CORK INSTITUTE OF TECHNOLOGY
INSTITIÚID TEICNEOLAÍOCHTA CHORCAÍ

Computing@CIT

Machine Vision

Lecture 4: Feature extraction and matching

Types of features

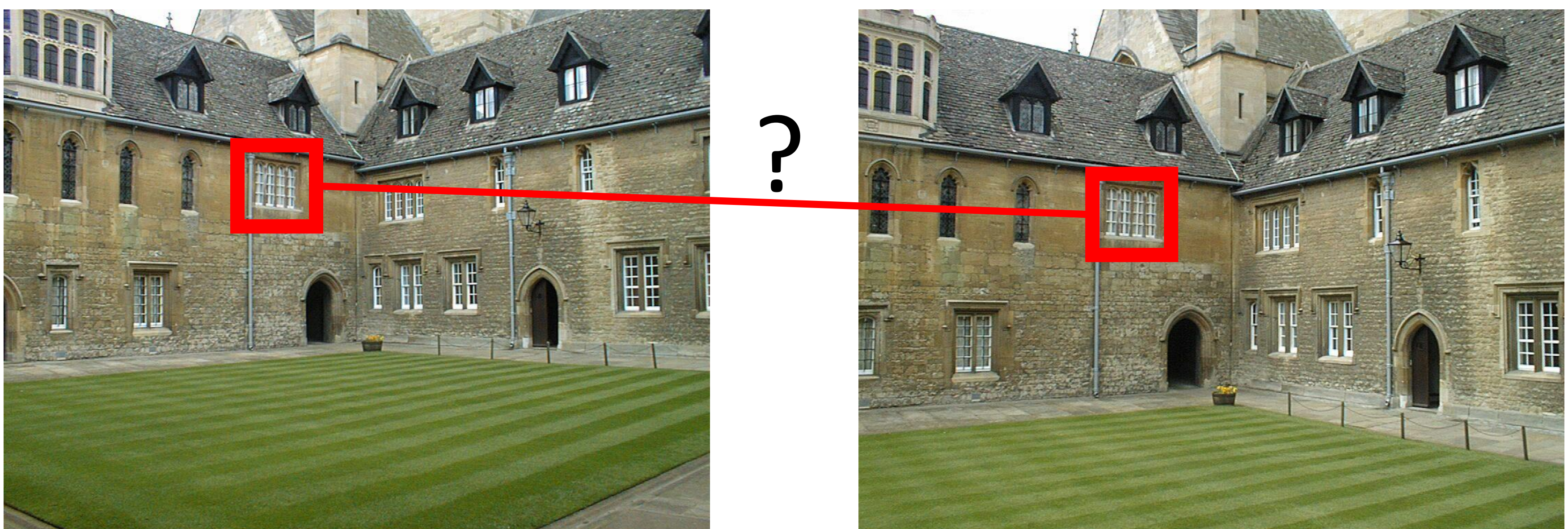
- Image features (or interest points) are <u>localisable regions</u> in the image, typically signifying something interesting in that location
- We can characterise them based on their extend

  - **Points** are 0-d locations in the image (x, y)
  - **Lines** are 1-d objects in the image $((x_1, y_1), \dots, (x_n, y_n))$
  - **Areas** are 2-d regions in the image

- We will focus on point features today, though the distinction between point and area features can be considered a matter of scale

Feature matching

- Assume that we have two images of the same object
- The task of feature matching is to find a feature detected in one image in the other image
- This is useful for identifying objects as well as for geometric scene reconstruction



Area-based matching

- The problem can be stated as trying to find a small patch in a larger image
- However, the other image will have undergone some geometric or radiometric transformation, therefore the problem is not as straightforward

5	9	3
4	1	8
4	1	7

?

3	4	8	2	3	4	6	9	0	1
4	6	0	6	8	6	2	4	2	2
1	1	6	3	9	2	1	2	5	7
7	4	3	4	3	4	1	5	2	4
5	3	2	6	8	5	9	3	1	3
3	5	2	3	5	4	1	8	2	5
1	2	5	1	2	4	1	7	6	2
2	7	3	6	4	3	5	6	3	1

Area-based matching

- If we know the geometric transformation, we can simply apply all possible transformations in a search pattern and try to find the best match
- In case of translation only we try all possible positions, but we can also try different rotations/sizes/etc.

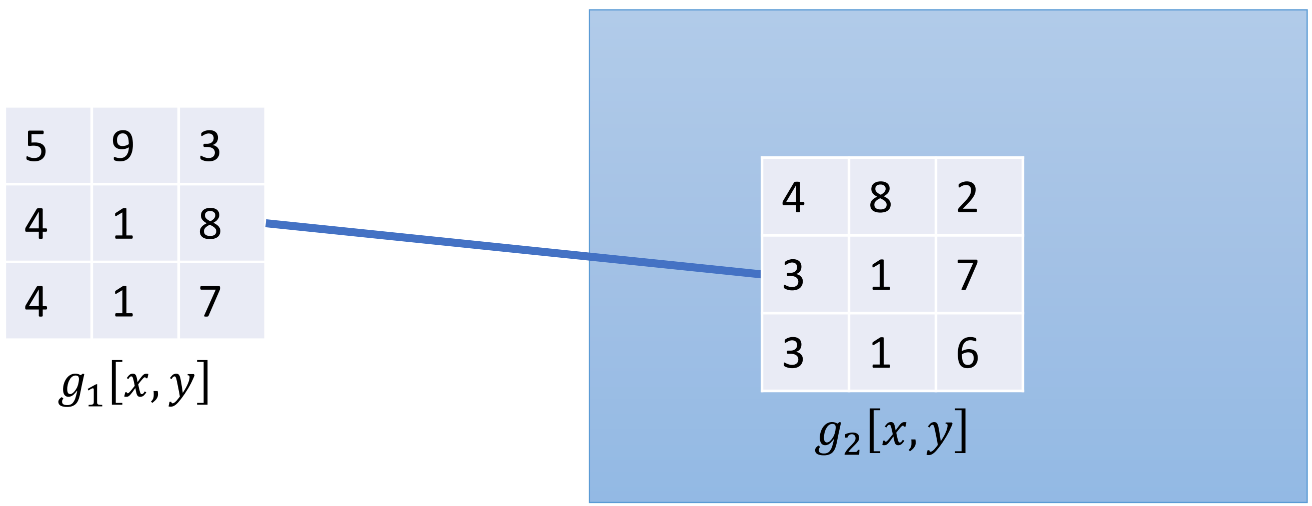
5	9	3
4	1	8
4	1	7

3	4	8	2	3	4	6	9	0	1
4	6	0	6	8	6	2	4	2	2
1	1	6	3	9	2	1	2	5	7
7	4	3	4	3	4	1	5	2	4
5	3	2	6	8	5	9	3	1	3
3	5	2	3	5	4	1	8	2	5
1	2	5	1	2	4	1	7	6	2
2	7	3	6	4	3	5	6	3	1

