Feature Matching

田知本。沈允中。徐士璿

- Block matching
  - Specify feature correspondence
  - What is feature?
    同樣一個物體在不同視角下得到兩張影像, 想要 identify 在這兩張影像中, 哪一個點, 是對應到物體上的同一個點
  - 通常找有特徵的點, ex: 位在角落上的點
  - 最簡單的方法: 用顏色來找, 但是會找到太多顏色一樣的點; 因爲一個點本身的 information 太少, 會有很多 ambiguous 的 match, 所以單獨一個點是不夠的

  - Block matching
    在這個點周圍建立一個 block(window), ex: 5x5, 假設說從這個視角到另一個視角, 這個 window 本身因爲是 local,很小,dense,沒什麼太大變化, 所以假設這個 5x5 window 裡 25 個點的顏色也不太會改變, 這樣子就會有一個比較好的 matching 效果
  - Ex: 紅色方塊, 在scanline上作search, 對每個不同的offset *d*形成一個方塊, 然後拿來跟這個(紅色)方塊做比較算cost, cost最低就表示最接近(接近: slides_8 - **Sum of Squared Disteance(SSD)**
    把這兩個紅色方塊裡的 pixel 值拿來兩兩相減後再平方再取和, 表示它們的"相似程度")
  - 想要找 feautre point, 可是 feature point 太 ambiguous, 所以要找一個 feature descriptor 來描述這個 feature, 要夠 distinctive. 所以把這點周圍這個 5x5 的 block 當成一個 25 維的 descriptor, 再去找看哪一個 25 維的 vector 跟這個 vector 最接近(by SSD). 在這裡 *feature descriptor = 25–d vector*

- Features
  - 一個 feature matching algorithm 被兩個 component 所決定
    1. *Detector*: 決定哪個點是 feature(不容易產生 false matching 的地方), 找出 feature 在哪裡
    2. 對這個 feature 做描述, 有了描述之後, 才能對兩個 feature 做比較 (ex: 25-d vector in block matching, 然後算它們的 distance)
  - 不同的 algorithm 有不同的 detector & descriptor
  - 怎樣是好的feature? slides_13**:**

1. *distinctive*: 很少有其他 feature 會得到一樣的 descriptor, 不同的 feature 對應到 descriptor 不一樣
2. *invariant*: feature 不會受 transformation 的影響

- 25-d 只對 translation invariant, 對 rotation & affine 等不是 invariant

➢ Harris corner detector

- 改進 Moravec detector
- 看這個點周圍的 window 來 recognize 這個 feature, 不須看整張 image
- 一個好的feature, 把這個window shift一下, 看到的東西會很不一樣, ex: slides_18
- *edge*: 沿著某個方向移動, 看到的 image block 差不多, 也就是說這個點在這個 dimension 上會有 ambiguity, 不是一個好 feature
- 所以我們要 detect 的是 corner, 也就是 shift 一下, 看到的東西會很不一樣的點

- Moravec detector:
  - 對(x, y)這個點, 加上shift(u, v), 看shift後的block跟原來的block長得一不一樣(slides_20)
  - w(x, y): window function, 不同的點給不同的 weight,
  - Box function in moravec detector: 這個 block 裡每個點都給它一樣的 weight
  - Measure shift(u,v) 產生出的 error 有多大, 希望找對所有(u, v), E(u,v)都很大的點(x, y), 表示不管怎麼動, 看到的 window 都很不一樣, 才是好的 feature
  - 把所有可能的(u, v)都算一次太慢, 所以只做四個方向的 shift. 對(u,v) = (1,0), (1, 1), (0,1), (-1, 1), 找最小的那個 E(u,v), 如果連最小的都很大, 那其他 E(u, v)一定也很大
  - 所以對每個點產生四個 vector ((1,0), (1, 1), (0,1), (-1, 1)), 對這四個點找最小的 E(u, v), 產生一張 image, 然後再找 local maxima of 這張 image (image of min{E}), 把這樣的點當作是 feature
  - 找右上會 cover 到上面, 找四個跟八個(u, v)應該差不多
  - 問題:

1. binary window function, response 很 noisy, min{E}不是一個很 smooth 的 image, 所以在 detect local maxima 時可能會很 noisy
2. 它只用了四個方向, 都是 45 度角或水平, 能 detect 的 shift 很有限
3. 對 edge 也會有很強的反應, 如果 edge 不是沿著 45 度角或水平,

還是會 min{E}很大

- Harris Corner Detector
  - 針對這三個問題, 分別提出改進
    1. 不 smooth => 用 Gaussian function
    2. 只測試 4 個方向 => 不只測試 4 個方向
       但計算量太大, 用泰勒展開式考慮所有的小 shift(slides_23)

<Taylor's expansion>
[formula]
E(u, v) becomes quadratic function, 它的 E(u, v) = E 的(u, v)畫出來的圖表示一個橢圓
不用把所有(u, v)帶入原本的 function, 只要利用這個 approximation+矩陣運算即可得到 E(u, v), 而 M 這個矩陣可以直接由 image 本身得到

==========

3. 對 edge 反應太強烈 => 利用新的 measurement 方法
   對M做eigen decomposition找eigenvalues lamda1, lamda2, 對應之橢圓爲(slides 26).
   lamda1, lamda2 大表示E(u, v)在該軸方向上變化較大, 再根據lamda1,lamda2 的大小關係(slides 27)判斷是否爲 corner
   因爲求eigenvalue比較慢, 再做一個簡化爲slide 28. 利用前面的原理求corner response R, R越大表示越是corner, 越小表示越不是corner

   從另一個角度來看(slides 30,31):
   - 對 flat 而言, gradient 分布在一個很小的圓內
   - 對 edge 而言, gradient 分布在一個很細長的橢圓內
   - 對 corner 而言, gradient 分布在一個很大的圓內

   Summary (slides 32)

- Harris 一些特性:
  - 對 affine intensity 是 invariant, 因爲是用 gradient, 整張圖亮度提升沒有影響
  - 對 intensity scaling 也是 invariant, threshold 再乘上一個 scale
  - 對rotation也是invariant, 旋轉後還是得到同樣的橢圓, 指

示方向變了(slides 40)

- 最大的問題 --- 對image scale是**non-invariant!** (slide 42), scale改變feature就不見了!!

➢ SIFT (Scale Invariant Feature Transform)

- 1. Detection of scale-space extrema (slide 44~47)

SIFT 是一個 transformation，它會把一個 image 換成一些 feature description。SIFT 分成四個步驟來做，第一個步驟是 detect scale-space 的 extrema。SIFT 最大的一個目的就是要改進 Harris corner detector 不是 scale-invariant 這個問題，那爲了要達到 scale-invariant，基本上就是要在所有的 possible scales 去 search stable features。理論上是這樣，但是實際上是不可能的，所以就採用有點像是去 sample scale space 的方法；之後會看到其實不用 sample 太大，sample 太大有時候可能會有壞處，所以理論上在一個 reasonable 的 sampling frequency 之下，就可以達到一個 scale-invariant 的 feature detection。

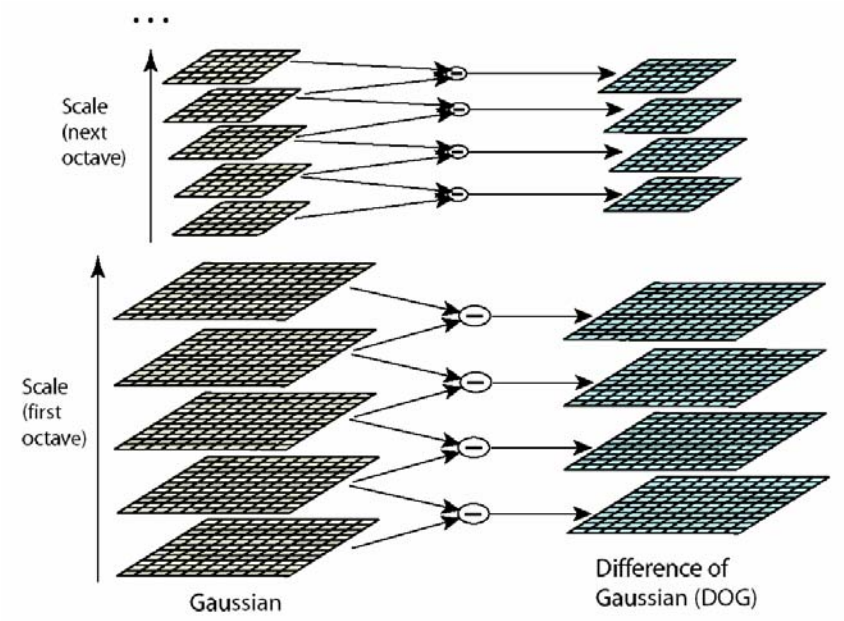
在 SIFT 裡面，是用 Difference of Gaussian (DoG) filter 來建立 scale space。之所以使用 DoG 有兩個原因：第一個原因是 DoG 是一個比較有效率，計算上比較方便快速的 filter；再來就是說它基本上跟 Laplacion of Gaussian 差不多 stable。

