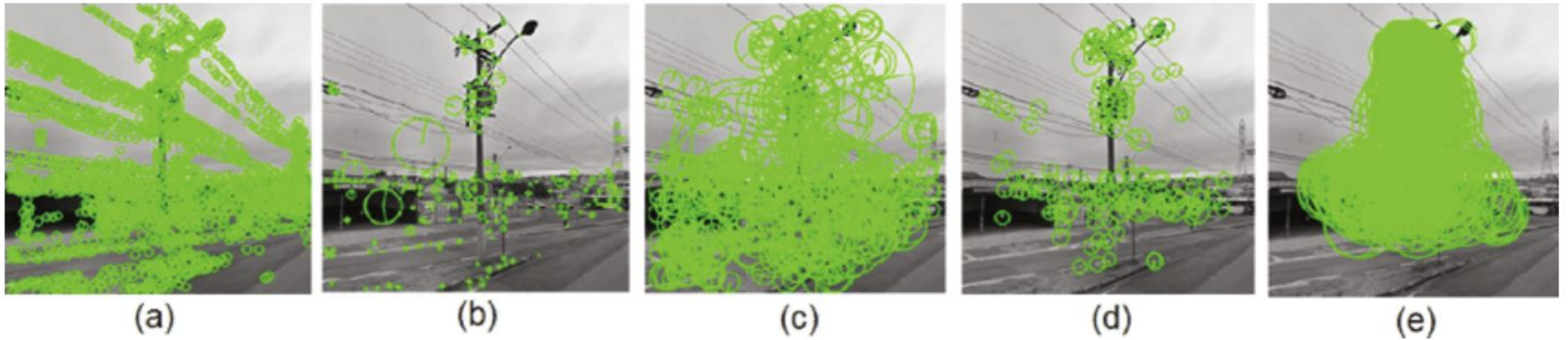
Each keypoint now has a location + scale + orientation + 128 rotation + scale + illumination (why?) invariant features associated with it

