Features

Features

This section will cover :

- Edges & Gradients
- Corners
- Contours

These topics will be **implemented** and **tested** in Python with OpenCV.

Edges & Gradients

Edges & Gradients

<u>What is an Edge?</u>

Who's asking?

A better question might be :

What does a *human* think an edge is?

or

What does a *machine* think an edge is?

Edges & Gradients

What do humans think an edge is?

Original

***Source: Berkeley Segmentation Data Set and Benchmarks 500 (BSDS500)**

P. Arbelaez, M. Maire, C. Fowlkes and J. Malik., “Contour Detection and Hierarchical Image Segmentation”, IEEE TPAMI, Vol. 33, No. 5, pp. 898-916, May 2011.

Edges & Gradients

What do machines think an edge is?

Algorithm 2

Edges & Gradients

<u>What causes edges?</u>

Edges come from various sources :

- Lighting discontinuities

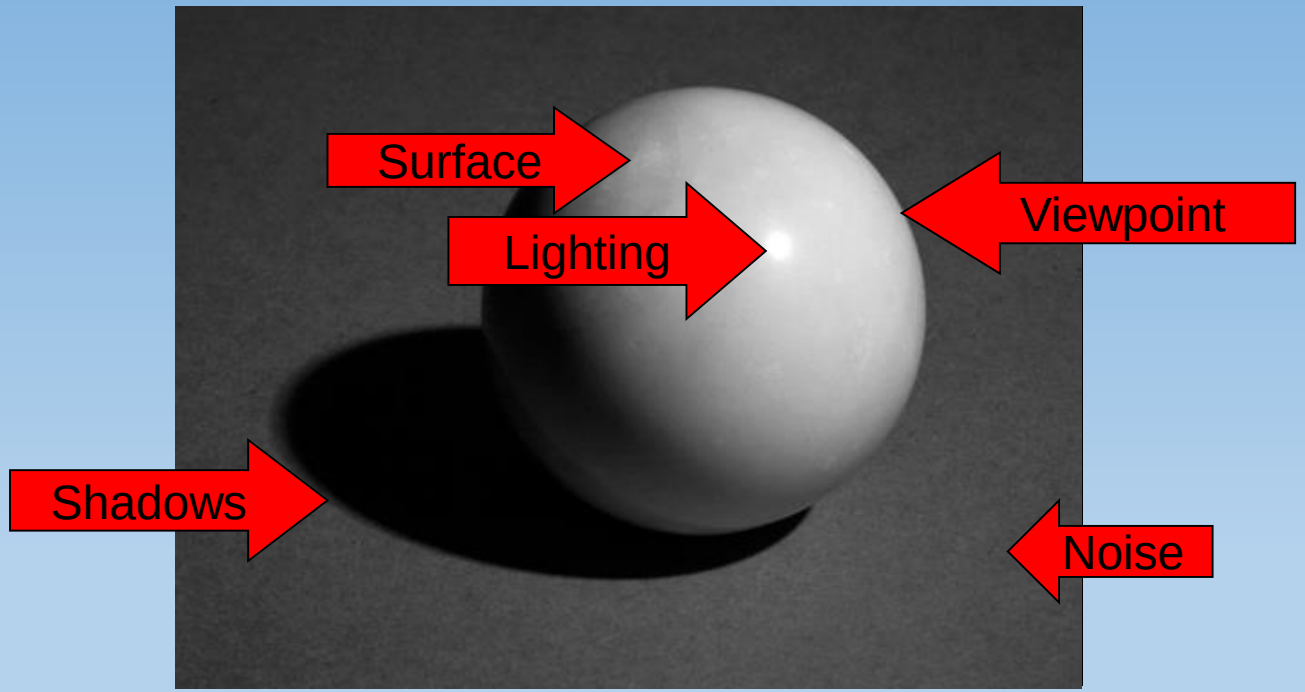
- Shadows

- Surface variations

- Noise

- Viewpoint (invisibility of the other side)

Edges & Gradients



Edges & Gradients

So what is an Edge?

An edge is a sharp change in the image.

This can be found by first examining the *gradients* of the image.

This gradient is then thresholded to identify the edges.

Edges & Gradients

Mathematically Speaking…

The gradient of the image can be calculated in a specific direction :

$$I_X = \frac{\partial I}{\partial x} \quad \text{and} \quad I_Y = \frac{\partial I}{\partial y}$$

The magnitude of these is the gradient image: $G = \sqrt{I_X^2 + I_Y^2}$

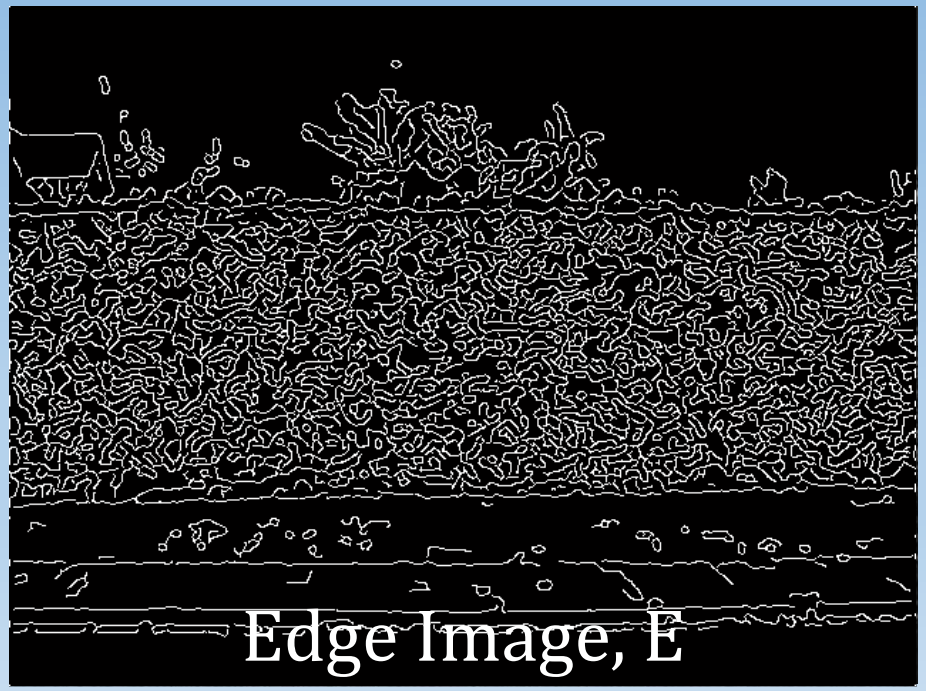
And the edge image is achieved by thresholding :

$$E = \begin{cases} 1 \; where \; G > T \\ 0 \; where \; G < T \end{cases}$$

Edges & Gradients

What is an Edge?

This is a hedge:



Edge Image, E

Edges & Gradients

In theory

The gradient of an image is the rate of change of the intensity.

It is usually calculated in a specific direction and defined by the partial derivatives :

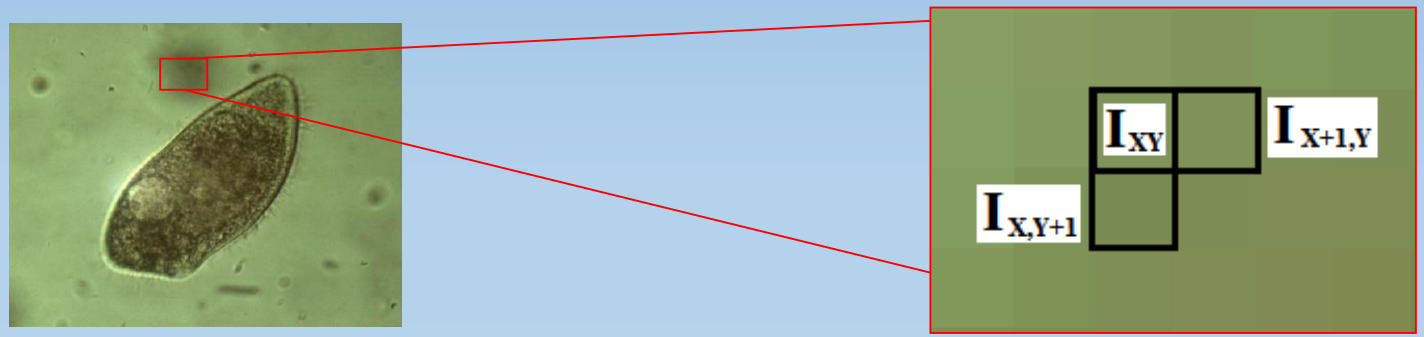
$$I_X = \frac{\partial I}{\partial x} \quad \text{and} \quad I_Y = \frac{\partial I}{\partial y}$$

Edges & Gradients

In Practice

Digital images are not continuous signals so derivatives make no sense.

Instead, they are approximated by simply subtracting the pixel intensity by its neighbour in the specific direction.



$$I_X = I_{X+1,Y} - I_{XY} \quad \text{and} \quad I_Y = I_{X,Y+1} - I_{XY}$$

Edges & Gradients

As Kernels

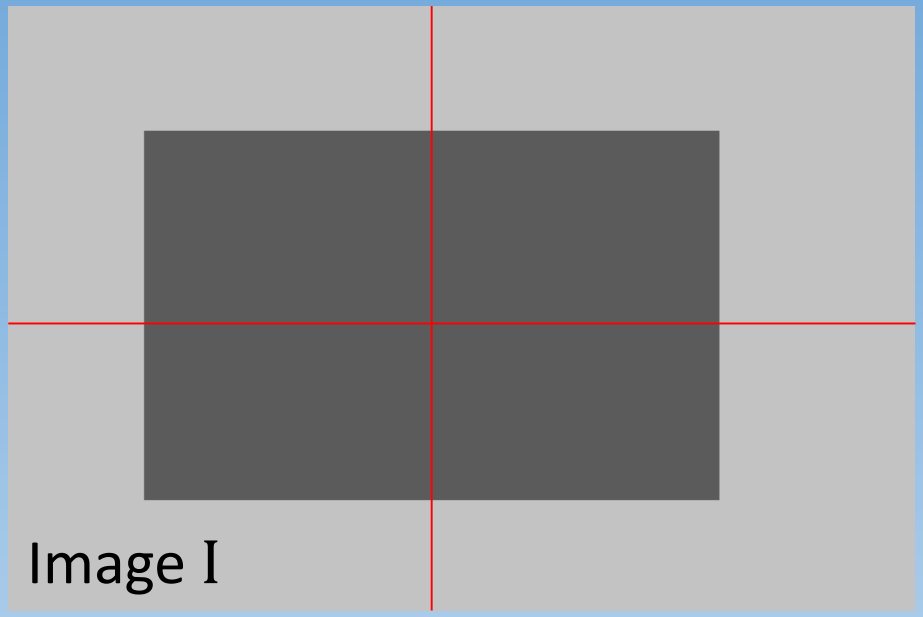
This discrete approximation can be described as kernels.

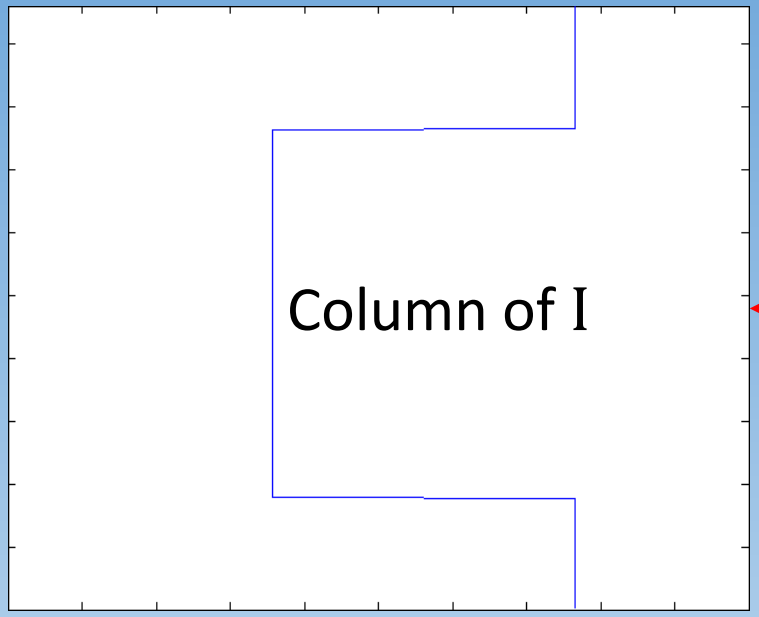
The gradients are then calculated as convolutions of the image with these kernels.

$$G_X = \begin{bmatrix} -1 & 1 \end{bmatrix} \quad \text{and} \quad G_Y = \begin{bmatrix} -1 \\ 1 \end{bmatrix}$$

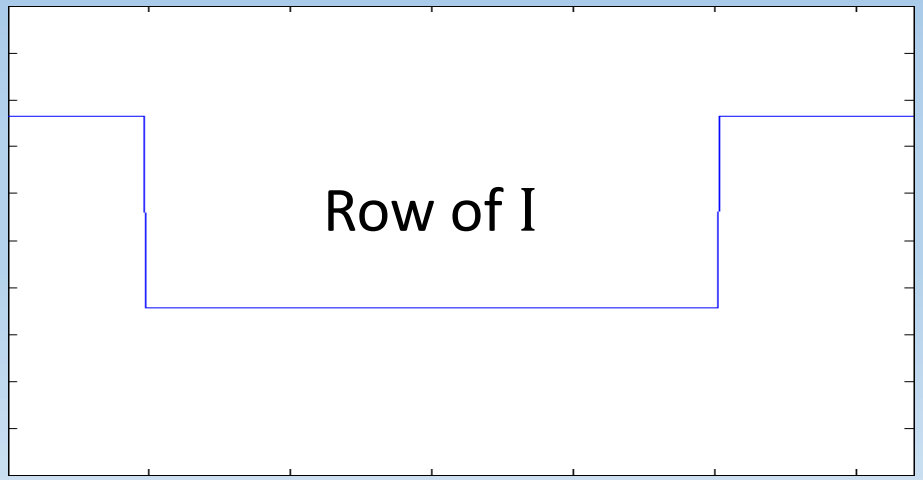
$$I_X = G_X * I \quad \text{and} \quad I_Y = G_Y * I$$