Area-based matching

- Regardless of the search pattern, which is determined by the expected geometric transformation, the problem boils down to comparing patches and determining if they are similar enough or not

- We are therefore looking for a metric to compare g_1 and g_2 that is invariant to the expected radiometric transformation



Cross-correlation

- If we expect the images to differ in brightness and contrast only then the **cross-correlation** is a useful metric of difference

$$\rho = \frac{\sum_{x,y}(g_1[x,y] - \mu_1)(g_2[x,y] - \mu_2)}{\sqrt{\sum_{x,y}(g_1[x,y] - \mu_1)^2}\sqrt{\sum_{x,y}(g_2[x,y] - \mu_2)^2}}$$

with

$$\mu_1 = \frac{1}{N}\sum_{x,y} g_1[x,y]$$

$$\mu_2 = \frac{1}{N}\sum_{x,y} g_2[x,y]$$

- It is invariant to linear histogram operations and normalised to $0 \leq \rho \leq 1$

Least-squares matching

- In case we expect other differences between the patches we can model them directly and look at a distance metric like this

$$\min_{\theta} \sum_{x,y} (g_1[x,y] - f[g_2; x, y, \theta])^2$$

- The purpose of f is to describe **invariants**, that we want the metric to be indifferent to
- There is always a trade-off between degrees of freedom and dissimilarity, as more free parameters mean everything will look more similar to each other and becomes less distinguishable

Feature localisation

- A possible criterion for characterising a good feature is that it is well localised within the image and can be matched with high accuracy to another image

- This means the feature has some local recognisable structure $g[x, y]$ that can be accurately matched

- We therefore assume that the image contains this recognisable structure at the feature position (x_0, y_0) perturbed by the additive noise $n[x, y]$

$$f[x, y] = g[x + x_0, y + y_0] + n[x, y]$$

- So that we can accurately determine the position

Feature localisation

- The following Taylor expansion of this equation gives some useful insight

$$f[x,y] = g[x+x_0, y+y_0] + n[x,y]$$

$$\approx g[x,y] + \left.\frac{\partial g}{\partial x}\right|_{x,y} x_0 + \left.\frac{\partial g}{\partial y}\right|_{x,y} y_0 + n[x,y\,]$$

- or for all $(x_1, y_1), \ldots, (x_n, y_n)$ in a local neighbourhood

$$(f[x_i, y_i] - g[x_i, y_i]) - n[x_i, y_i] \approx g_x[x_i, y_i]x_0 + g_y[x_i, y_i]y_0$$

- This position can be most accurately determined, if all brightness tangents in a local patch intersect in a single point (i.e. a corner)

Feature localisation

- We can stack all these into a vector

$$\underbrace{\begin{pmatrix} f[x_1, y_1] - g[x_1, y_1] \\ \vdots \\ f[x_n, y_n] - g[x_n, y_n] \end{pmatrix}}_{l} - \underbrace{\begin{pmatrix} n[x_1, y_1] \\ \vdots \\ n[x_n, y_n] \end{pmatrix}}_{n} \approx \underbrace{\begin{pmatrix} g_x[x_1, y_1] & g_y[x_1, y_1] \\ \vdots & \vdots \\ g_x[x_n, y_n] & g_y[x_n, y_n] \end{pmatrix}}_{A} \begin{pmatrix} x_0 \\ y_0 \end{pmatrix}$$

- The least-squares solution that minimises the noise $n^T n$ is

$$\begin{pmatrix} x_0 \\ y_0 \end{pmatrix} = (A^T A)^{-1} A^T l$$

- Note, that this estimate is more accurate than the pixel grid

Sub-pixel accuracy

- If we now assume that the noise vector has a covariance of $\sigma^2 I$ we can apply linear error propagation and obtain as covariance matrix for the unknown feature location

$$\Sigma = \frac{\sigma^2}{n-2}(A^T A)^{-1}$$

$$= \frac{\sigma^2}{n-2}\begin{pmatrix} \sum g_x^2 & \sum g_x g_y \\ \sum g_x g_y & \sum g_y^2 \end{pmatrix}^{-1}$$

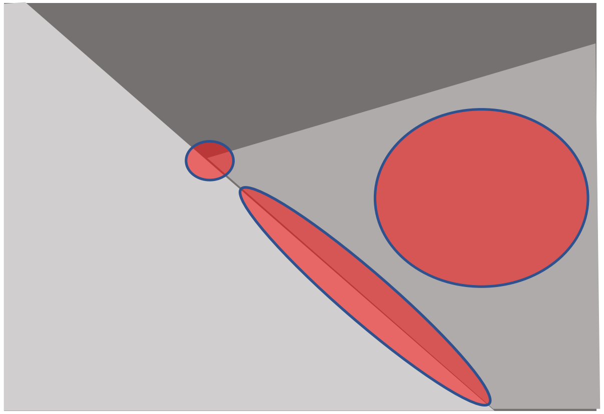
- Note, that this accuracy can be much better than the pixel grid, i.e. we are able to achieve significant sub-pixel accuracy given that the local patch is large enough and well shaped

Feature localisation

- In conclusion, the inverse **structure tensor**

$$\Sigma = \frac{\sigma^2}{n-2}\begin{pmatrix} \sum g_x^2 & \sum g_x g_y \\ \sum g_x g_y & \sum g_y^2 \end{pmatrix}^{-1}$$

- tells us how accurately we can determine the location of a recognisable structure in the image
- For a good point feature the shape of this accuracy ellipsoid should be small and round
- We can also use the shape to distinguish different types of features



Calculating the gradients

- We have already seen how to calculate image gradients through convolution with Derivatives of Gaussians

$$g_{x;\sigma_d} = I \otimes -\frac{x}{2\pi\sigma_d^4} \exp -\frac{x^2 + y^2}{2\sigma_d^2}$$

$$g_{y;\sigma_d} = I \otimes -\frac{y}{2\pi\sigma_d^4} \exp -\frac{x^2 + y^2}{2\sigma_d^2}$$

