COMP3005/Computer Vision

6

Local interest points

Summary

The detection of local interest points that are stable under varying imaging conditions has a huge number of applications in computer vision (we'll see these in future lectures). Research in this area goes back as far as the 1960s and 70s. The Harris and Stephens corner detection technique developed in 1988 is a classic example of a detection technique with impressive robustness. A related problem to the detection of interest points is the problem of scale (the size at which an object appears in an image). Scale space theory allows interest point techniques to be developed that are invariant to changes in scale (i.e. the object moving further away). The Difference-of-Gaussian blob detector is an example of such a scale-space blob detection technique.

Key points

Finding stable points: repeatability and robustness

- Finding stable interest points is a key problem in computer vision, with many applications (more on this in the next few lectures!).
- What makes a good local interest point?
    - Invariance to brightness changes
    - Sufficient texture variation in the local neighbourhood
    - Not too much texture variation in the local neighbourhood
    - Invariance to changes in relative angle between camera and object
- There are a number of types of stable points we can find, but we'll just focus on two: Corners and Blobs.

The Harris Corner Detector

- The Harris & Stephens corner detector is considered to be one of the *classic* algorithms in computer vision.
    - Developed just up the road at Roke Manor in Romsey!
- Simple idea:
    - Consider the brightness of a small patch of the image (i.e. the sum of pixel values)
    - Now consider slightly shifting that patch in all directions
        - if the brightness remains the same, then the original patch is not a stable point
        - if the brightness changes by a large amount in all directions, then the original patch is stable.
    - We can write a mathematical function that computes the weighted sum-squared difference of the window and shifted window. Given a point (x, y) and shift $(\Delta x, \Delta y)$ this can be written as:

$$E(x,y) = \sum_{W} f(x_i, y_i)[I(x_i, y_i) - I(x_i + \Delta x, y_i + \Delta y)]^2$$

      where $I(\bullet,\bullet)$ denotes the image and x_i and y_i are all the points in the window W centred at (x, y). $f(\bullet,\bullet)$ is called the window function, and it just allows the respective weights of pixels to be changed depending on distance from the centre point (x, y). For purposes of discussion we'll assume that $f(\bullet,\bullet)$ is always 1 and drop it from further working, however in practice we use a Gaussian response function to weight pixels nearer the centre higher.

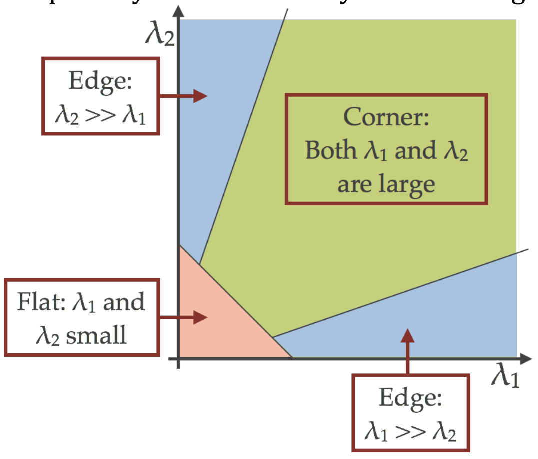
- The term corresponding to the shifted window in the above difference equation can be approximated using the first order terms of a Taylor expansion:

$$I(x_i + \Delta x, y_i + \Delta y) \approx I(x_i, y_i) + [I_x(x_i, y_i) \quad I_y(x_i, y_i)] \begin{bmatrix} \Delta x \\ \Delta y \end{bmatrix}$$

  where $I_x(\bullet,\bullet)$ and $I_y(\bullet,\bullet)$ denote the partial derivatives in x and y, respectively.
- Substituting this expansion in the difference equation and simplifying leads to the following:

$$\begin{aligned}
E(x,y) &= \sum_W [I(x_i, y_i) - I(x_i + \Delta x, y_i + \Delta y)]^2 \\
&= \sum_W \left(I(x_i, y_i) - I(x_i, y_i) - [I_x(x_i, y_i) \quad I_y(x_i, y_i)] \begin{bmatrix} \Delta x \\ \Delta y \end{bmatrix} \right)^2 \\
&= \sum_W \left(-[I_x(x_i, y_i) \quad I_y(x_i, y_i)] \begin{bmatrix} \Delta x \\ \Delta y \end{bmatrix} \right)^2 \\
&= \sum_W \left([I_x(x_i, y_i) \quad I_y(x_i, y_i)] \begin{bmatrix} \Delta x \\ \Delta y \end{bmatrix} \right)^2 \\
&= [\Delta x \quad \Delta y] \begin{bmatrix} \sum_W \left(I_x(x_i, y_i)\right)^2 & \sum_W I_x(x_i, y_i) I_y(x_i, y_i) \\ \sum_W I_x(x_i, y_i) I_y(x_i, y_i) & \sum_W \left(I_y(x_i, y_i)\right)^2 \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta y \end{bmatrix} \\
&= [\Delta x \quad \Delta y] \mathbf{M} \begin{bmatrix} \Delta x \\ \Delta y \end{bmatrix}
\end{aligned}$$