SIFT features example



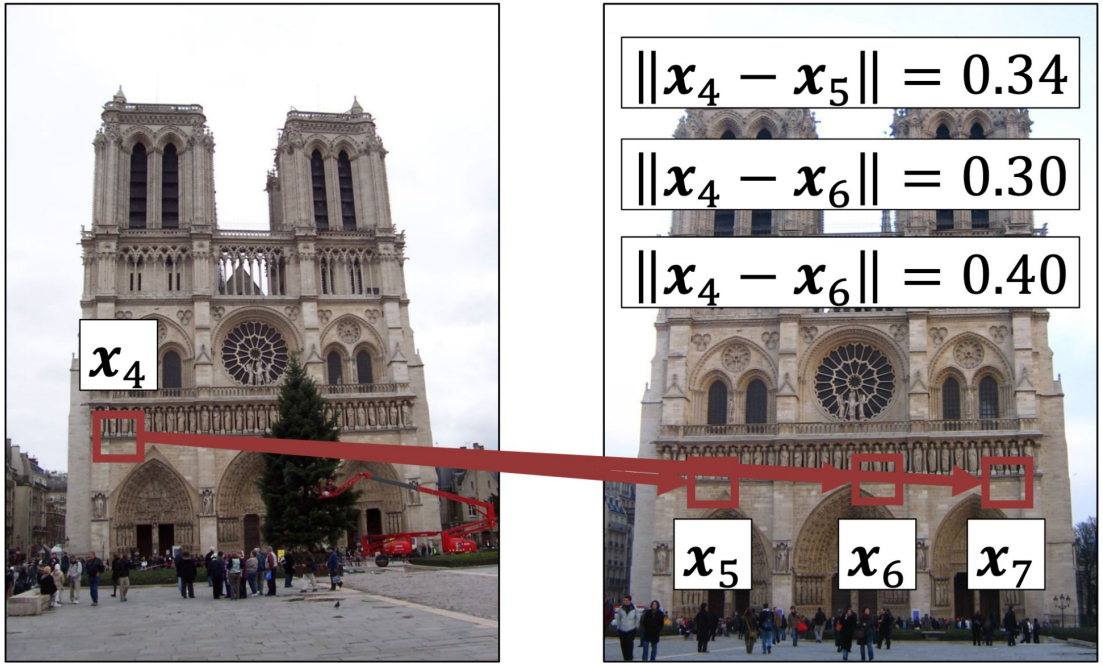


Many invariant feature detectors out there

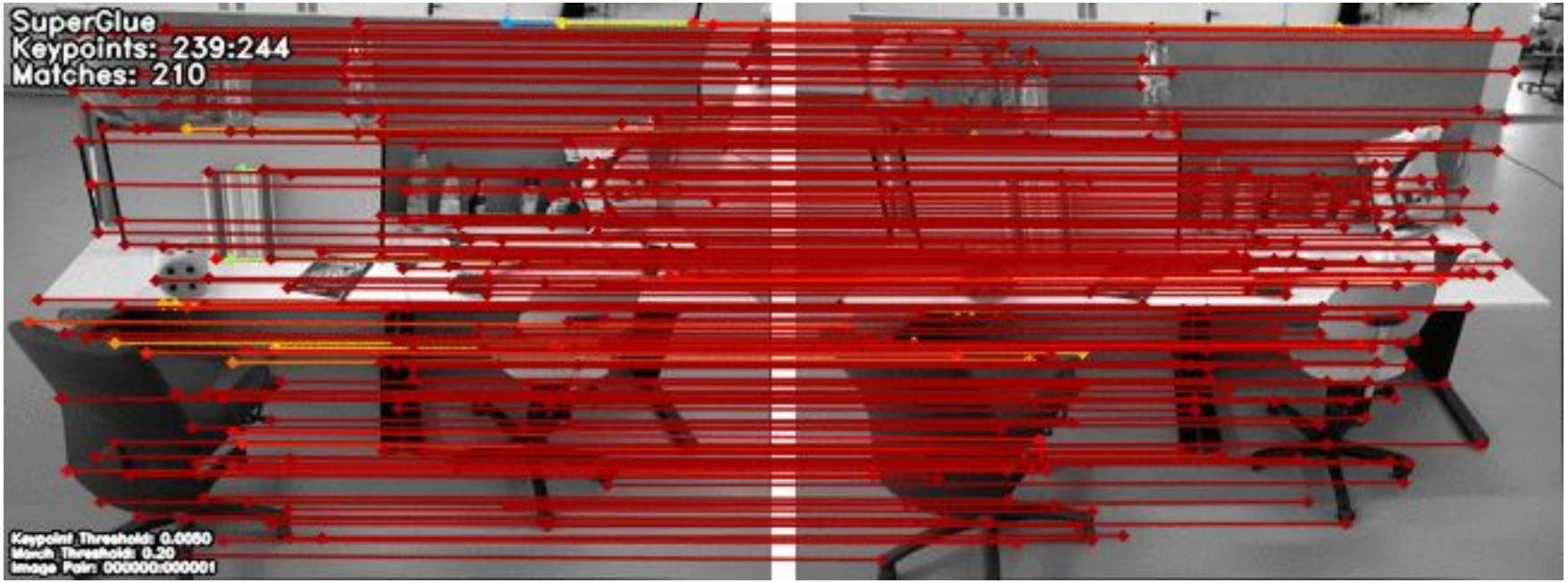


Feature extraction methods. (a) FAST. (b) SIFT. (c) SURF. (d) BRISK. (e) ORB. (Color figure online)

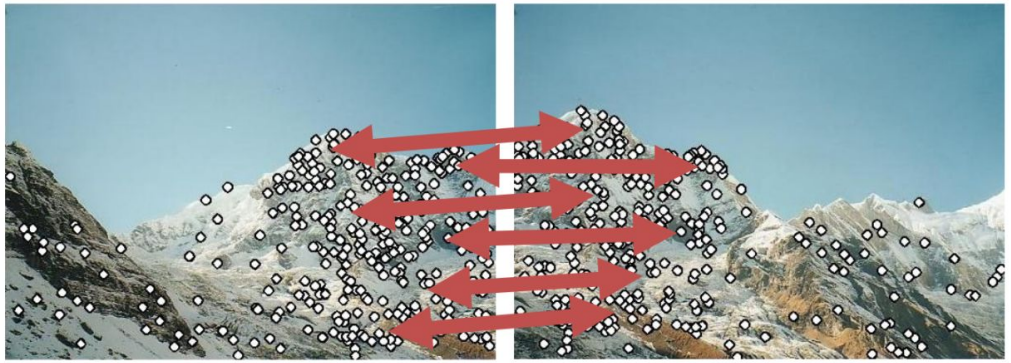
Once you have keypoint descriptors, match keypoints across images by minimum distance