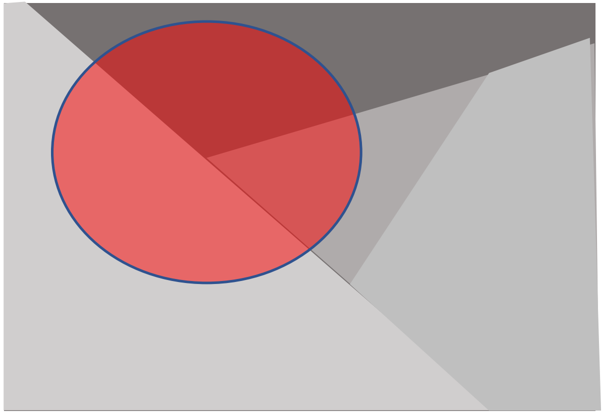
- Note that this operation depends on a scale parameter σ_d

Selecting the window size

- We have not yet discussed over which pixels the summation should be carried out
- Obviously it needs to be linked to the size of the feature we want to detect in the image, so that the radius does not overlap another feature

- To control the windows size over which the structure tensor is computed we can introduce a weight function that decreases with distance from the centre

$$w_{\sigma_w}[r] = \frac{1}{\sigma_w \sqrt{2\pi}} e^{-\frac{r^2}{2\sigma_w^2}}$$



- Here we use a scale parameter σ_w to control the window size

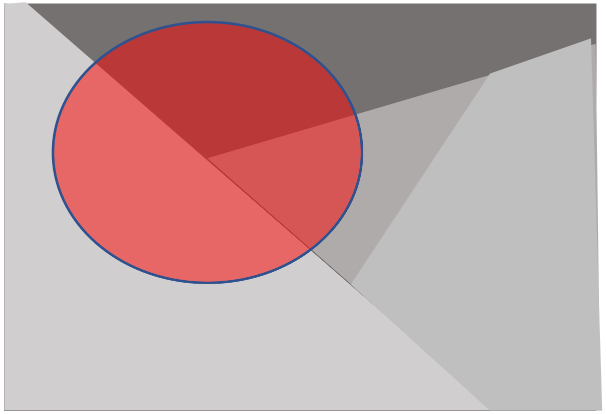
Selecting the window size

- Using the weight function we can calculate a weighted structure tensor for each point $p = (x, y)^T$ summing only over a $3\sigma_w$ neighbourhood p_i which is sufficiently non-zero as follows

$$S_{\sigma_w;\sigma_d}[p] = \sum_i w_{\sigma_w}[|p - p_i|] \begin{pmatrix} g^2_{x;\sigma_d}[p_i] & g_{x;\sigma_d}[p_i] g_{y;\sigma_d}[p_i] \\ g_{x;\sigma_d}[p_i] g_{y;\sigma_d}[p_i] & g^2_{y;\sigma_d}[p_i] \end{pmatrix}$$

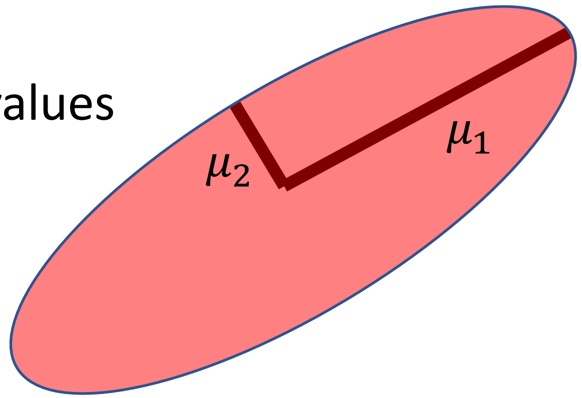
- Note how the structure tensor calculated this way depends on two scale parameters $\sigma_d < \sigma_w$

- σ_w determines the size of the patch, while σ_d determines the integration scale for the computation of the derivative



Interest points

- The shape of the covariance ellipsoid Σ can be characterised by the lengths of its axes, i.e. its eigenvalues
- The accuracy at point-features should be circular and small, therefore the eigenvalues should be small and equal $\mu_1 = \mu_2 \leq T$
- A line-feature is characterised by an elongated covariance ellipsoid, therefore one eigenvalue should be large and the other small, i.e. $\mu_1 \geq T_1$ and $\mu_2 \leq T_2$
- A homogeneous area is characterised by both eigenvalues being large, i.e. $\mu_1 \geq T$ and $\mu_2 \geq T$



Interest points

- Because the accuracy covariance is proportional to the inverse of the structure tensor $\Sigma \sim S^{-1}$ we can formulate these rules in terms of the eigenvalues of S: $\lambda_1 = \frac{1}{\mu_2}$ and $\lambda_2 = \frac{1}{\mu_1}$

- Different rules have been proposed, the Shi–Tomasi algorithm determines interest points based on where $\min(\lambda_1, \lambda_2)$ is large

- The computation of eigenvalues is costly, so we sometimes use the fact that

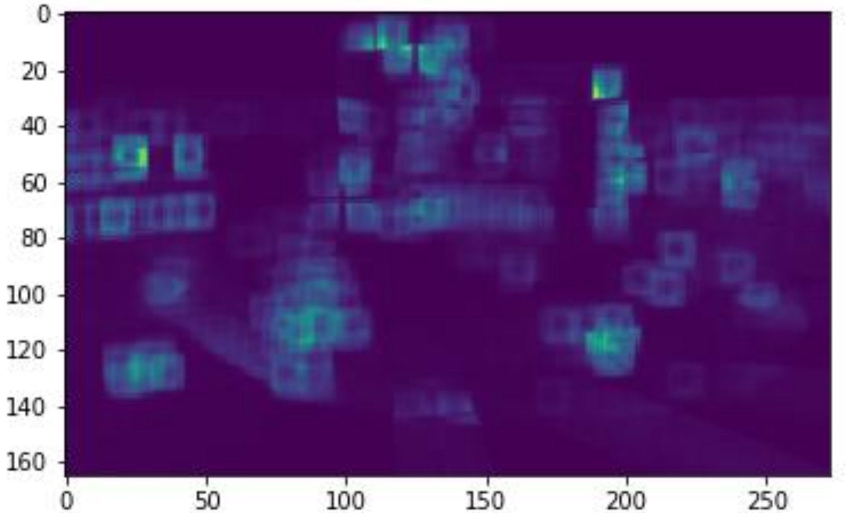
$$\det S = \lambda_1 \lambda_2$$
$$tr\, S = \lambda_1 + \lambda_2$$