Edges & Gradients





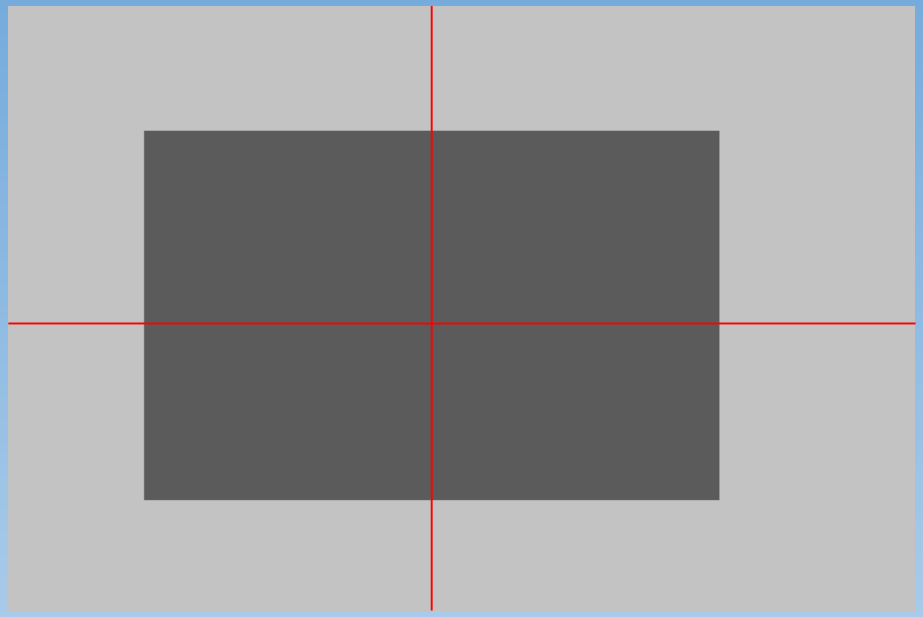
Intensity →

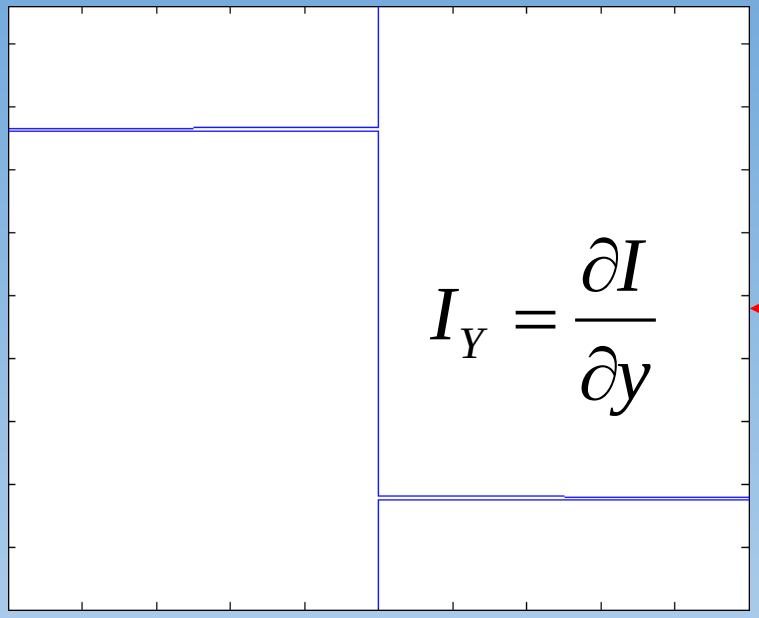


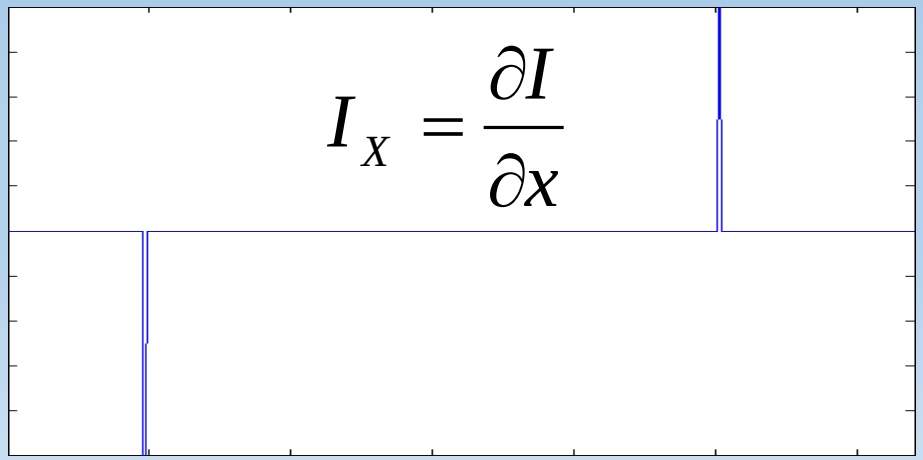
Intensity →

These graphs show the intensity
along a row
and along a column

Edges & Gradients

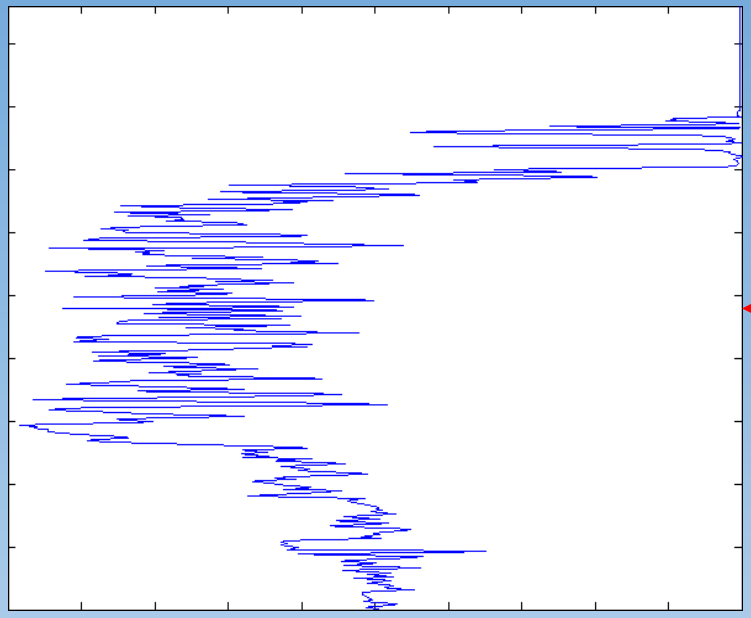


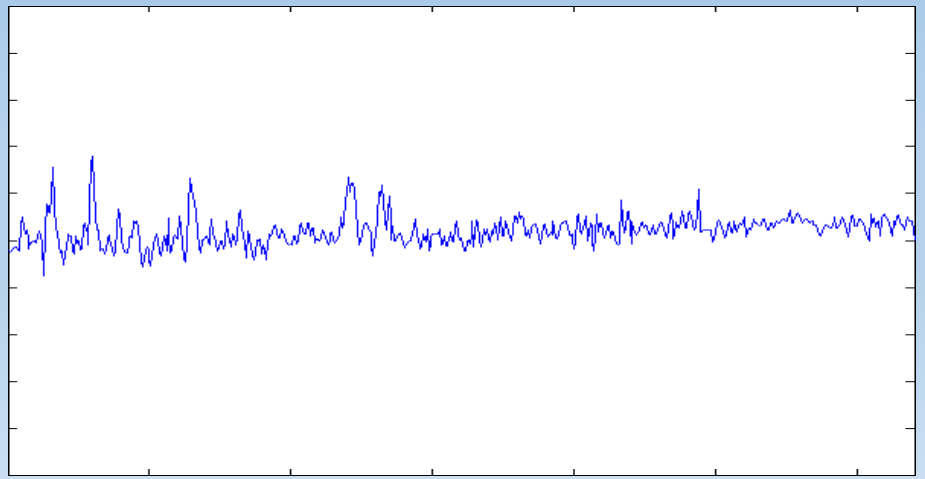




These graphs show the gradient along the same row and column

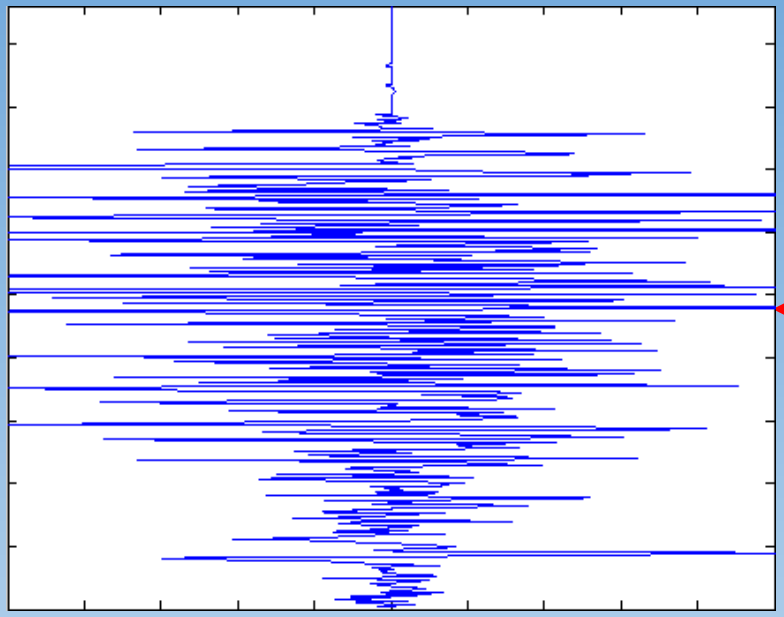
Edges & Gradients

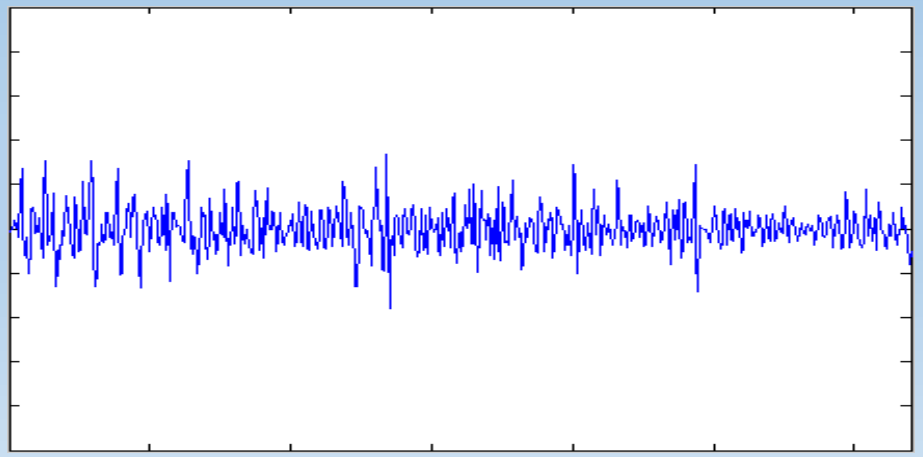




Intensity
along a row
and along a column

Edges & Gradients





Gradient
along the same row
and column

Edges & Gradients

Noise Effects

In the real image, the problem of noise is apparent.

Firstly, the intensity variation is affected.

Then, this is accentuated when the intensity gradient is calculated.

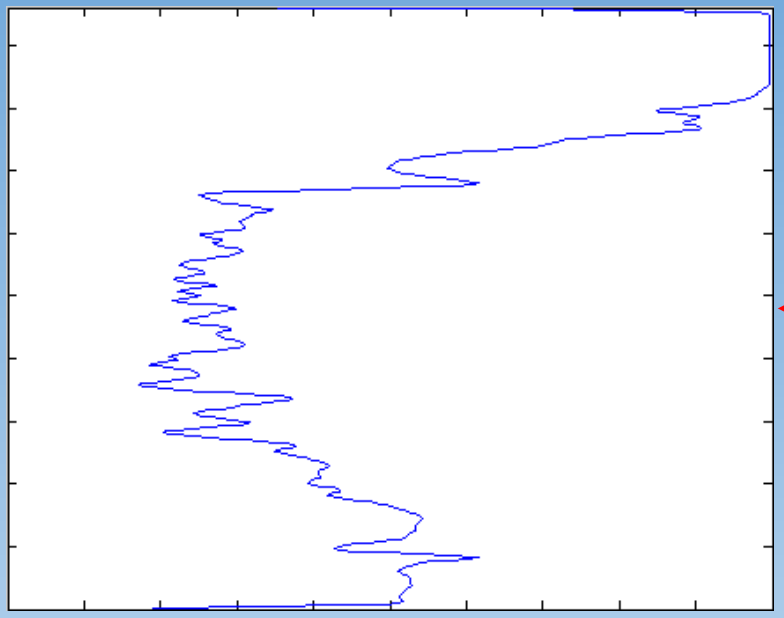
Edges & Gradients

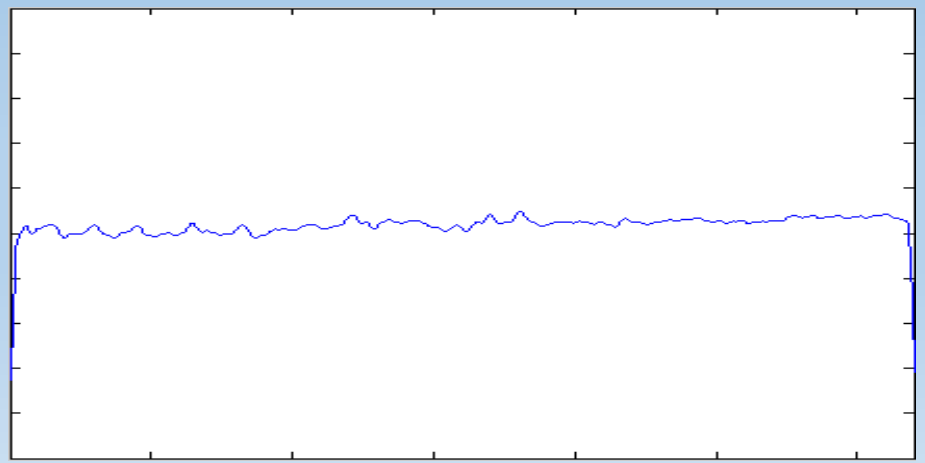
Noise Filtering

To overcome effects of noise, the image is first filtered using a noise filter, e.g. a Gaussian filter:

$$G = \frac{1}{27}\begin{bmatrix} 1 & 4 & 1 \\ 4 & 7 & 4 \\ 1 & 4 & 1 \end{bmatrix}$$