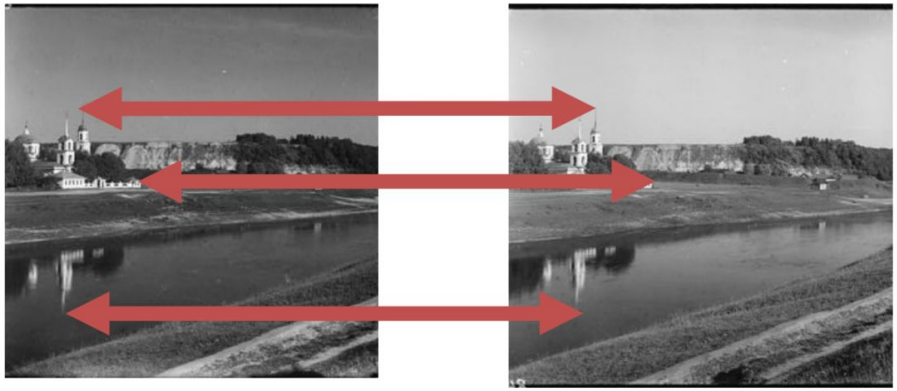


If there are many keypoints that match, we can solve transformations to align the two images

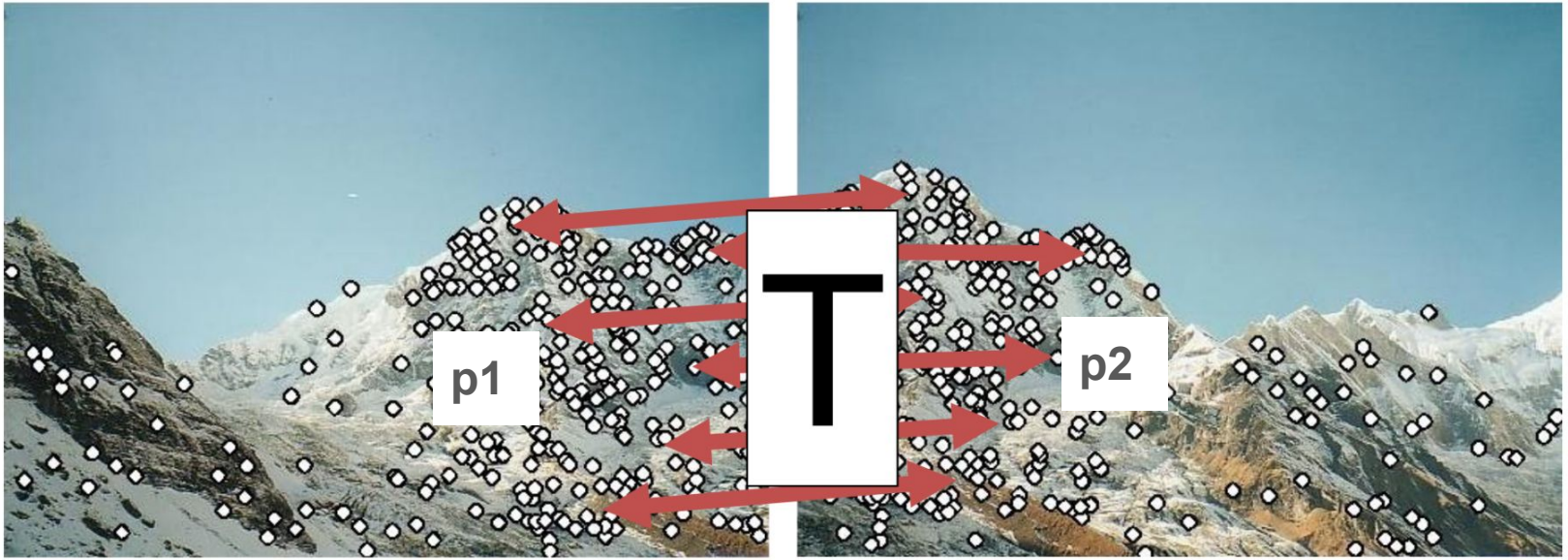


Given pairs **p1**,**p2** of correspondence, **how do I align?**





Solving for a Transformation



4) Find transformation T such that p1 = Tp2 (will cover transformations in next lecture).

Next: code lab

Next week: Image transformations

Original Image

After rotation of 45°