- Harris & Stephens proposed as criterion to consider the maxima of

$$\mathrm{M} = \det S - \kappa\, tr^2 S$$

Non-maximum suppression

- We can compute a structure tensor for each pixel and derive a metric for how corner-like this pixel could be
- To extract a list of feature points all that remains to do is threshold the metric and filter pixels based on their neighbours to make sure every response is a local maximum

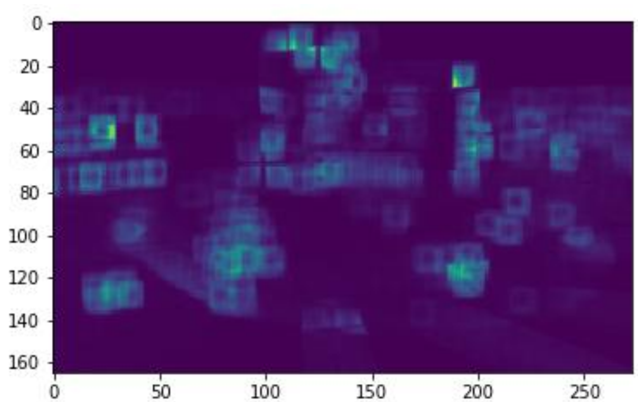


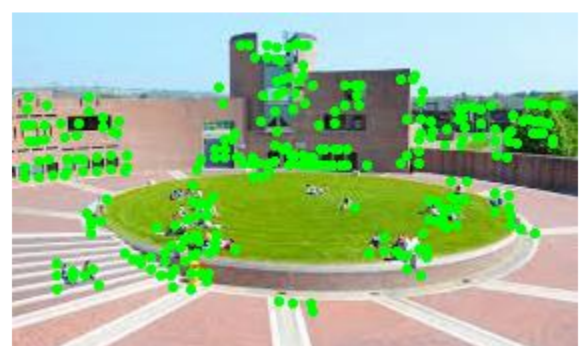
Non-maximum suppression

- We can compute a structure tensor for each pixel and derive a metric for how corner-like this pixel could be
- To extract a list of feature points all that remains to do is threshold the metric and filter pixels based on their neighbours to make sure every response is a local maximum

7	5	9
9	10	4
7	7	2

11	5	9
9	10	4
7	7	2





Discrete implementation

- In the previous derivation we need to calculate the weighted structure tensor for each pixel
- This involves five convolutions for every pair (σ_w, σ_d) of scale parameters we want to consider

- While this approach yields the most accurate results, we can approximate some of the computations using smaller and discrete convolution kernels
- Instead of using derivative of Gaussian kernels to calculate g_x and g_y we can approximate these calculations by using for example

$$g_x = I \otimes (+1 \quad 0 \quad -1)$$
$$g_y = I \otimes (+1 \quad 0 \quad -1)^T$$

- This fixes the scale σ_d to the resolution of the image

Discrete implementation

- Also, in order to calculate the structure tensor instead of using a continuous weighting function we can simplify this by summing over a small neighbourhood patch of size $(2s, 2s)$ as follows

$$S = \sum_{x=-s}^{s} \sum_{y=-s}^{s} \begin{pmatrix} g_x^2[x,y] & g_x[x,y]g_y[x,y] \\ g_x[x,y]g_y[x,y] & g_y^2[x,y] \end{pmatrix}$$

- Now we can control the scale σ_w with the resolution of the image

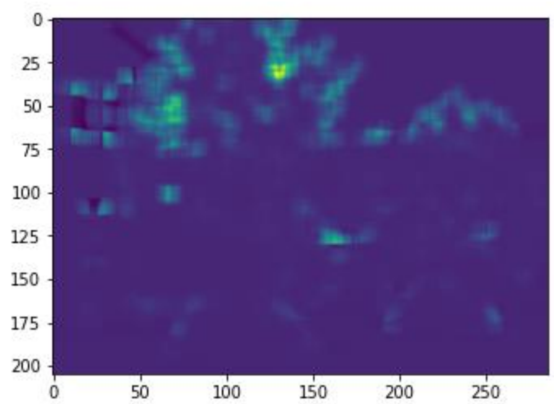
- In conclusion, we can fix the relative ratio between σ_w and σ_d with the patch size parameter s and the overall scale with the resolution of the image
- Note, that this approximation is only using integers in the computation

Interest points

```
harris = cv2.cornerHarris(img, 10, 3, 0.04)
shi = cv2.cornerMinEigenVal(img, 10)
```

Window size

Derivative kernel size



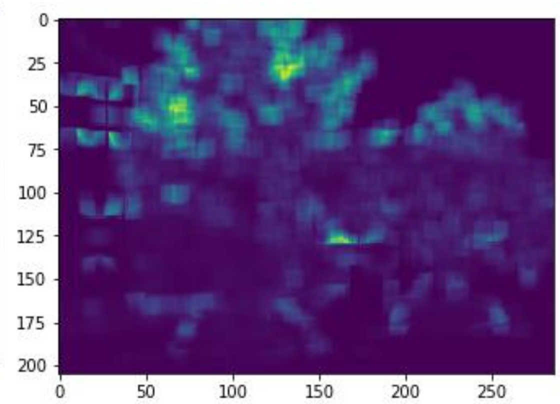


Image pyramids

- For operating on different scales it is practical to pre-process the image
- We have seen how scales can be linked to resolution, therefore we typically compute **image pyramids** to work with multiple scales
- An image pyramid is a collection of down-sampled versions of the original input image
- To compute an image pyramid we apply a low-pass filter and sub-sample at a lower resolution, iterating until the lowest resolution is reached