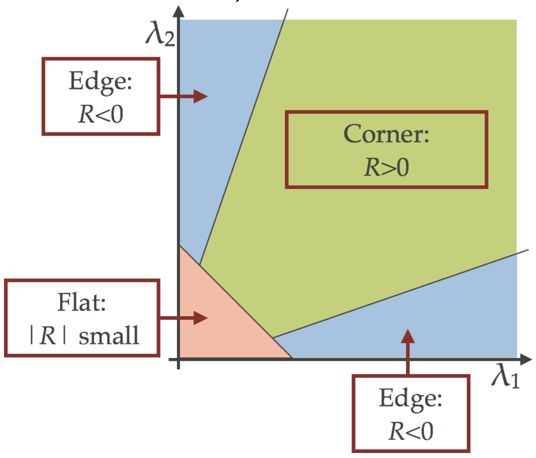
  - The square symmetric matrix **M** concisely describes the *shape* of the local weighted difference function.
  - It's basically encoding the image derivatives in the *x*, *y* and *xy* directions. It's actually the second moment matrix, more commonly referred to as the *structure tensor.*
- As with the covariance matrices we explored in Lecture 3, the eigenvectors and eigenvalues of this matrix tell us explicitly about the principle directions and principle rates of change.
  - We actually only need to consider the eigenvalues λ_1 and λ_2:
    - If λ_1 and λ_2 are both small, there is little change in $E(x, y)$ in any direction, and the windowed region must have approximately constant intensity.
    - If one eigenvalue is high, and the other is low, then the window must be on an edge in the image (moving the window along the edge, in the direction of the eigenvector corresponding to the smallest eigenvalue would have little effect, but moving orthogonal to that direction would have large effect).
    - If both eigenvalues are large, then a shift in any direction would give a large change in $E(x, y)$. This would indicate a corner.
  - Graphically this is shown by the following diagram:



- Rather than using the absolute values of the eigenvalues directly, Harris and Stephens came up with a scheme that avoids explicitly computing the Eigendecomposition by formulating a corner response function ($R(x, y)$in terms of the determinant and trace of **M**:

$$\det(\mathbf{M}) = M_{00}M_{11} - M_{01}M_{10} = M_{00}M_{11} - M_{10}^2 = \lambda_1\lambda_2$$
$$\mathrm{trace}(\mathbf{M}) = M_{00} + M_{11} = \lambda_1 + \lambda_2$$
$$R = \det(\mathbf{M}) - k\ \mathrm{trace}(\mathbf{M})^2$$

  where k is an empirically set small constant (usually 0.04...0.06).
- Interestingly this response function defines exactly the same space corner/edge/flat that we saw above, as shown in the following diagram:

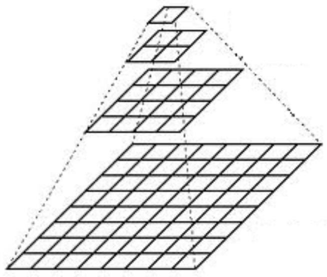


- Finally to actually find corners, you compute the corner response function R for each pixel, and keep only those over a predetermined threshold. You then filter out points that are not local maxima of R within a small window (usually just the 8 neighbouring pixels).
  - Note: in terms of implementation, you don't compute the partial derivatives for each window separately, but rather compute the derivatives on the whole image first (e.g. by convolving with Sobel, or performing finite differences). You can then use convolution across the gradient images to efficiently compute the weighted summed window response. After doing this, computation of the structure tensor at a given position is trivial!

Scale in computer vision

- In some of the demos from previous lectures, we've seen the effect of scale. Put simply, if an object moves closer to the camera it appears larger, and in more detail. As it moves further away it gets smaller and has less detail.
  - For algorithms that use a fixed size window (i.e. Harris and Stephens, or local thresholding), this can pose bit of a problem if the object is likely to appear at different distances.
  - **Scale space theory** is a formal framework for handling images at different scales by representing an image as a family of smoothed images parameterized by the size of the smoothing kernel used for suppressing fine detail.
    - The single parameter t of this family is referred to as the *scale parameter*, with the interpretation that image structures of spatial size smaller than about $\sqrt{t}$ have largely been smoothed away in the scale-space level at scale t.
  - **A Gaussian scale space** is the main type of scale space you'll encounter. Unsurprisingly the smoothing function is the Gaussian kernel.
    - The details of why this is the case are not important for this course, but needless to say it is provable that the Gaussian scale space has the desired properties (the "scale space axioms") for image representation.

- Formally the Gaussian scale space of an image $f(x, y)$ is the family of derived signals $L(x, y; t)$ defined by the convolution of the image with the 2d Gaussian kernel:
$$g(x, y; t) = \frac{1}{2\pi t} e^{-(x^2+y^2)/2t}$$
such that
$$L(\cdot,\cdot; t) = g(\cdot,\cdot; t) * f(\cdot,\cdot),$$
where the semicolon in the argument of L implies that the convolution is performed only over the variables x, y, while the scale parameter t after the semicolon is just indicating the scale level.
    - This definition is valid for all $t \geq 0$, however in practice only a finite set of discrete levels of t is usually considered.
        - Common to use integer powers of 2 or $\sqrt{2}$.
    - Also note $L(x, y, 0) = f(x, y)$
- If you double the scale then you effectively halve the resolution, so without loss of information you can subsample the image by a factor of 2 in each direction (this is a direct result of Nyquist-Shannon sampling theorem).
    - This leads to a "*Gaussian Pyramid*" representation, which has computational advantages, as images with higher scales are smaller and thus faster to process and use less memory.