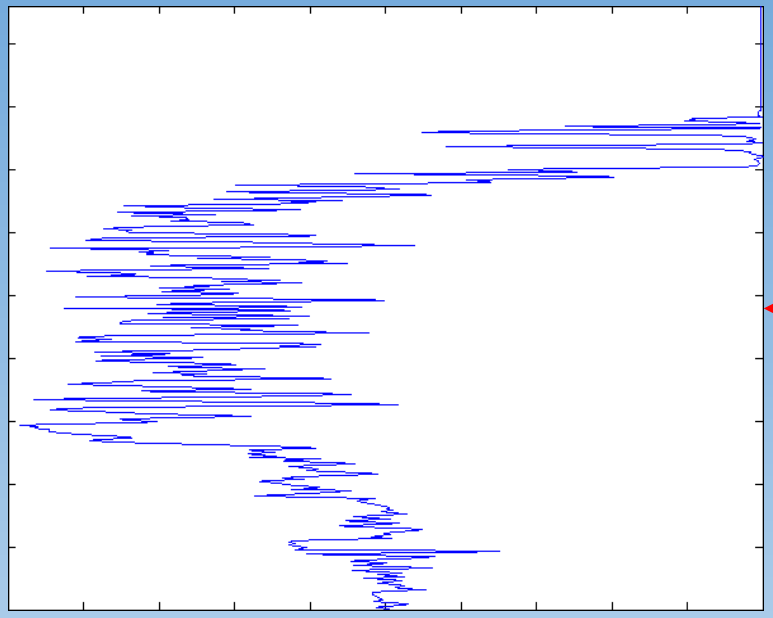
Edges & Gradients

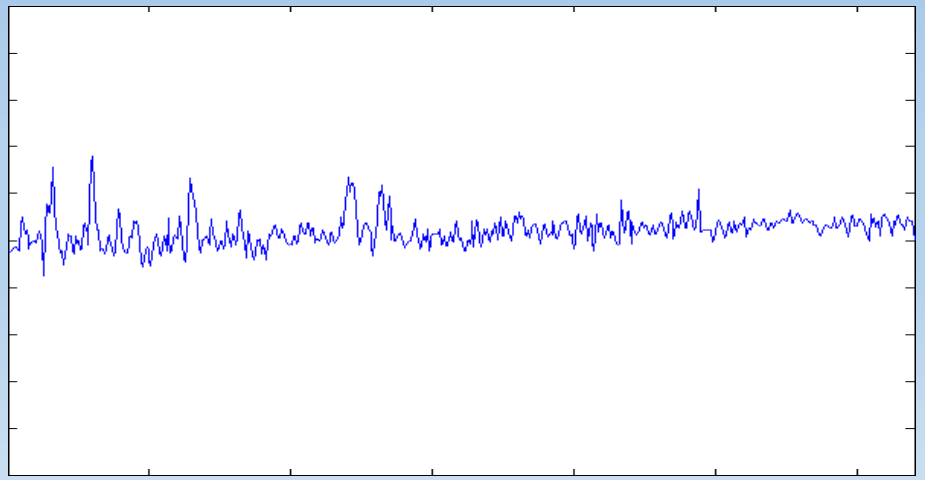




Filtered intensity
along a row
and along a column

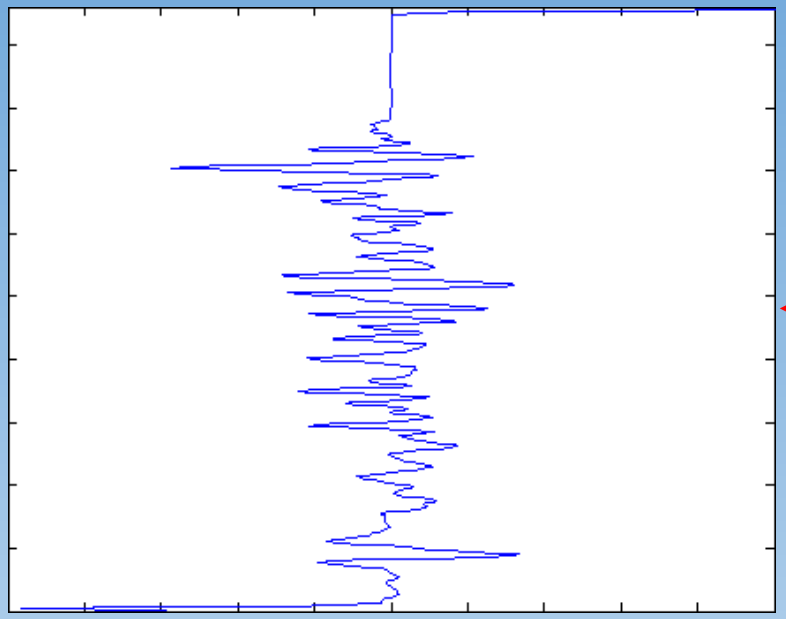
Edges & Gradients

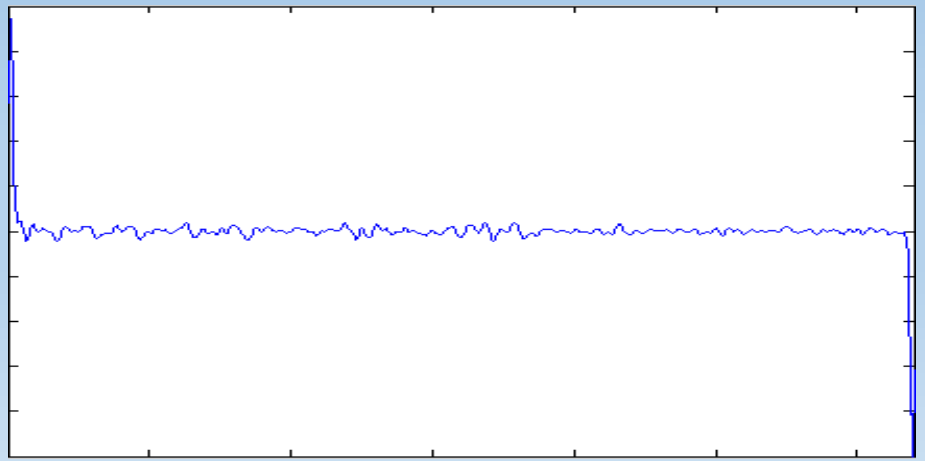




Unfiltered intensity along the same row and column

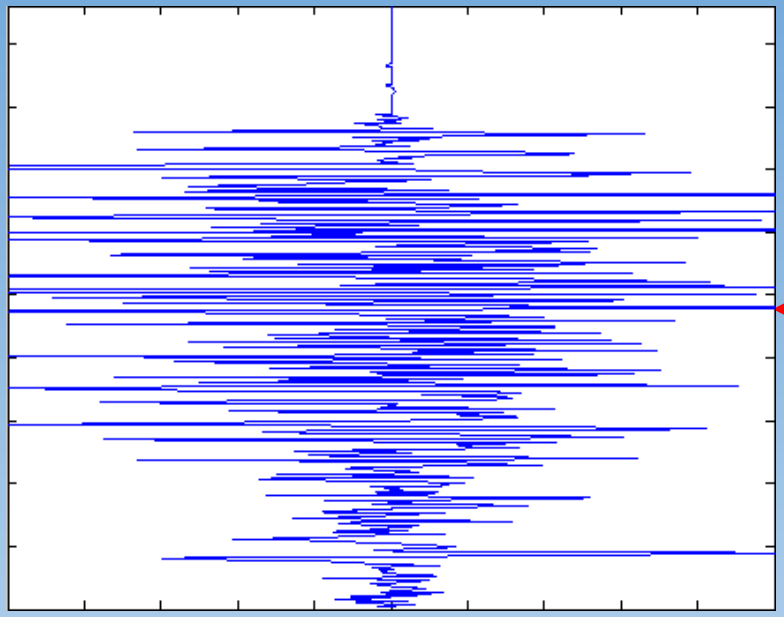
Edges & Gradients

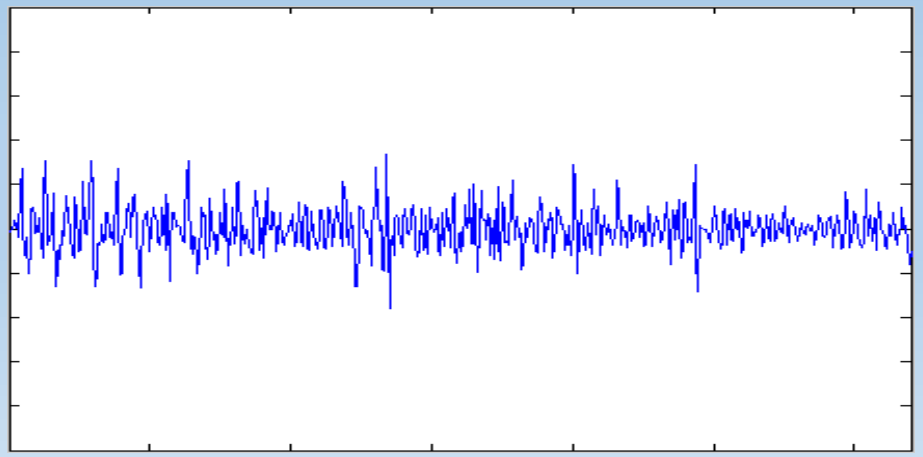




Filtered gradients
along a row
and along a column

Edges & Gradients





Unfiltered gradients along the same row and column

Edges & Gradients

Gradient Filters

The effect of filtering is most dramatic in the gradient profile.

Noise filtering is sometimes done separate from gradient calculation.

More often, the two are combined into improved gradient filters.

Edges & Gradients

Better Filters

The gradient approximation kernels can be improved.

They are crude, sensitive to noise and directionally biased.

Better filters exist, including Sobel, Prewitt, Roberts Cross, etc.

Here, we will investigate the Sobel filter as an example.

Edges & Gradients

Sobel Filters

To improve on the basic filters, Sobel (and poor forgotten Feldman!) came up with an improvement on the simple filters:

$$G_X = \frac{1}{4}\begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \quad \text{and} \quad G_Y = \frac{1}{4}\begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

Edges & Gradients

Sobel Gradients

These operators are used in the same way as the simple filters :

$$I_X = G_X * I \quad \text{and} \quad I_Y = G_Y * I$$

The gradient magnitude and direction can also be found :

$$G = \sqrt{I_X^2 + I_Y^2} \quad \text{and} \quad \theta = \tan^{-1}\left(I_Y / I_X \right)$$

Edges & Gradients

Polarity

As a result of using a gradient filter, the output will have not only gradient magnitude and direction but also *polarity*, i.e. positive and negative edges.

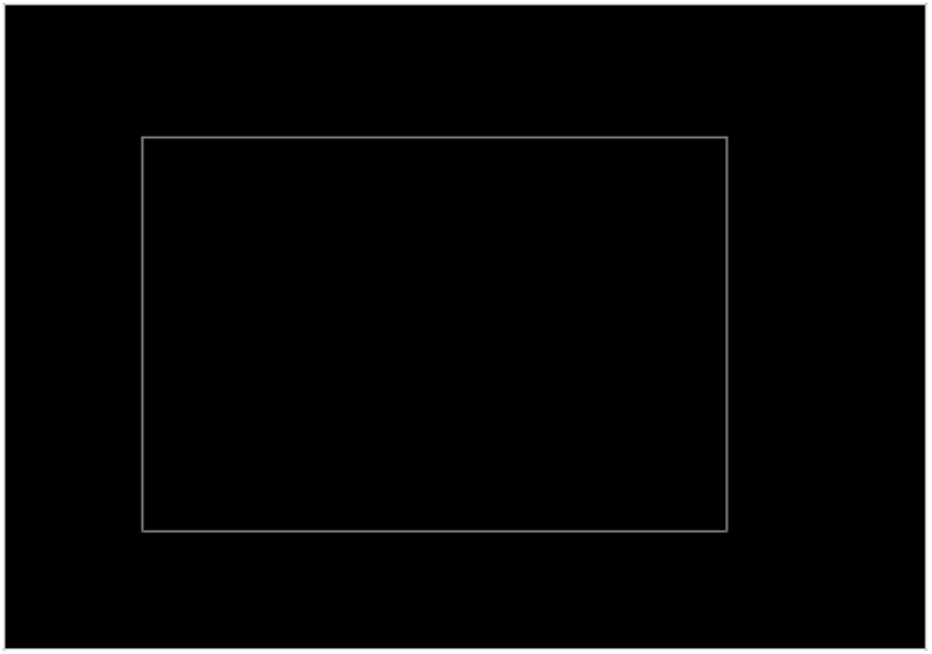
This can cause problems with visualisation and so the resultant gradient images are often normalised to the range 0 to 255 with mid-grey representing homogeneity (zero gradient)….