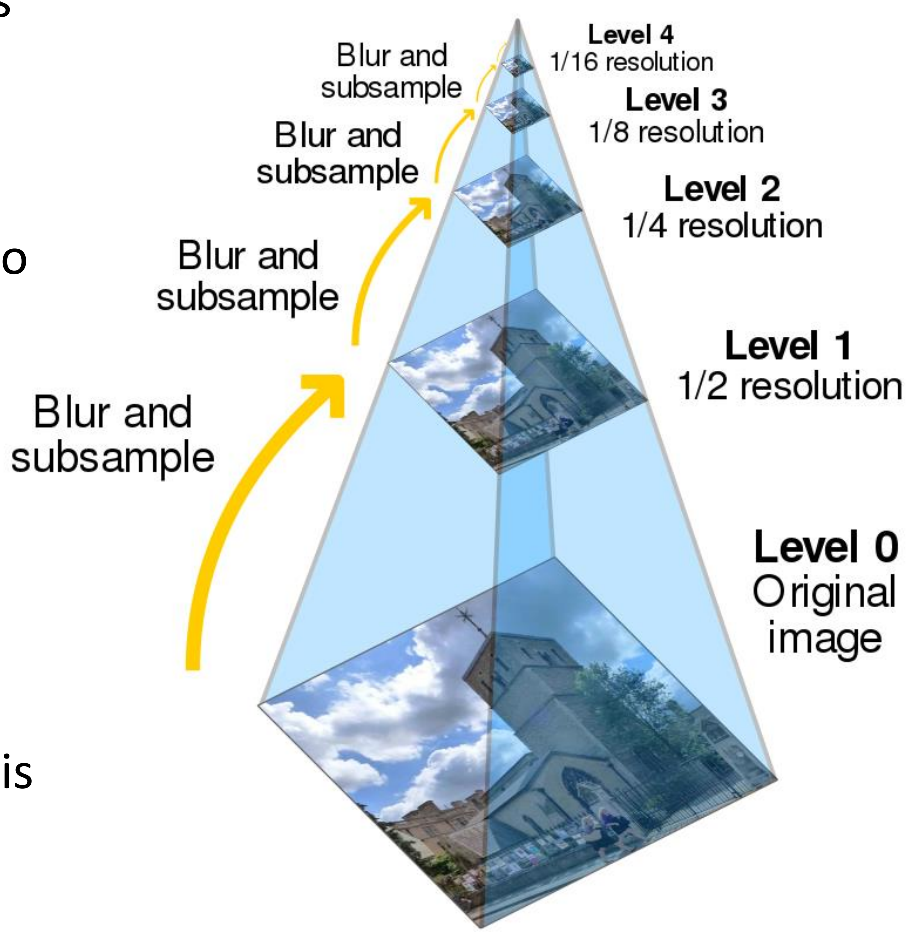
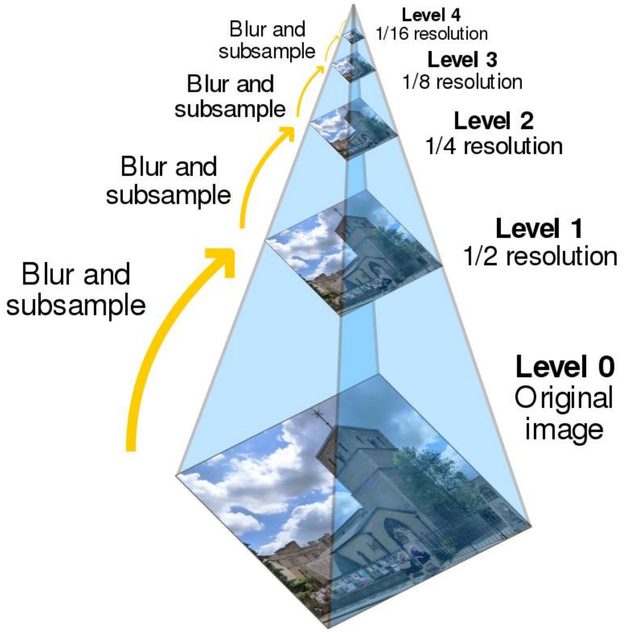


Image pyramids

```
pyramid = {0:img}
for i in range(5):
    pyramid[i+1] = cv2.pyrDown(pyramid[i])
```

Scale space



- We can consider scale as an additional dimension

- Using a Gaussian smoothing kernel

$$g[x, y; \sigma] = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}$$

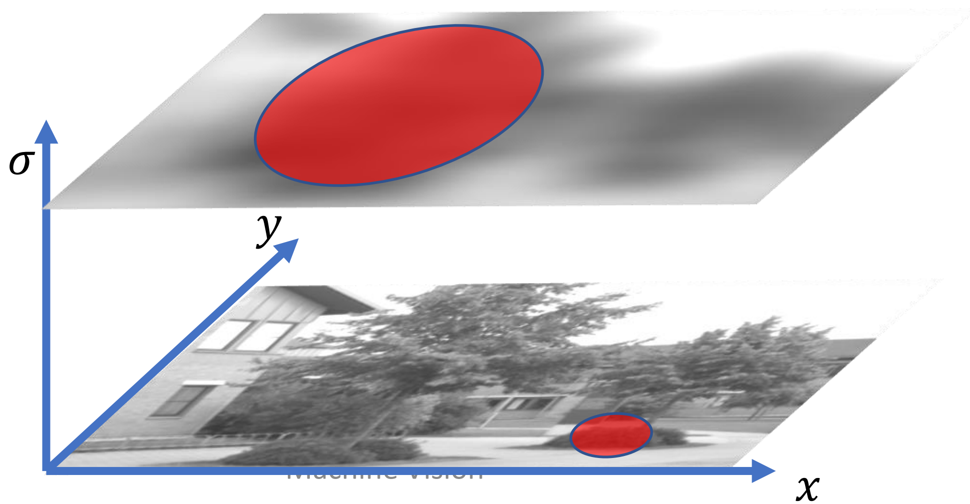
- The scale space representation of the image $I[x, y]$ is

$$L[x, y, \sigma] = (g[\cdot,\cdot; \sigma] \otimes I[\cdot,\cdot])[x, y]$$

- Note, that when using an image pyramid to store a scale-space representation we need to be careful what pixels align between scale layers

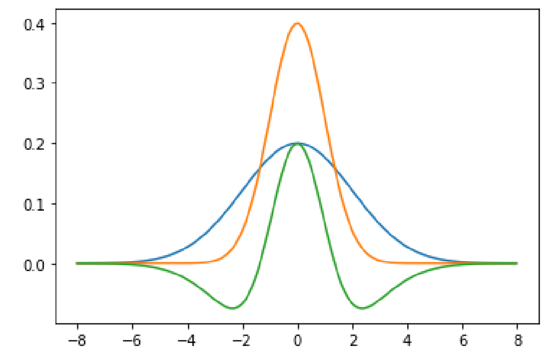
Scale-invariant feature detectors

- So far we have fixed the scale parameter and derived interest point detectors operating at a given input scale
- Scale-invariant feature detectors search for locations of interest points not only in the image (x, y) plane for given scale parameters, but rather look for locations (x, y, σ) in the 3-d scale-space representation
- The result is again a list of features, but now every feature also comes with a specific scale t at which it was most prominent