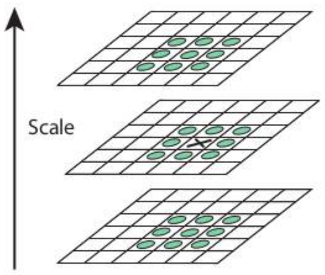


Multiscale Harris and Stephens

- Extending the Harris and Stephens detector to work across scales is easy…
    - We define a Gaussian scale space with a fixed set of scales and compute the corner response function at every pixel of each scale and keep only those with a response above a certain threshold.

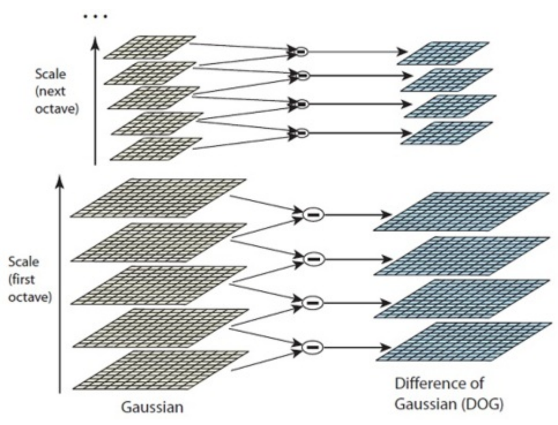
Local extrema in the difference-of-Gaussian scale space

- Recap: Laplacian of Gaussian (LoG) is the 2nd differential of a Gaussian convolved with an image (the kernel is shaped like a *Mexican hat*).
    - By finding zero-crossings of this function you get an edge detector (Marr-Hildreth).
- By finding local maxima or minima you get a *blob* detector!
    - The scale normalised Laplacian operator applies this idea to a Gaussian scale space:
$$\nabla^2_{norm} L(x, y; t) = t(L_{xx} + L_{yy})$$
    - Local extrema (minima or maxima) of this function in both scale and space (i.e. 8 neighbours at scale t, and the 9 neighbours at the scales above and below) indicate blobs of size $\sim\sqrt{2t}$.
    - **Very useful property**: if a blob is detected at $(x_0, y_0; t_0)$ in an image, then under a scaling of that image by a factor s, the same blob would be detected at $(sx_0, sy_0; s^2t_0)$ in the scaled image.



- In practice (for computational efficiency, and coding simplicity), instead of using the LoG function, a simple approximation is made using the Difference of Gaussians (DoG):
$$\nabla^2_{norm} L(x, y; t) \approx \frac{t}{\Delta t}(L(x, y; t + \Delta t) - L(x, y; t - \Delta t))$$
    - This basically means you just need to build a Gaussian scale space, and then subtract adjacent scales to produce a DoG scale space in which you search for extrema.

    - For computational efficiency, every time you double scale, you can half the image size, so a pyramid can be constructed.
        - Referred to as an *oversampled pyramid* as there is more than 1 image (scale) per pyramid level.
        - The set of equally sized images between a doubling of scale is called an *octave*.



Further reading

- Harris & Stephens:
    - Original paper: C. Harris and M. Stephens (1988), A Combined Corner and Edge Detector. http://www.bmva.org/bmvc/1988/avc-88-023.pdf
    - Material also partially covered in Mark's book pp. 159-165
- Scale space:
    - The Wikipedia page has lots of details: http://en.wikipedia.org/wiki/Scale_space_representation
- Blob detection:
    - The Wikipedia pages give a good overview:
        - http://en.wikipedia.org/wiki/Blob_detection#The_Laplacian_of_Gaussian
        - http://en.wikipedia.org/wiki/Difference_of_Gaussians
    - Lowe's seminal SIFT paper (which we'll look at in more detail next lecture) has a good description of finding blobs using the DoG in section 3: David G. Lowe, **"Distinctive image features from scale-invariant keypoints,"** *International Journal of Computer Vision,* 60, 2 (2004), pp. 91-110. http://www.cs.ubc.ca/~lowe/papers/ijcv04.pdf

Practical exercises

- You can try playing with the `org.openimaj.image.feature.local.interest.HarrisIPD` class in OpenIMAJ to get a feel for the kinds of corner features found by the Harris and Stephens method. The `createInterestPointMap()` method will allow you to see the output of the response function *R* across the image (don't forget to normalize it before display though).
    - You could also compare the `HarrisIPD` to the `HessianIPD`, which is a blob detector (the *determinant of the Hessian* blob detector) built using the structure tensor.
- Chapter 5 of the OpenIMAJ tutorial covers finding DoG blobs (and goes a bit further by computing features for each of those blobs and matching them, which is the subject of the next lecture).