Original

Horizontal

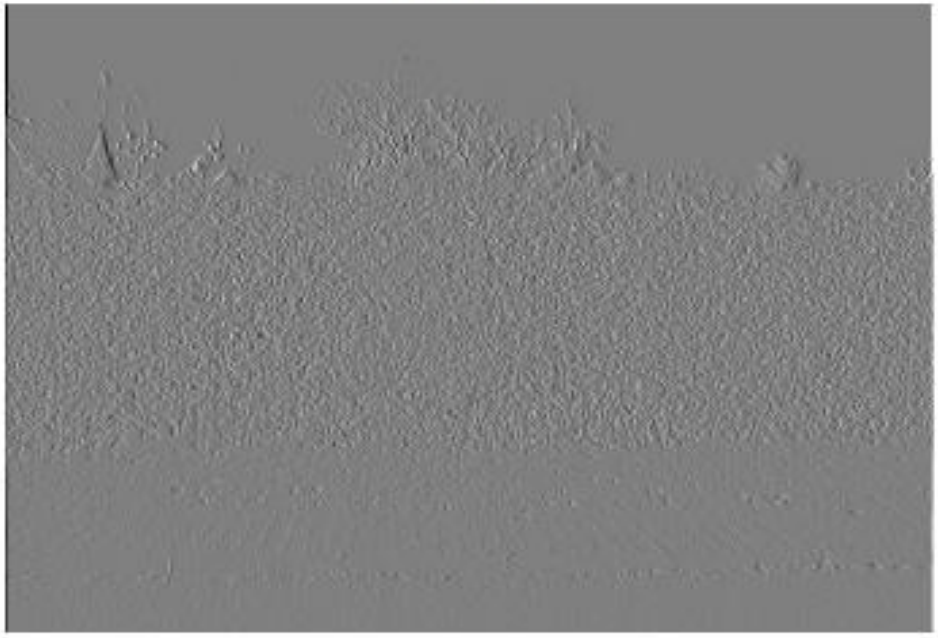
Vertical

Magnitude

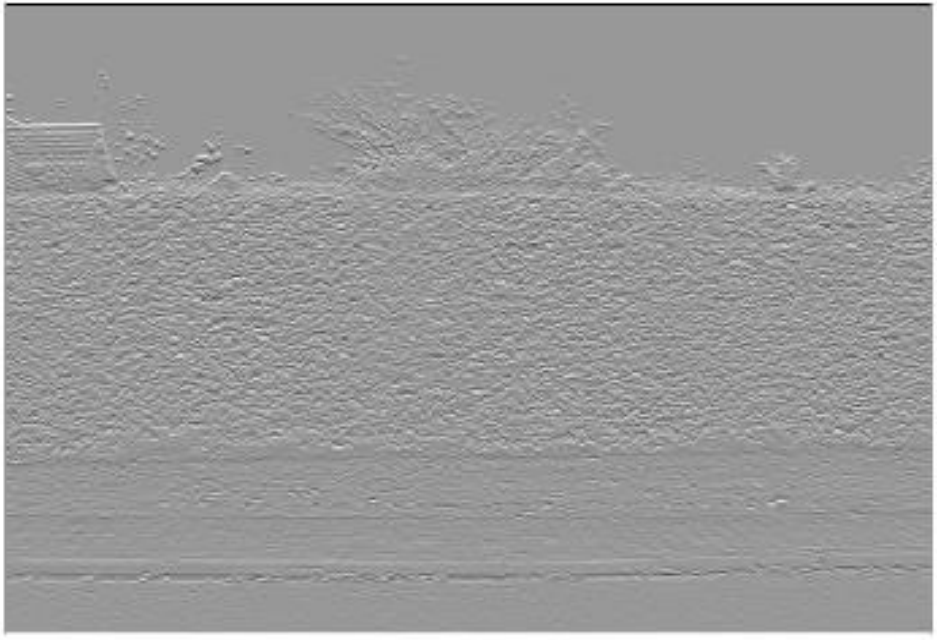


Original

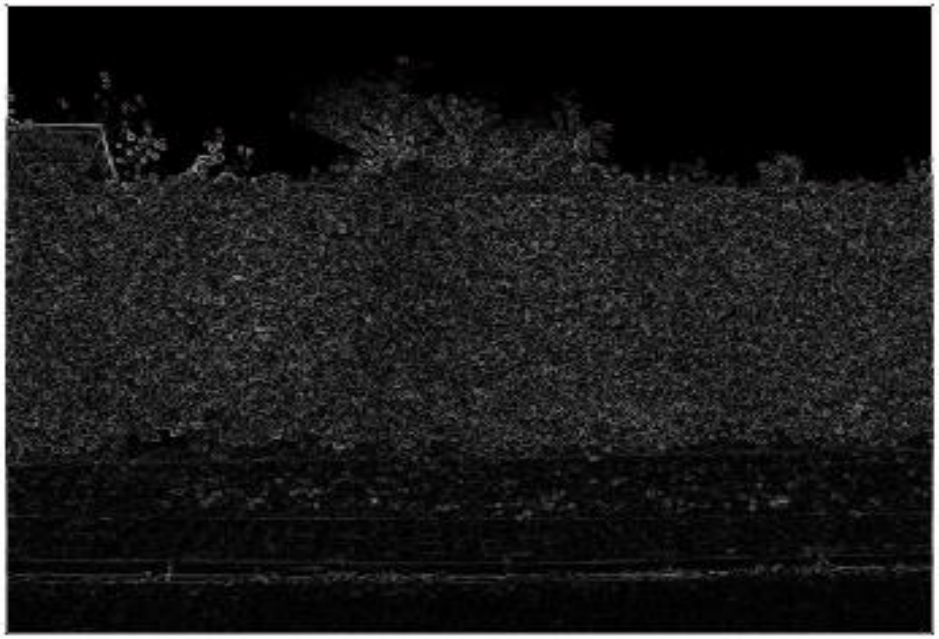
Horizontal



Vertical



Magnitude



Edges & Gradients

From Gradients to Edges

The result of applying a gradient filter will be the gradient of the image.

From this, we can acquire the magnitude of the gradient.

However, while this suggests the “edginess” of a pixel, it does not specify which pixels are on edges.

For that we need an edge detector.

Edges & Gradients

Edge Detectors

An edge detector should return a result of “edge” or “not an edge” for every pixel in the image.

The easiest step from a gradient to an edge detector is by thresholding.

In more sophisticated designs, the issues of noise and threshold selection are addressed.

Examples include Marr, Haralick and Canny detectors.

Edges & Gradients

Canny Edge Detector

The Canny Edge Detector sorts edges into three categories:

“definitely edges”, “maybe edges” and “definitely not edges”.

This reduces the problem to dealing only with the “maybe edges”.

It also includes noise smoothing and edge thinning for added clarity.

In Python with OpenCV

Sobel

To extract the gradients, we can use the *Sobel* function:

```
Ix = cv2.Sobel(G,ddepth=cv2.CV_64F,dx=1,dy=0)
Iy = cv2.Sobel(G,ddepth=cv2.CV_64F,dx=0,dy=1)
```

`G` is the original grayscale image.

The 64-bit ddepth (`ddepth=cv2.CV_64F`) allows for the handling of negative numbers.

The parameters `dx` and `dy` set the orders of the x and y derivatives.

Canny

To extract the edges, we can use the *Canny* function:

```
E = cv2.Canny(I,threshold1=100,threshold2=200)
```

`I` is the original image.

`threshold1` and `threshold2` set the upper and lower thresholds for “definitely edges” and “definitely not edges”.

This code will return the strongest edges with gradients above 100.

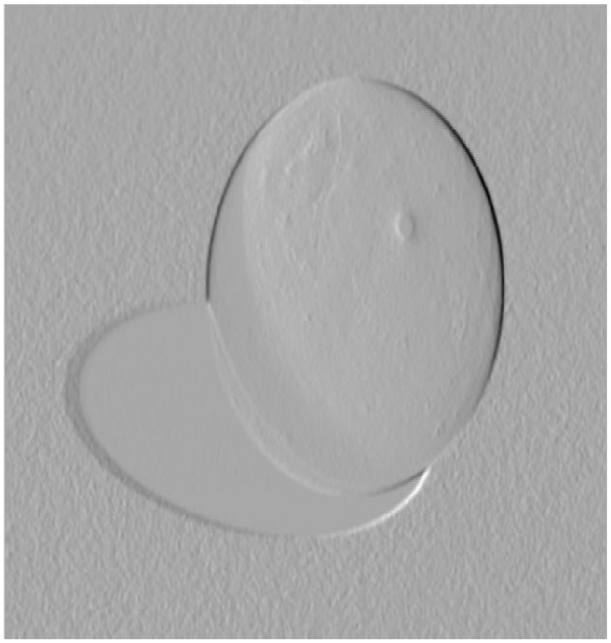
Task : Edges & gradients

1. Open any image;
2. Use Sobel to find the horizontal and vertical gradients;
3. Create an edge mask using Canny (255 for edges 0 for not);
4. On the edges, show the magnitude of the gradients.

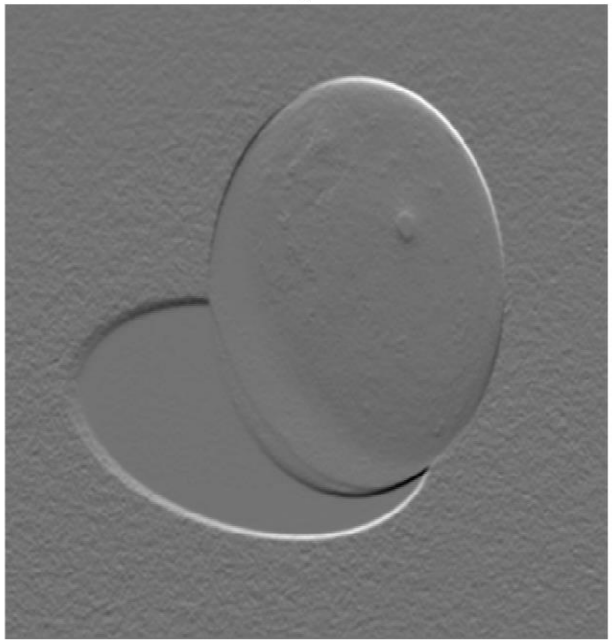
Advanced Task:

Devise a way to visualise the direction of the gradients on the edges.

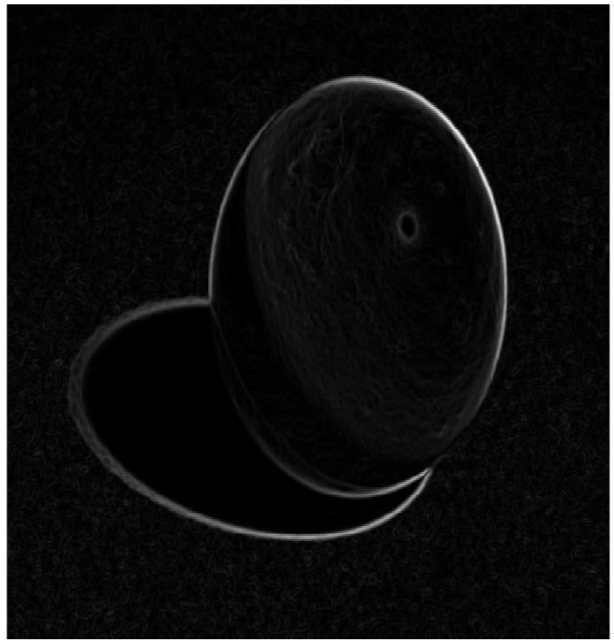
Ix



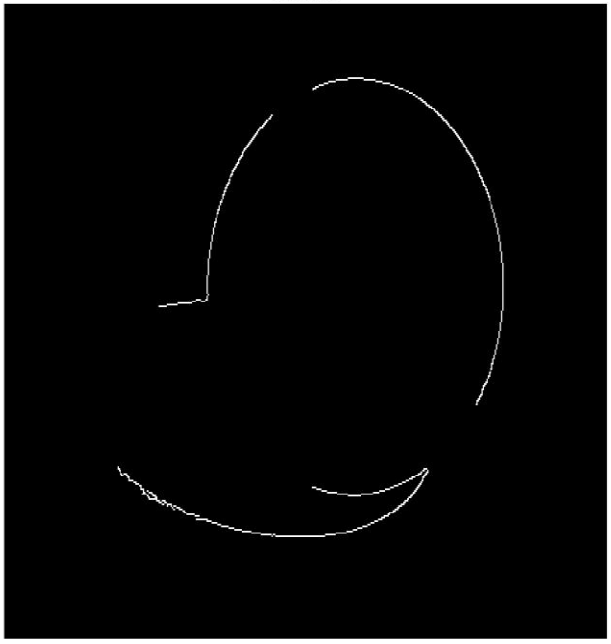
Iy



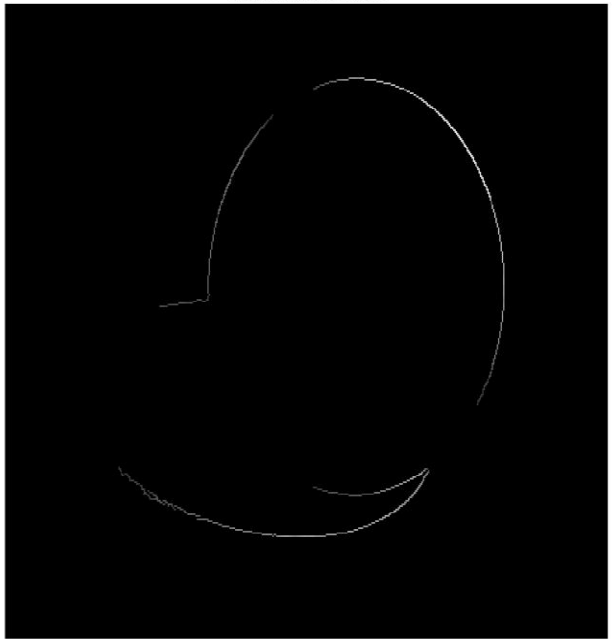
Gradient Magnitude



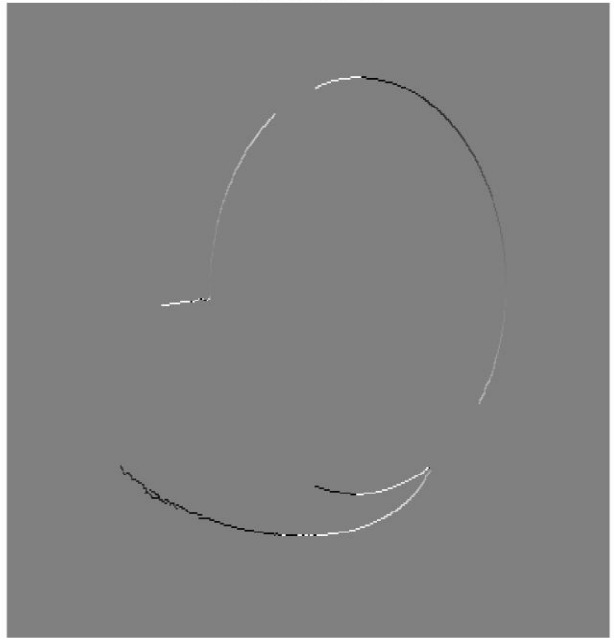
Edge Mask



Gradient Magnitude of Edges



Gradient Direction of Edges



Corners

Corners

What is a good feature to track?