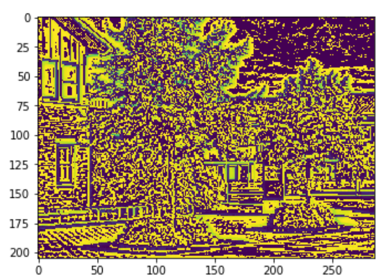
Difference of Gaussians

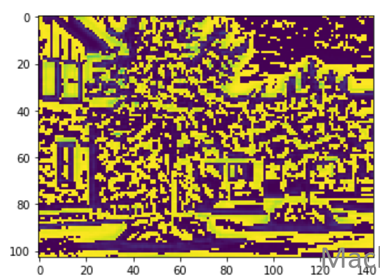
- The **Scale-Invariant Feature Transform (SIFT)** algorithm uses Difference of Gaussians (DoG) (not derivative!!!) in scale space as criterion
- A DoG is defined as the difference image between two adjacent parallel layers in scale space

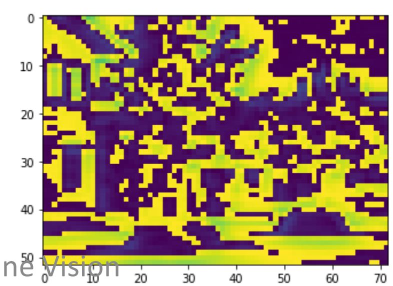


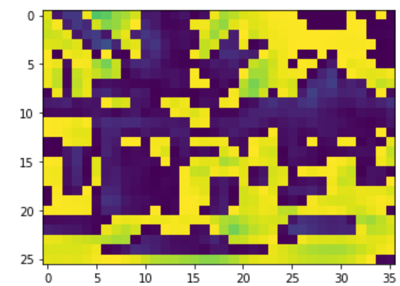
$$D[x, y, i] = L[x, y, k_i\sigma] - L[x, y, k_{i+1}\sigma]$$

- It is a discrete approximation of the Laplacian of the Gaussian, which acts as a band-pass filter







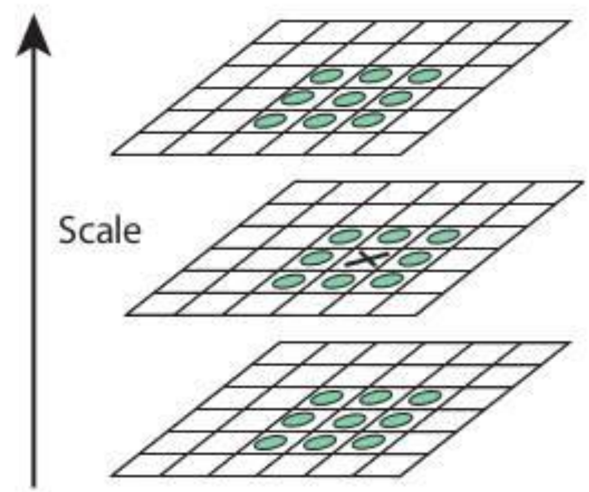


Non-maximum suppression in scale space

- Interest points in scale space are then extracted by finding local maxima in the 3 dimensional cube of DoGs:

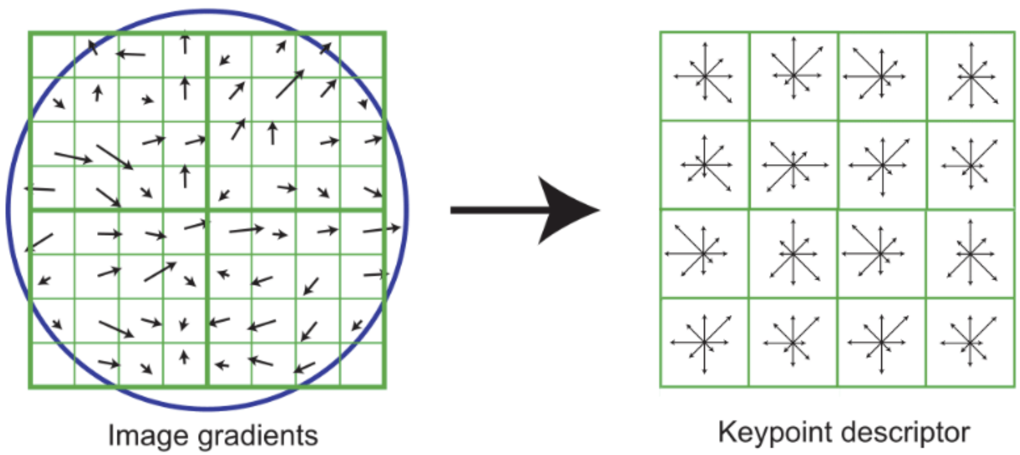
$$D[x, y, i] = L[x, y, k_i\sigma] - L[x, y, k_{i+1}\sigma]$$

- Resulting in a list of feature points together with their size



Feature descriptors

- The SIFT algorithm also proposes a feature descriptor
- The basic idea is to use **gradient histograms** to create a vector that describes the local patch normalised by scale and rotation
- The patch is partitioned into fields and the gradient orientations within the field are discretised and counted



- Every feature is then associated with a descriptor vector, which can be efficiently processed

SIFT in OpenCV

- The SIFT algorithm is patented and requires to re-compile OpenCV with the following option

OPENCV_ENABLE_NONFREE

```
sift = cv2.xfeatures2d.SIFT_create()
keypoints, descriptors = sift.detectAndCompute(img, None)
```

Canny edge detector

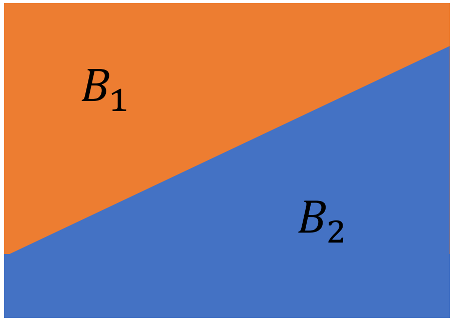
- So far we have focused on the extraction of point features

- We could also look at patches where there is a dominant gradient direction, i.e. where one eigenvalue of the structure tensor is significantly larger than the other $\lambda_1 \gg \lambda_2$

- We already saw that at such a brightness edge the gradients are

$$g_x = \sin\theta (B_2 - B_1)\delta[x \sin\theta + y \cos\theta + \rho]$$

$$g_y = -\cos\theta (B_2 - B_1)\delta[x \sin\theta + y \cos\theta + \rho]$$



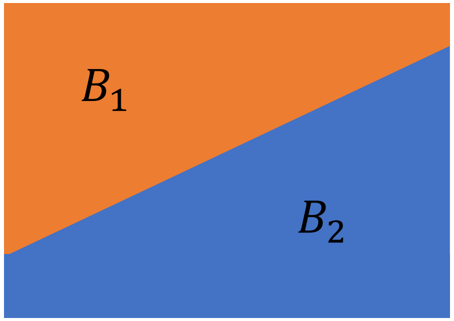
Canny edge detector

- From this follows that the strength of the edge can be calculated as

$$M = \sqrt{g_x^2 + g_y^2} = |B_2 - B_1|\delta[x \sin\theta + y \cos\theta + \rho]$$

- Finding edges can then be accomplished by thresholding $M \geq T$
- At such edges we can also calculate the orientation as

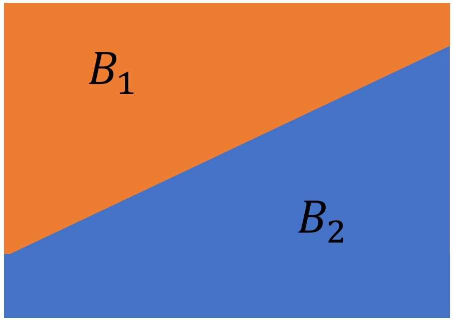
$$\theta = \tan^{-1}\frac{g_x}{g_y}$$



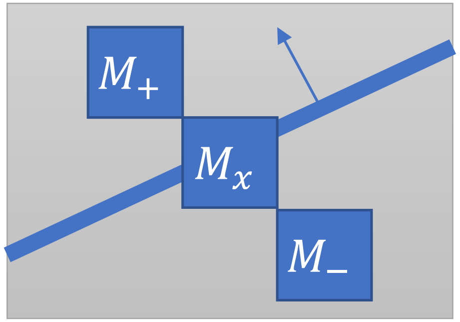
Canny edge detector

- The orientation can be used to perform non-maximum suppression on the edge strength $M[x,y]$ orthogonal to the edge (instead of in all directions, as with the point features)
- We therefore retain as edges only those above threshold T_1 for which the neighbours orthogonal to the edge are below a second threshold

$$M_x > T_1$$
$$M_+ < T_2$$
$$M_- < T_2$$



$I[x,y]$



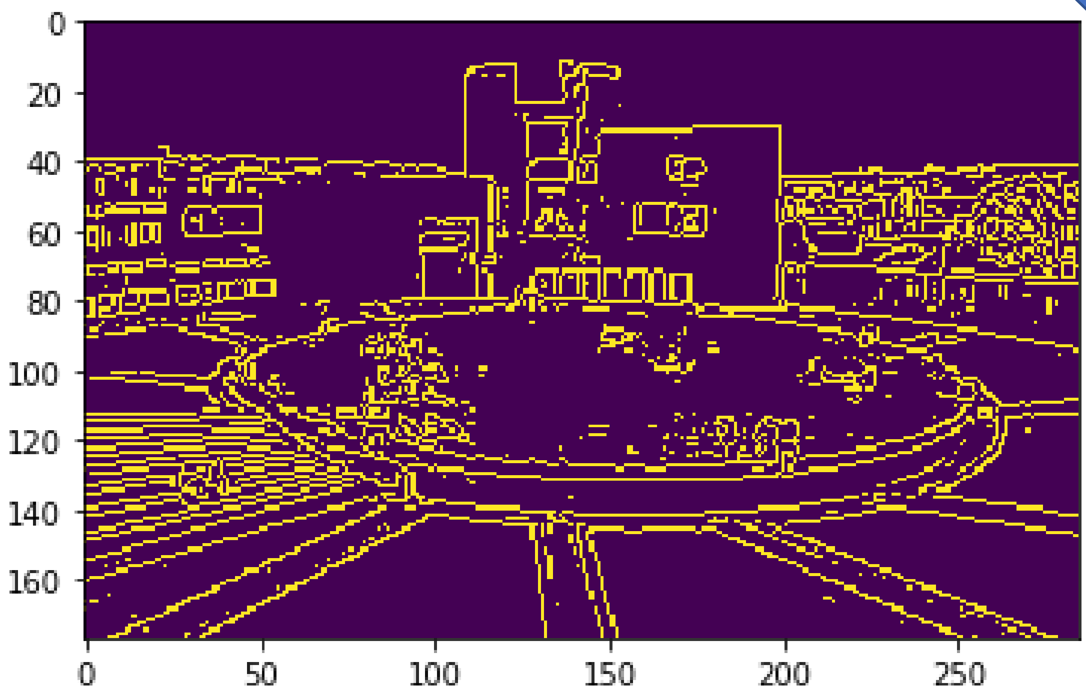
$M[x,y]$

Canny edge detector

```
canny = cv2.Canny(img, 100, 100)
```

Second threshold for non-maxima suppression

1st threshold on edge strength



Hough transformation

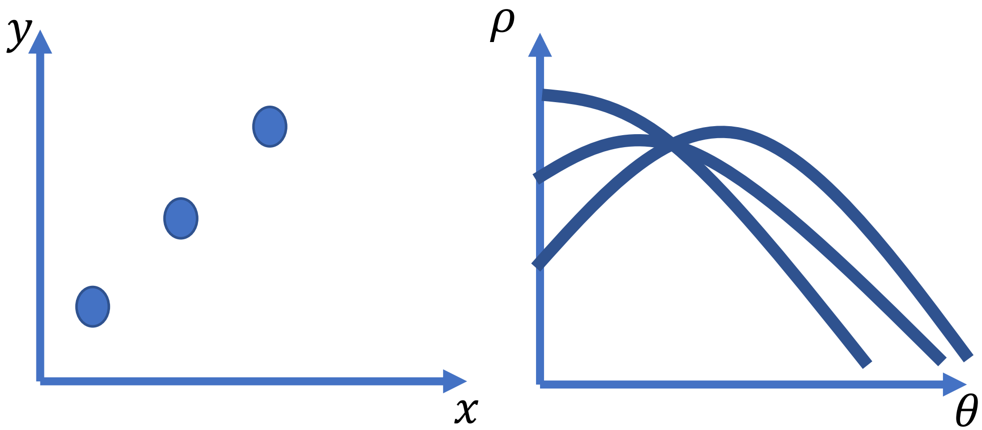
- While the previous approach provides all points that are on edges, it is sometimes useful to group those together that form geometric primitives (i.e. lines, circles, etc.)
- We will see now how to extract straight line segments from the image, but the approach can be applied to other primitives as well
- The first step is to define a parameterised model that all points on the desired primitive should adhere to; in the case of straight lines that could be