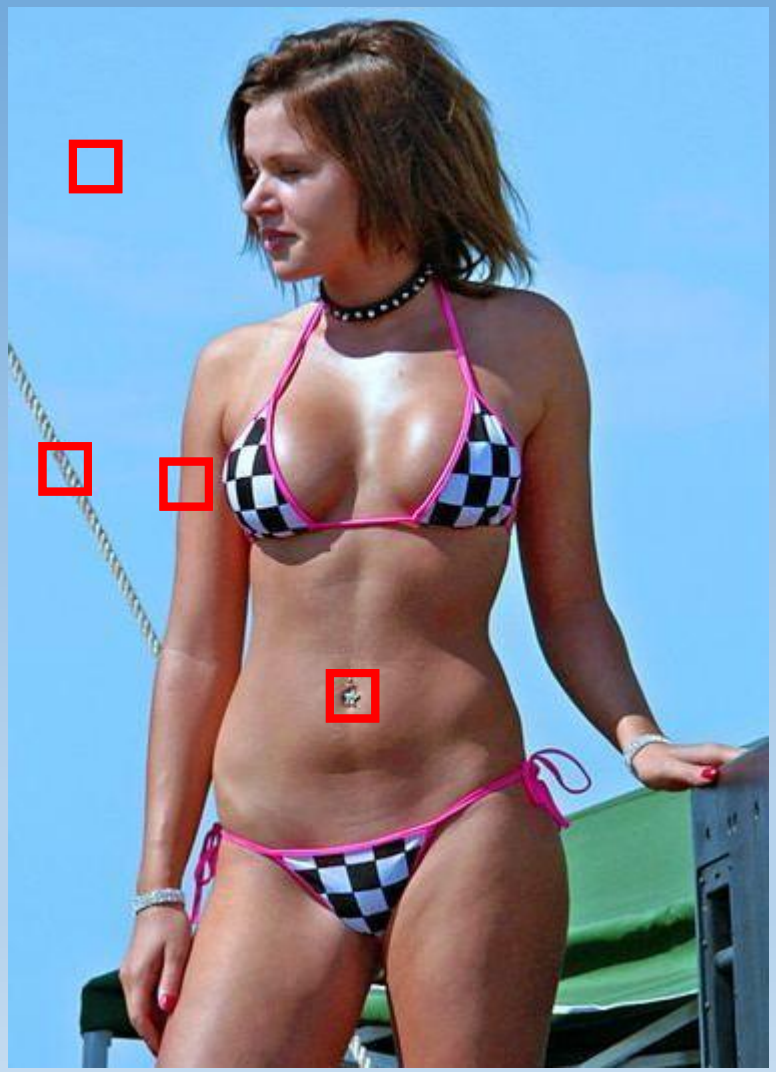
A good feature is interesting.

While edges are distinctive features in an image, they are not necessarily uniquely identifiable.

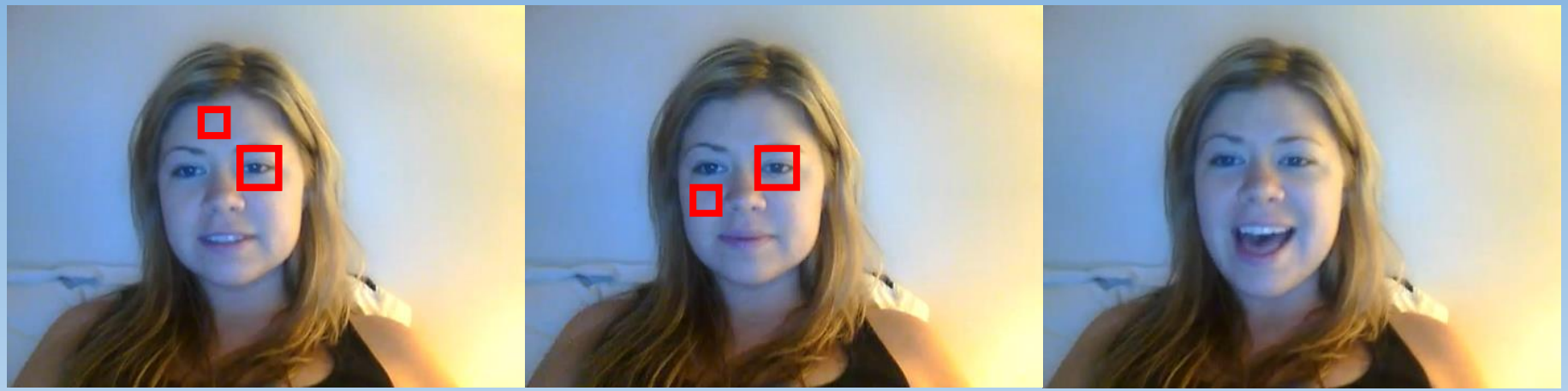
This makes them less ideal for computer vision algorithms such as matching and tracking.

For this we need corners.

Corners



Corners



Corners

Defining Good Features

A good feature has the following attributes:

- Distinct from other image areas;
- Robust to occlusion or clutter;
- Insensitive to noise;
- Insensitive to brightness variations;
- Invariant to scale and orientation.

Corners fit all these criteria and so make perfect features.

Corners

Corner Detectors

Many corner detectors exist including Hessian, Harris, Laplacian,

Difference of Gaussian (DoG), etc.

Here, we will investigate some commonly used ones.

Generally, corner detection begins with a look at *Autocorrelation*.

Corners

Autocorrelation

Autocorrelation, also known as *self similarity*, describes how interesting an image region is.

It looks at the horizontal and vertical gradients in the image area and determines whether the area is homogenous or variant.

Autocorrelation can pick up edges and texture but it is often used to seek out corners which are good image features.

Corners

Mathematically speaking…

Autocorrelation can be described for a given image, pixel or region by the *autocorrelation matrix*:

$$M = \begin{pmatrix} I_X^2 & I_X I_Y \\ I_X I_Y & I_Y^2 \end{pmatrix}$$

I_X and I_Y are the horizontal and vertical gradients respectively.

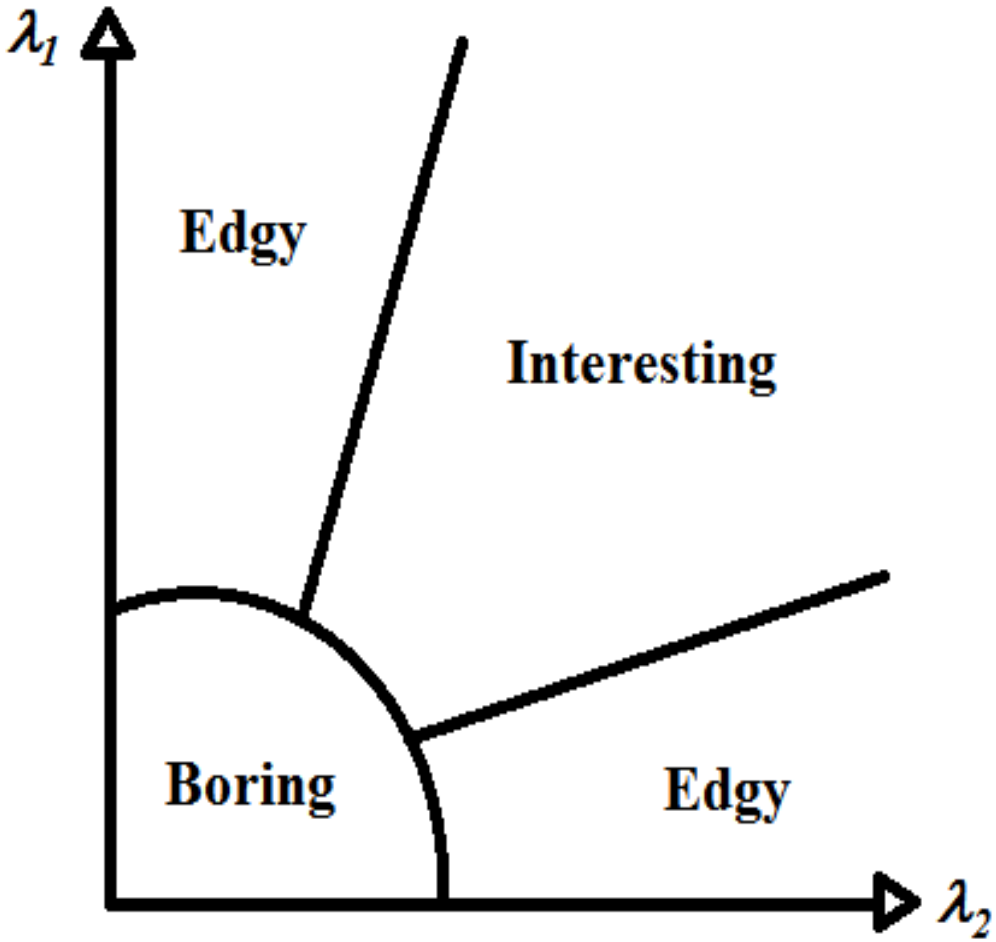
Corners

Interpretation

The eigenvalues of this matrix, λ_1 and λ_2, give a lot of information about the image area:

- No large eigenvalues suggests homogeneity;
- One large eigenvalue suggests an edge;
- Two large eigenvalues suggests a corner.

Corners are interesting.


λ_1
Edgy
Interesting
Boring
Edgy
λ_2

Corners

‘Cornerness’

Harris and Stephens (1988) came up with an approximation of the eigenvalues and returned a ‘cornerness’ measure.

This eliminated the need for calculation and testing of the eigenvalues.

The ‘cornerness’ measure, H, includes a tunable parameter, α, which is almost always set to 0.04:

$$H = I_X^2 I_Y^2 - (I_X I_Y) - \alpha(I_X^2 + I_Y^2)$$

Corners

Harris Algorithm

The implementation of the Harris algorithm involves the following steps:

1. Calculate I_X and I_Y by applying horizontal and vertical gradient filters (e.g. Sobel);

2. Calculate I_X^2, I_Y^2 and $I_X I_Y$;

3. Apply a Gaussian smoothing filter to each of these;

4. Calculate the Harris image, H.

5. Apply a threshold to isolate the strongest corners.

Corners

Example

Let’s try this with the following image…


Valeo

Corners

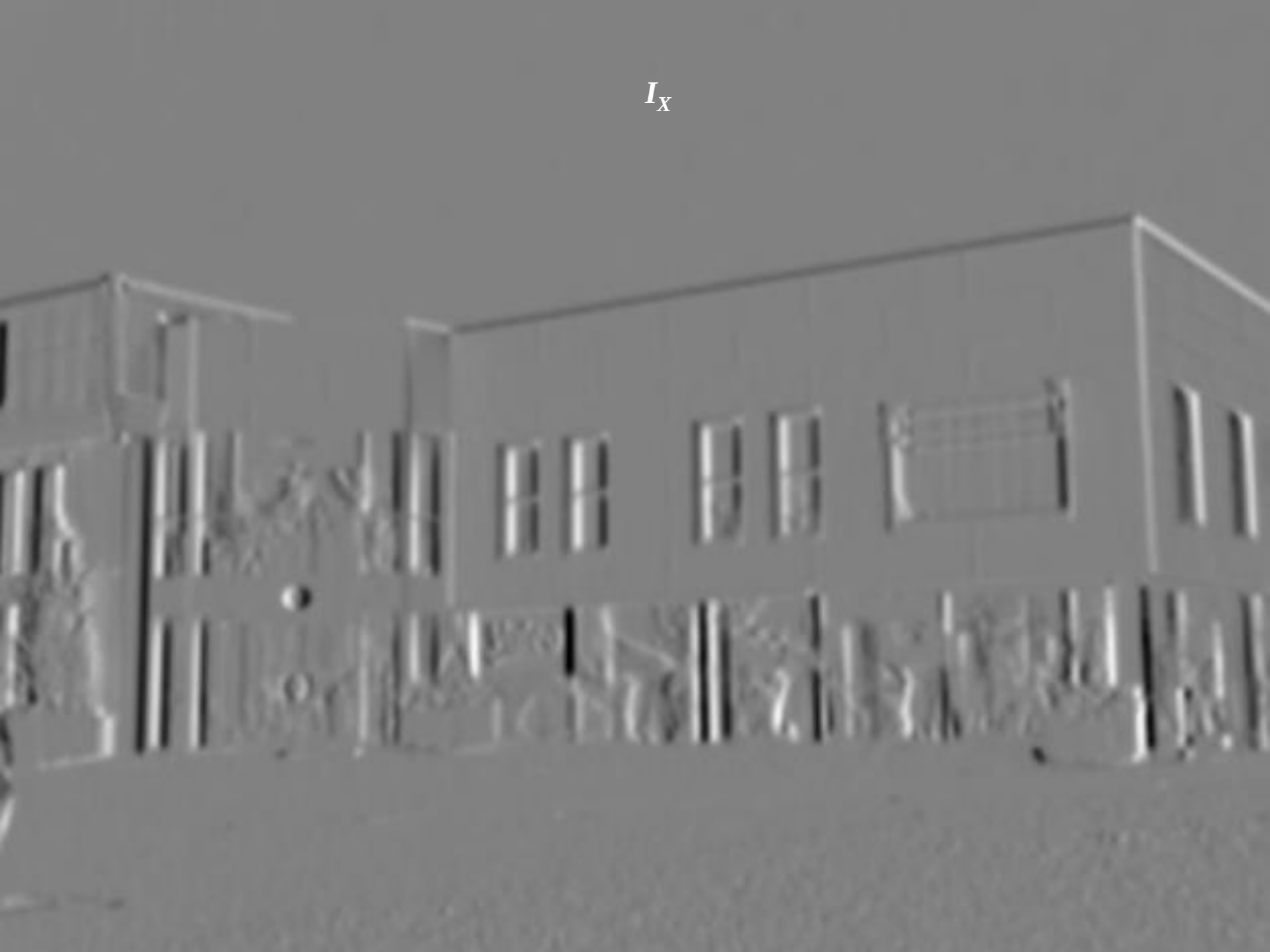
Gradients

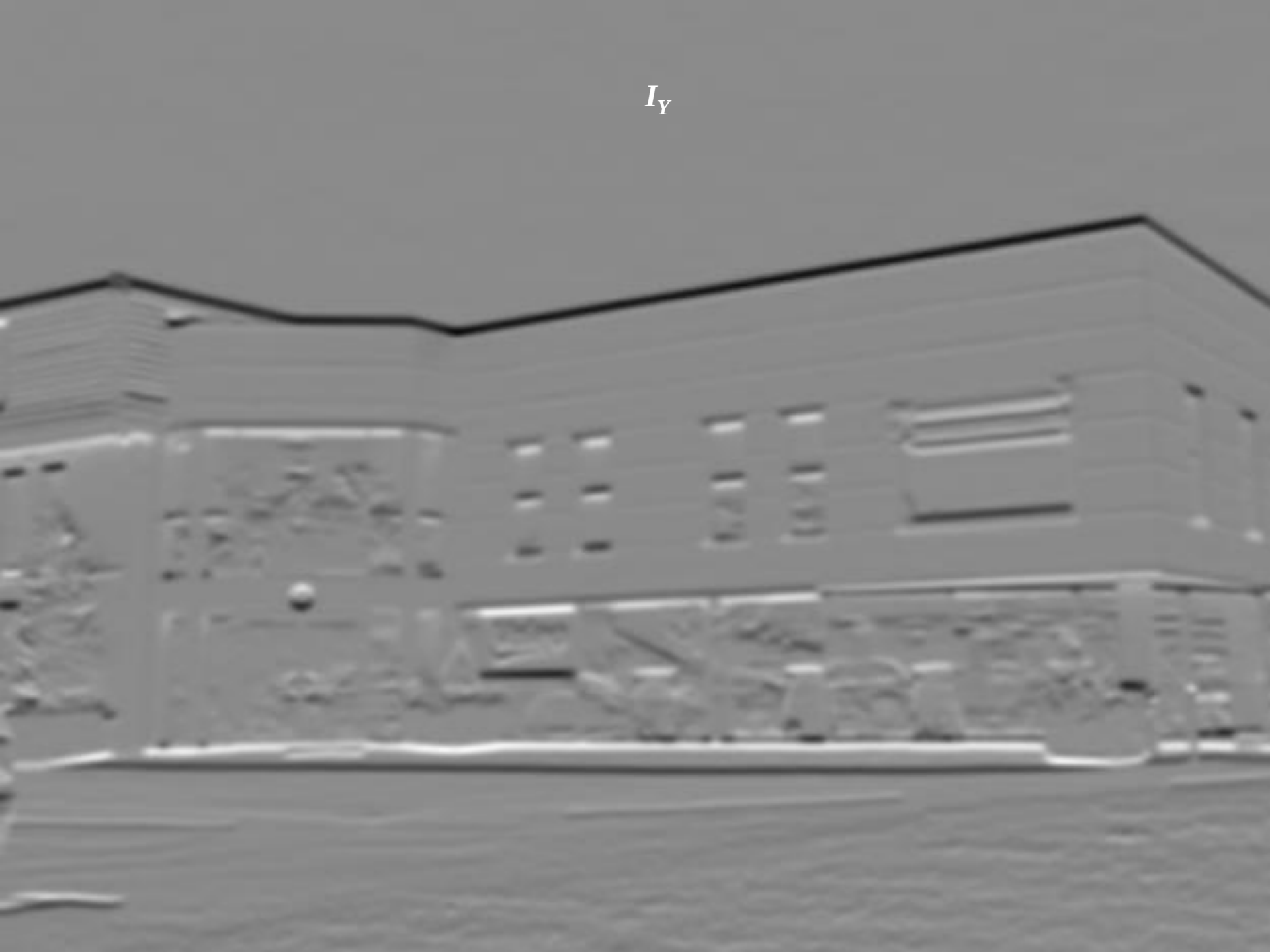
First, the horizontal and vertical gradients, I_X and I_Y, of the image are found by applying horizontal and vertical Sobel filters.

I_X is the horizontal gradient and highlights strong vertical edges.

I_Y is the vertical gradient and highlights strong horizontal edges.

Both can be in the range -1 to 1 since gradients have polarity.


I_X


I_Y

Corners

Autocorrelation Matrix

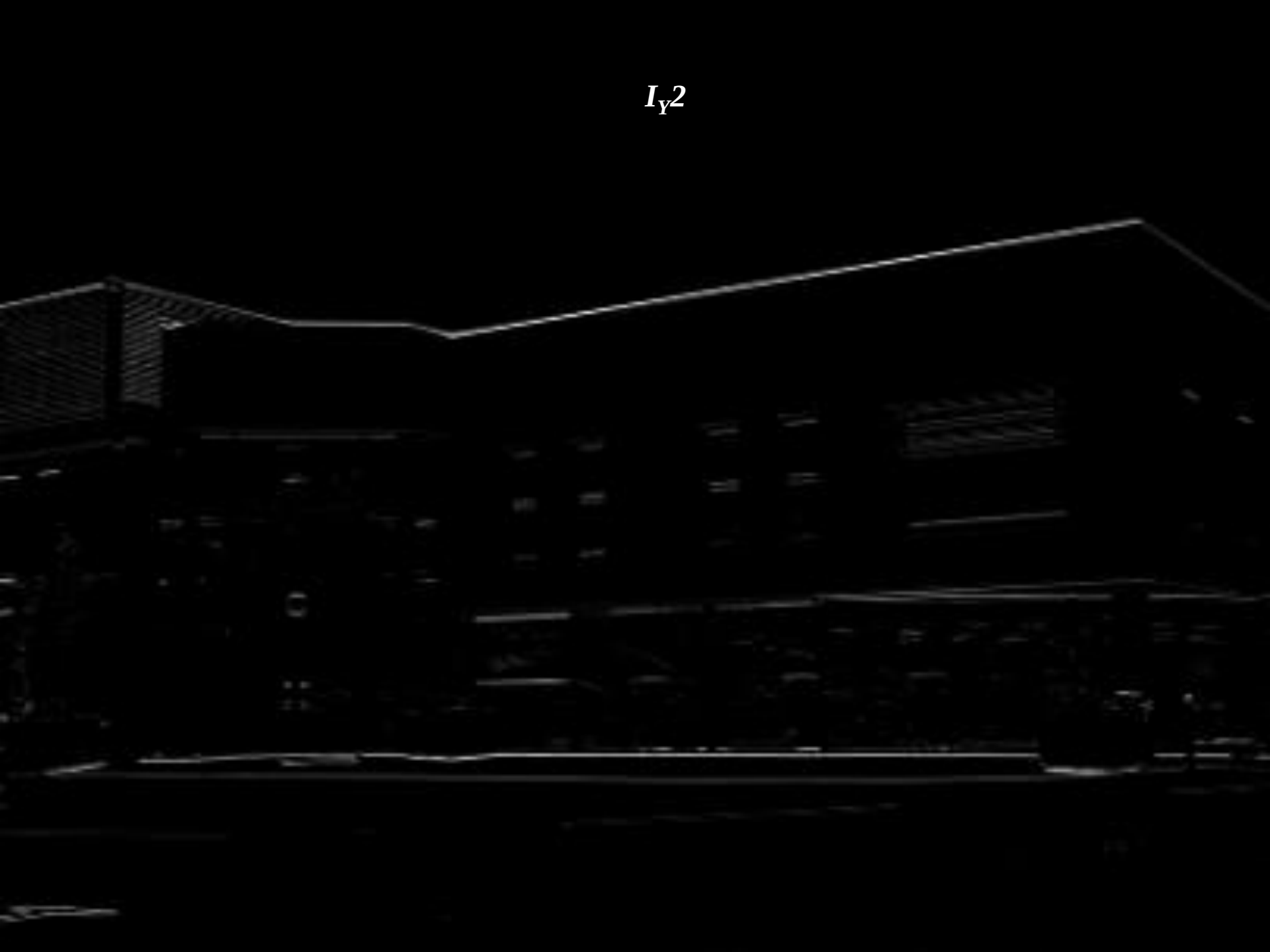
Next, the elements of M, I_X^2, I_Y^2 and $I_X I_Y$, are found.

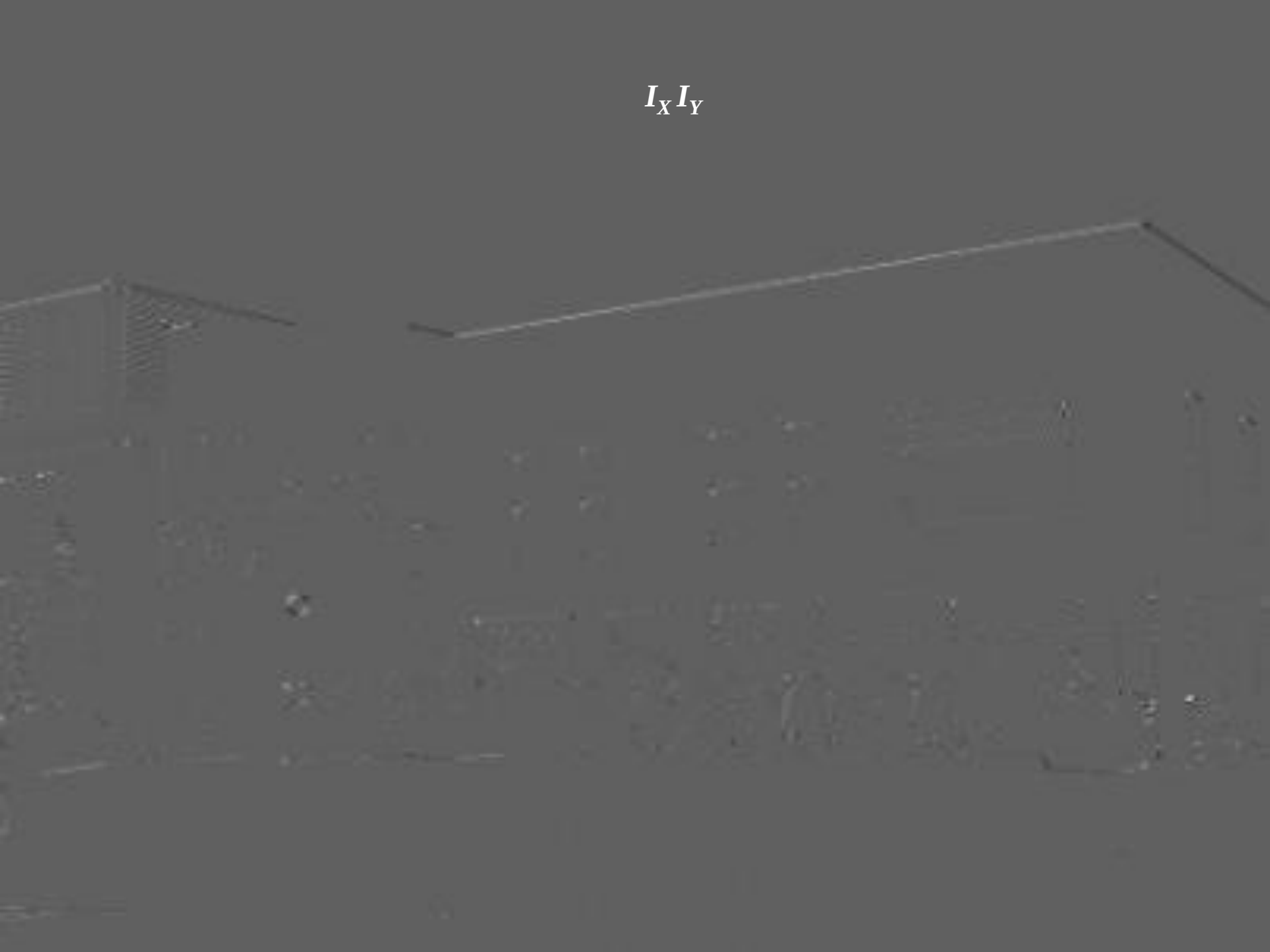
Once I_X and I_Y have been found, these can be multiplied by themselves and each other to create I_X^2, I_Y^2 and $I_X I_Y$.