$$\forall x, y:\ x\sin\theta + y\cos\theta + \rho = 0$$

- With the line parameters θ and ρ

Hough transformation

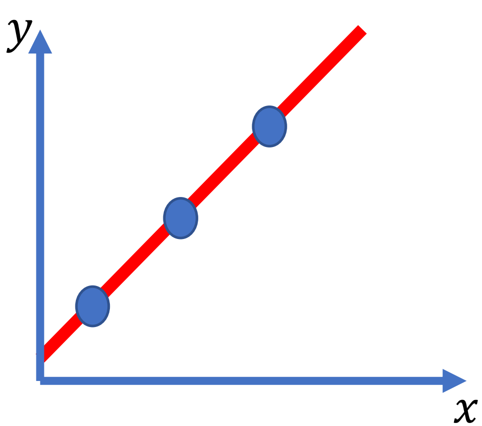
- The idea of the Hough transformation is now to go through the edge image, and for every detected edge pixel plot all the possible parameters of lines θ, ρ passing through this edge pixel

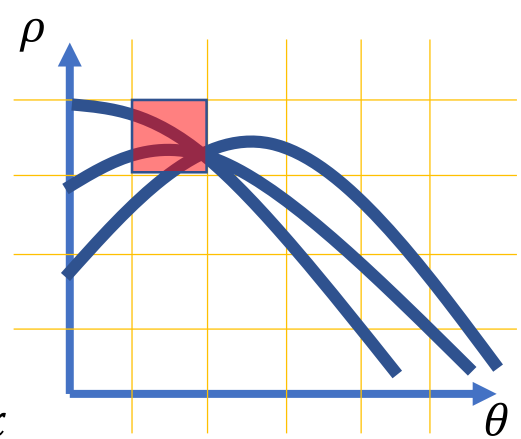


- We observe, that the plots intersect in a single point for all points that are on a straight line

Hough transformation

- We now discretise the Hough space and let each edge point vote for all lines that pass through it

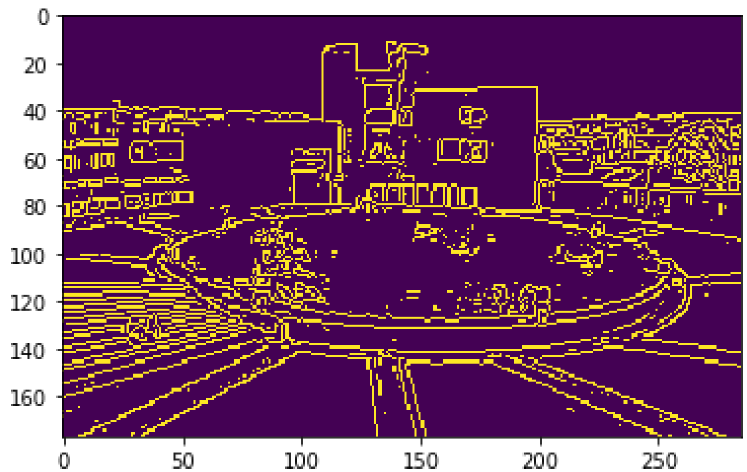


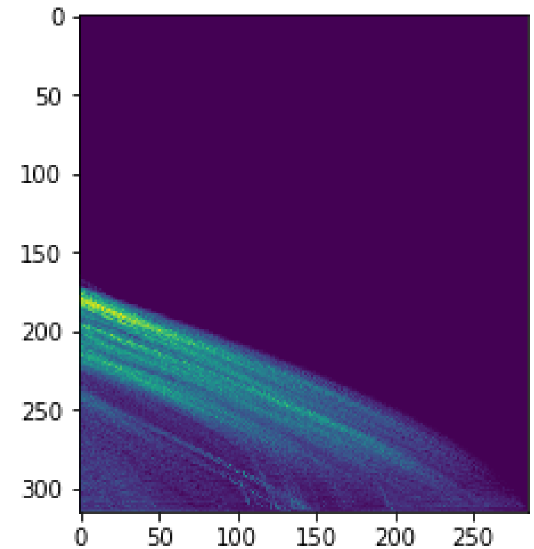


0	0	0	0	0	0
2	3	**3**	1	0	0
1	1	2	2	1	0
0	0	0	1	2	1
0	0	0	0	1	2

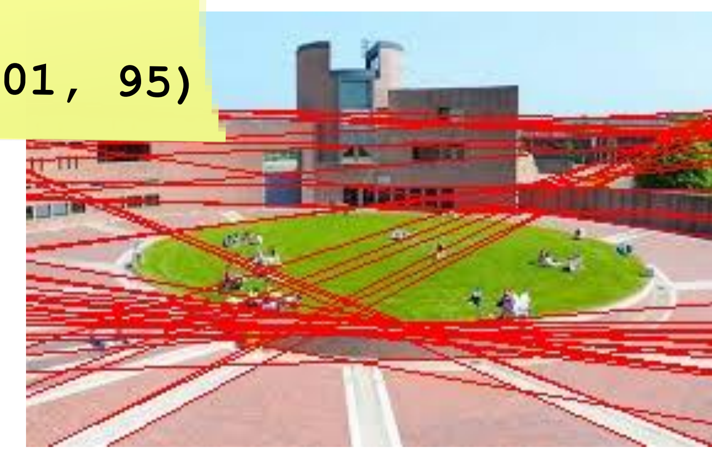
- The maximum values in Hough space the correspond to line features in the image

Hough transformation






```
lines = cv2.HoughLines(canny, 1, 0.01, 95)
```



Markers

- Feature detection and matching is inherently difficult, therefore markers are used in most industrial applications
- Controlling the environment is typically key to being able to apply machine vision methods

Markers

- Feature points can be detected with sub-pixel accuracy
- Markers are designed, to make sure that optimal sub-pixel accuracy is achieved by using shapes with a defined centre and sharp edges
- They also typically contain a code that can be directly processed, thereby avoiding error-prone feature matching

Thank you for your attention!