A Gaussian filter is applied to these to smooth them.

I_X^2 and I_Y^2 are both positive due to the squaring while $I_X I_Y$ is in the range -1 to 1.


I_X2


I_Y2


I_X I_Y

Corners

Harris Image

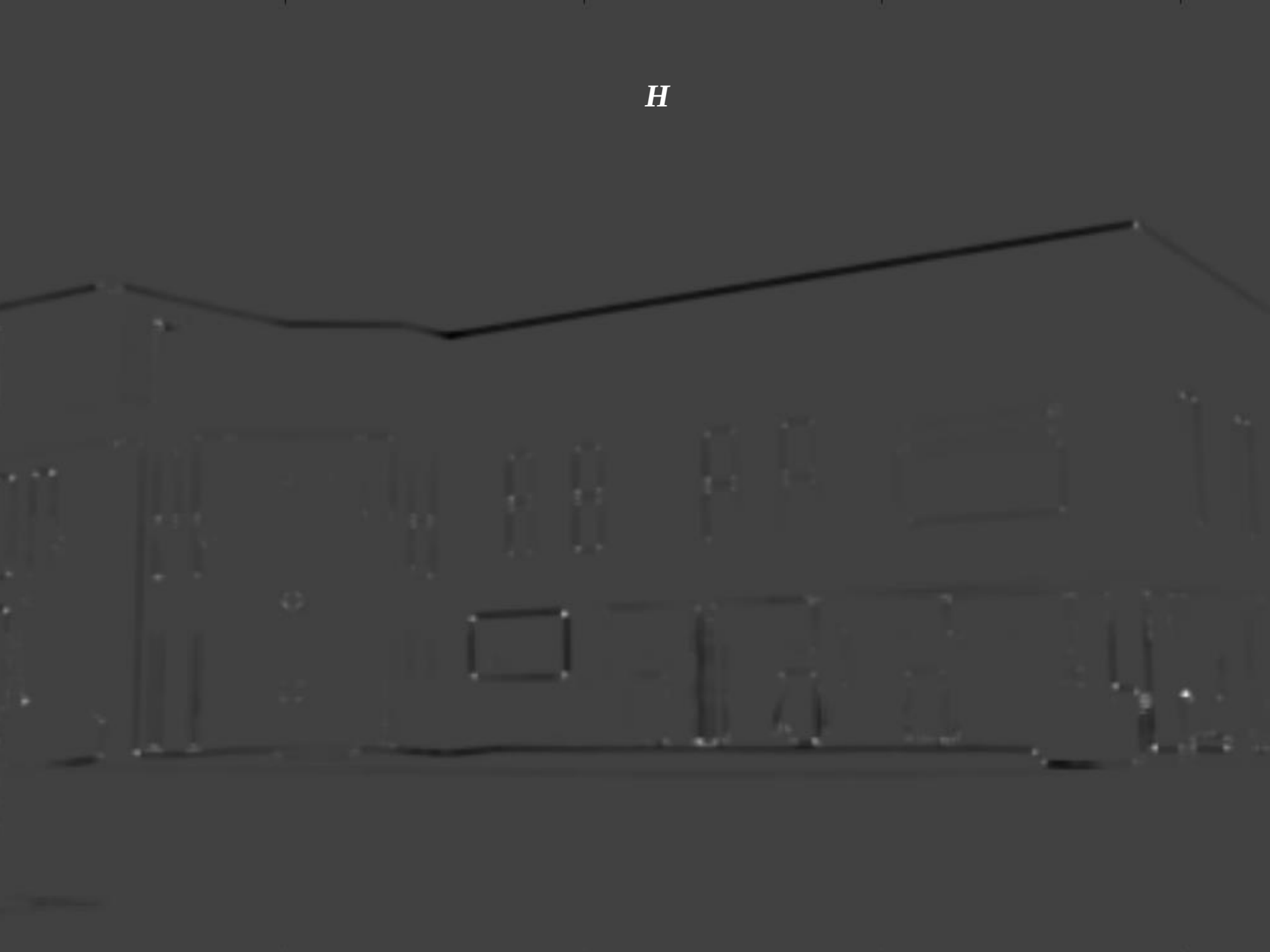
Finally, the Harris image H, showing the 'cornerness' of each pixel, is constructed.

The Harris image is calculated using the previous images combined with the tuneable α factor:

$$H = I_X^2 I_Y^2 - (I_X I_Y) - \alpha(I_X^2 + I_Y^2)$$

This is in the range -1 to 1 with high negative values denoting edges and high positive values meaning corners.

H



Corners

Corner Map

A threshold is applied to find only the strongest corners.

This threshold is usually set by examining the statistical properties of the image, H.

The thresholded image, C is a map of the corners in the image:

$$\boldsymbol{C} = \begin{cases} 1 & \text{where } \boldsymbol{H} \geq \mathrm{T} \\ 0 & \text{where } \boldsymbol{H} < \mathrm{T} \end{cases}$$

C

Harris Corners

Corners

Improvements

The Harris Detector does a good job of finding the corners which make good features.

However, it misses some corners and finds multiple corners in the same location.

To improve on this, we return to the eigenvalues of the autocorrelation matrix.

This leads us to the *Shi-Tomasi Detector*.

Corners

Shi-Tomasi Corners

Six years after Harris and Stephens, Shi and Tomasi (1994) introduced a small change that made a noticeable improvement.

They returned to the Autocorrelation eigenvalues and noted that only the minimum eigenvalue need be found and tested to identify a corner.

$$ST = \min(\lambda_1, \lambda_2)$$

This added better discrimination without significantly increasing computation.

Corners

Additional Features

Shi-Tomasi also introduced a number of additional features:

- The minimum distance between corners can be set;
- Non-maximal suppression retains only the local maximum (best corner) in each area;
- A fixed number of corners can be retained.

Corners

Shi-Tomasi Algorithm

The implementation of the Shi-Tomasi algorithm extends beyond the calculation of their corner measure, ST:

1. Calculate ST (similar to calculation of H previously).

2. Perform non-maximal suppression.

3. Threshold using a set quality level to find corners.

4. Sort corners by quality measure in descending order.

5. Remove each corner where there is a stronger corner at a distance less than the set minimum distance.

6. Retain only the *N* strongest corners where *N* is set by the user.


Valeo
Shi-Tomasi Corners


Valeo
Harris Corners

In Python with OpenCV

Harris

To extract the cornerness, we can use the *cornerHarris* function:

```
H = cv2.cornerHarris(G,blockSize=5,ksize=3,k=0.04)
```

`G` is the original grayscale image.

`blockSize` sets the size of the Gaussian filter.

`ksize` sets the Sobel kernel size.

`k` is the alpha factor.

`H` is the Harris image showing the 'cornerness'.

Shi-Tomasi

For Shi-Tomasi corners, we use the *goodFeaturesToTrack* function:

```
Corners =
cv2.goodFeaturesToTrack(G,maxCorners=50,
qualityLevel=0.1,minDistance=10)
```

`G` is the original grayscale image.

`maxCorners` sets number of corners to find.

`qualityLevel` sets threshold for the strength of the corners.

`minDistance` sets the minimum distance between corners.

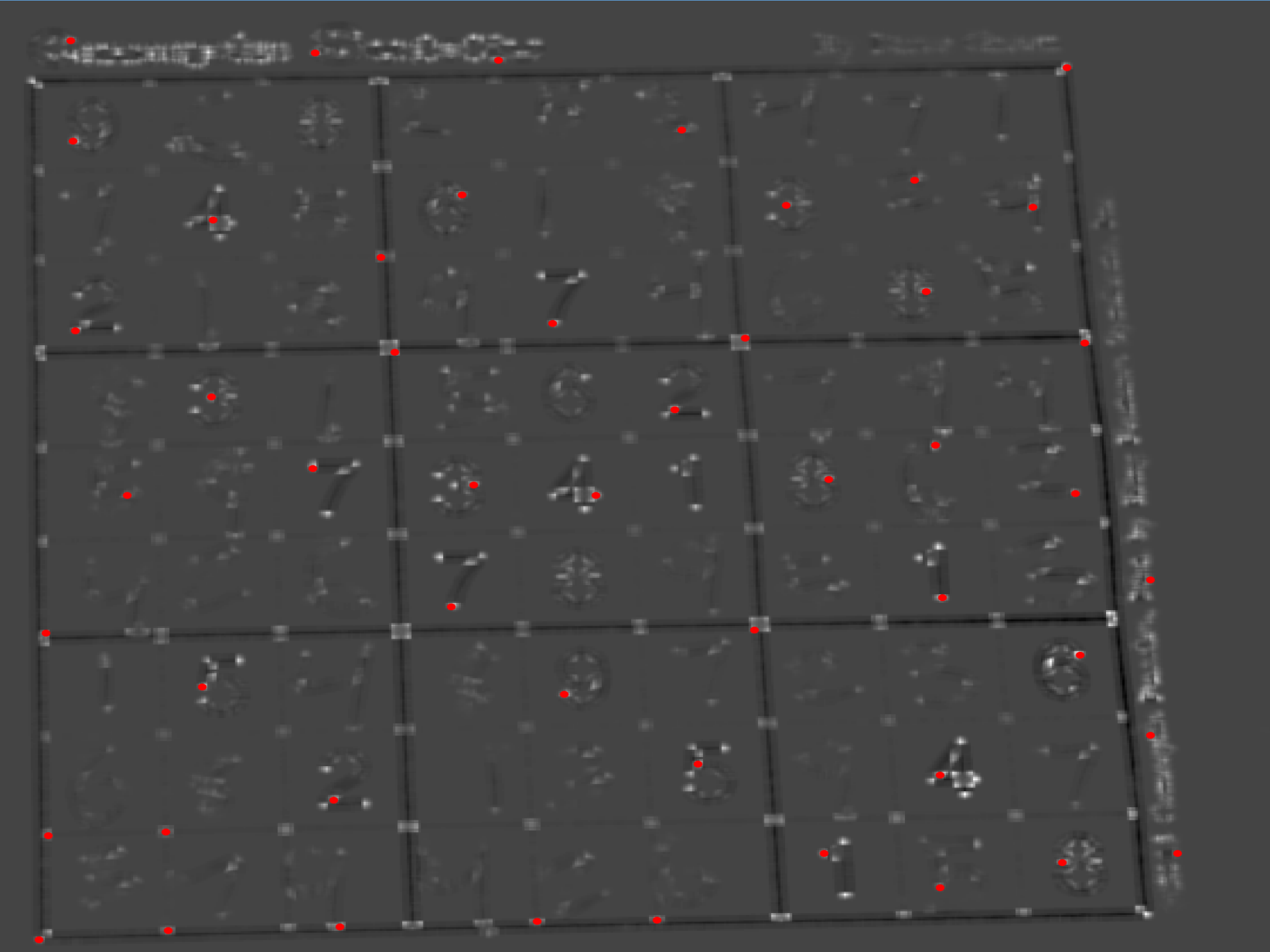
`Corners` is the output vector of corners in the form (X,Y).

Task : Corners

1. Open “Sudoku.jpg”;
2. Create a Harris corner image for this image;
3. Use Shi-Tomasi to find the location of the strongest corners.

Advanced Task:

Plot the Shi-Tomasi corners on the Harris image.



Contours

Contours

Both edges and corners are useful features for tracking, matching and other computer vision algorithms.

However, we often also want to know about the shapes in an image and select one that is of interest.

Morphology can be used to clean up extracted binary shapes but for larger-scale shape analysis, contours are more useful.

Contours

What is a contour?

A contour is a curve which outlines the boundary of a shape.

It is a set of continuous, connected points.

By investigating the parameters of the contour, we can find out about the shape it outlines.

This can help us find the object we want and eliminate the rest.

Contours

Extracting Contours

Since natural images have less distinct boundaries, it's often useful to extract edges or regions of interest before finding contours.

So contour extraction is usually done on binary images.

Once the contours are extracted, their features are examined to find the one that we want, e.g. the longest contour.

Contours

Cleaning the Contour

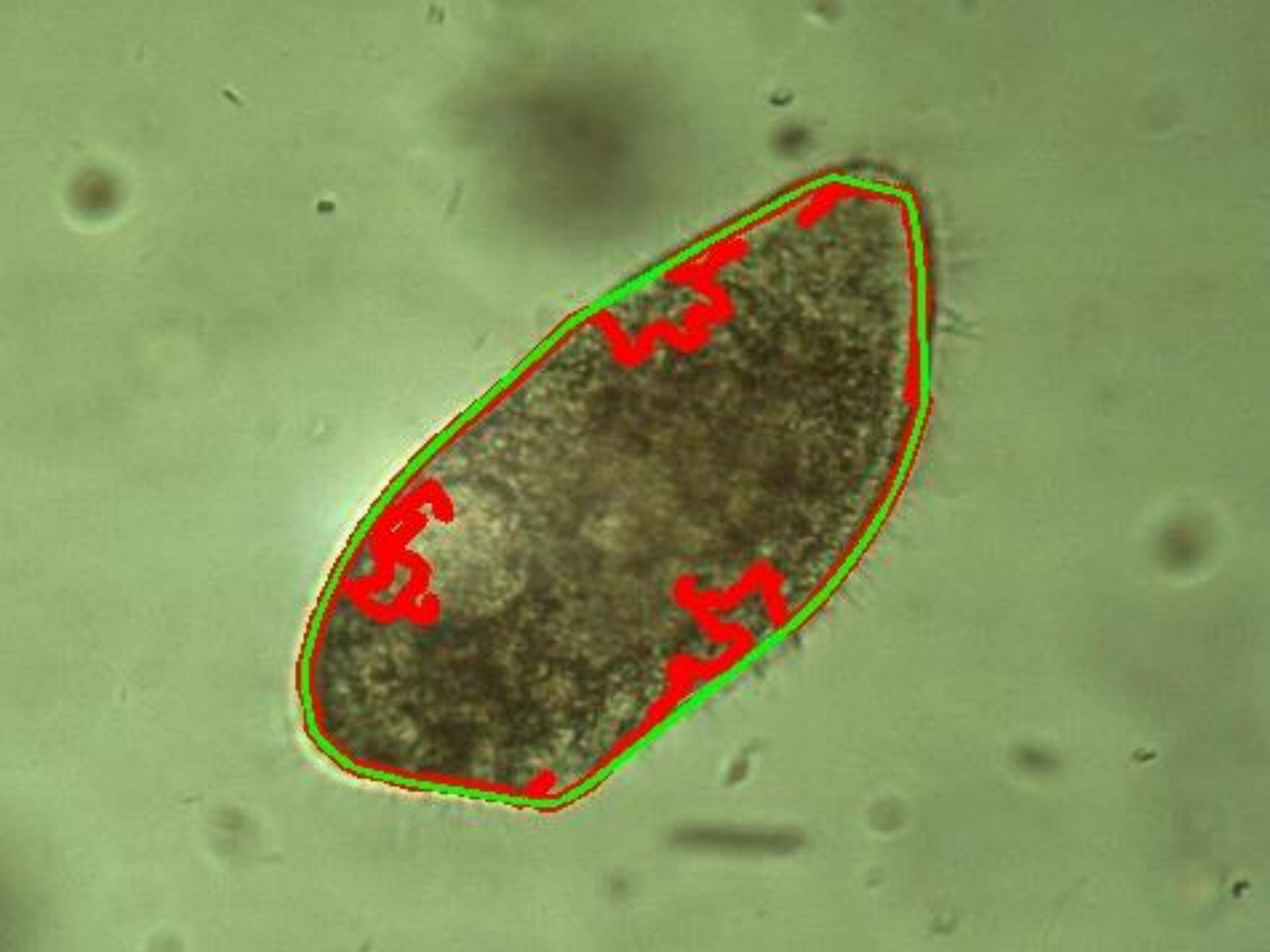
Often an extracted contour can be a bit squiggly.

In order to fix this, we can look at the *convex hull* of the contour.

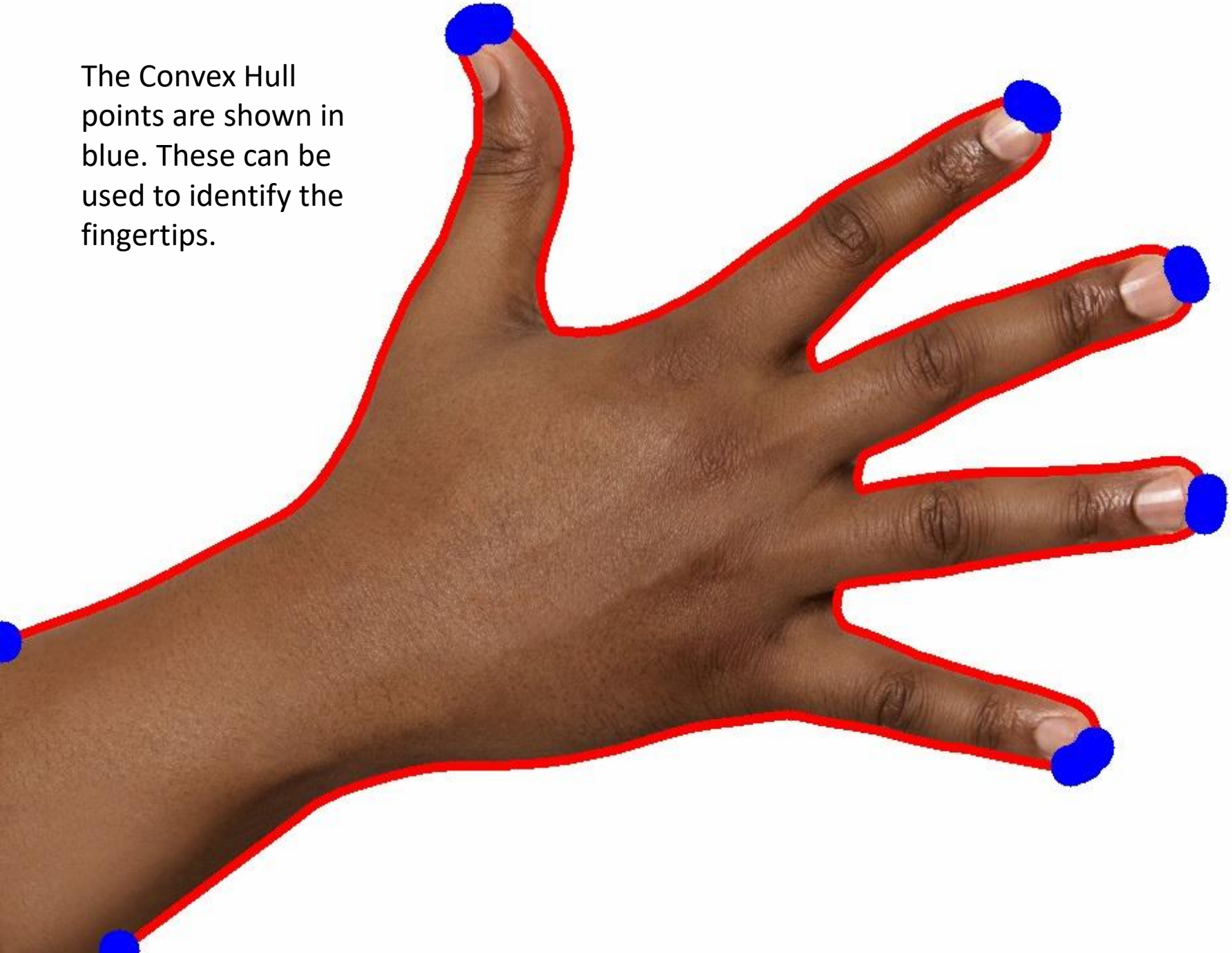
This is the outer part of the contour that never turns inwards.

This ensures that any interior contours are ignored.

This technique is useful with tricky shapes and is also often used in gesture analysis.



The Convex Hull points are shown in blue. These can be used to identify the fingertips.



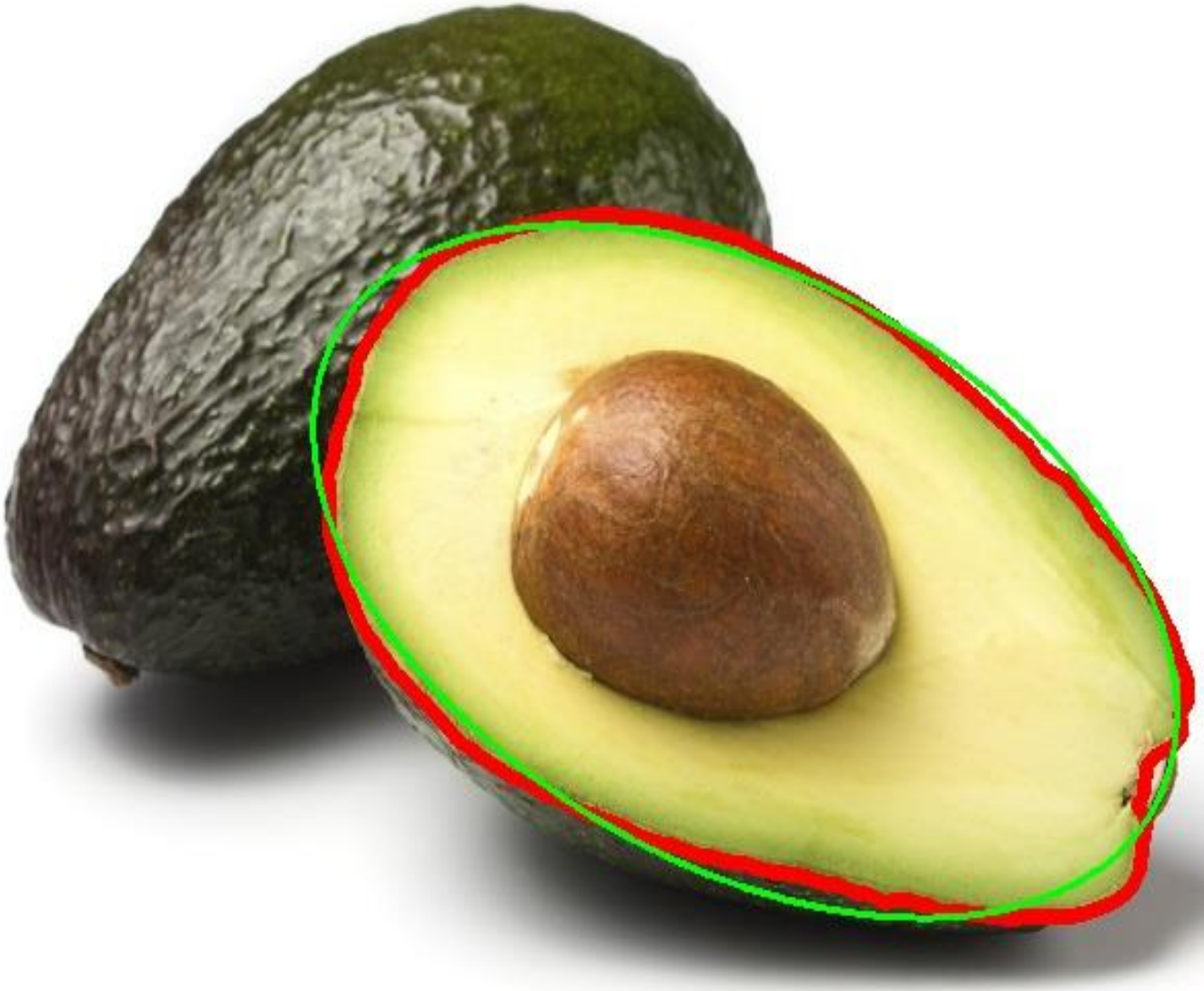
Contours

Bounding the Contour

Bounding the contour or fitting a known shape can help extract even more parameters.

The dimensions, ratios and orientation can be extracted this way.

There are many shape fitting algorithms available but the most useful are bounding rectangles and ellipses.



Contours

Contour Features

The purpose of contour extraction is to find features that describe the shape of an object.

To do this, we need to access the contour's features.

The main features of interest are the area and length but others including the orientation and centroid can also be found.

These can help identify the contour of interest and can tell us something about the shape of the object.

In Python with OpenCV

Find Contours

To extract the contours, we can use the *findContours* function:

```
contours,_ = cv2.findContours(B,
    mode=cv2.RETR_EXTERNAL,
  method=cv2.CHAIN_APPROX_NONE)
```

`B` is the original image (usually binary).

`contours` is the output vector of contours.

`mode` sets the output listing order for the contours.

`method` allows approximation of the contour by less points.

Draw Contours

To draw the contours, we can use the *drawContours* function:

```
I = cv2.drawContours(I, c, contourIdx=-1,
    color=(0,0,255), thickness=5)
```

`I` is the original image.

`c` is the contour or contours to be drawn.

`contourIdx` specifies which contour. If negative, all are drawn.

`color` and `thickness` set the drawing style.

Note : this function also alters the input image so keep a copy!

Convex Hull

To draw the contours, use the *convexHull* function. The hull points can be drawn as a contour but to connect them, use the *polylines* function:

```
hull = cv2.convexHull(c)
cv2.polylines(I, pts=hull, isClosed=True,
color=(0,255,255))
```

`c` is the contour of interest and `hull` is its convex hull.

`isClosed` specifies whether the contour is closed.

This code will connect the points of the convex hull of the contour `c` with a yellow line.

Bounding Rectangle

To bound the contour with a rectangle, use the *boundingRect* function.

This can then be drawn using the *rectangle* function:

```
x,y,w,h = cv2.boundingRect(c)
cv2.rectangle(I,(x,y),(x+w,y+h),(0,255,0),2)
```

`c` is the contour of interest.

`x,y,w,h` are the coordinates (`x,y`) of the top left corner and the width and height of the bounding rectangle respectively.

This code draws the bounding rectangle of the contour `c` in green.

Ellipse Fitting

To find the best-fit ellipse, use the *fitEllipse* function. This can then be drawn using the *ellipse* function:

```
ellipse = cv2.fitEllipse(c)
cv2.ellipse(I,ellipse, color=(0,255,0))
```

`c` is the contour of interest.

The `ellipse` object is the *rotated rectangle* that bounds the ellipse.

This code draws the best-fit ellipse of the contour `c` in green.

Contour Length

To find the length of the contour, use the *arcLength* function:

```
L = cv2.arcLength(c, closed=True)
```

`L` is the length in pixels of the contour `c`.

`closed` specifies whether the contour is closed.

The length feature is extremely useful in finding the object of interest.

Contour Area

To find the area of the contour, use the *contourArea* function:

```
A = cv2.contourArea(c)
```

`A` is the area in pixels of the contour `c`.

The area feature is also useful in finding the object of interest.

Sorting Contours

There is a *sorted* function in Python that can help order the contours in order of size (using either length or area):

```
contours = sorted(contours, key=cv2.contourArea, reverse=True)
```

`contours` is the list of contours extracted previously.

Now, `contours[0]` will represent the largest contour, `contours[1]` the next largest, etc.

Task : Contours

1. Open “Micro.jpg”;
2. Use either thresholding or edge detection to create a binary mask;
3. Extract and draw the boundary of the cell by finding the largest contour in the image.

Advanced Task:

Find the orientation of the cell by fitting an ellipse.


Angle = 44 o

Features

In this section you have learned about :

- Edges & Gradients
- Corners
- Contours

These topics were **implemented** and **tested** in Python with OpenCV.

Questions?