- DoG filter (slide 48~49)

就跟之前 Gaussian filter 一樣，由 σ 來決定 kernel 的 size、scale...。給定一個 σ，就可以決定一個 Gaussian filter，用這個 filter 對 image 做 filtering 之後所產上的 L，實際上就可以視爲一個 scale 爲 σ，跟 σ 相關的，而且 scale-variable 的一個 image。那 DoG filter，基本上只是說，給定兩個不同 scale 的 Gaussian filter，相減之後對原來的 image 做 filtering。Apply 這個 filter，相當於就是把 image 對不同 scale 的 Gaussian filter 所產生的 images 相減，他們的效果是一樣的。因爲 DoG filter 基本上只是比原來 Gaussian filter 多做一次減法而已，所以非常 efficient，因此這裡才會用 DoG filter 來建立 scale-space。

接下來的問題就是這個 scale-space 應該長什麼樣子，sample 距離多遠等等。理論上來講，最底層就是沒有做過 Gaussian filter，把 σ double 了之後就跳到另外一個 octave。而把 σ double 了之後，sample rate 就可以減半，相當於 512*512→256*256，然後再 double...，一直做下去。在同一個 octave 裡面，再做一些 sampling，K 就是下面一層的 scale : K* σ ，所以就可以把 K 視爲是兩層相近的 scale 之間的比例是多少，那爲了要讓一個 octave 之間有 s 層

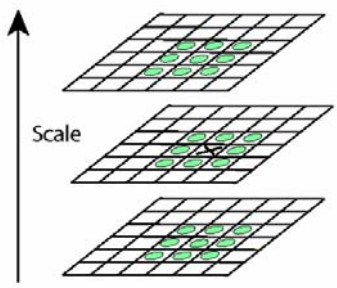
(level)，所以我們定 K=2^(1/s)，因此當做完一個 octave 的時候，frequency 就會剛好 double。



這樣子的做法就跟以前在建 Pyramid 一樣，只是以前 Laplacian 每做一層 frequency 就 double，這樣子的 sample 太過於 sparse。所以這裡爲了得到比較 dense 的 sampling，因此又在每一層 octave 多切了好幾層，每一層基本上我們需要 s+3 張 image，s+1 張是從 σ 到 2σ 之間所產上的，另外因爲在求 extrema 的時候爲了要跟鄰居相比，所以我們還需要兩張是比 σ 小一點和比 2σ 再大一點的 image。所以可以看成是 Laplacian pyramid 的 extension。每一層在經過 Gaussian filtering 過後，把 image 相減就可以得到相當於是經由 DoG filter 所產生的 image，之後就在這個 space 上面做運算。

- 那什麼是extrema呢? (slide 50)

就是這個 pixel(x, y)在 26 個鄰居裡面(上一個 scale 的 9 個點 + 這個 scale 的 8 個點 + 下一個 scale 的 9 個點)，它必須要是 local maximum 或 minimum。因此在第一個步驟裡面，我們就可以把所有是 local extrema 的點給選出來，當作是 feature 的 candidate。



- s怎麼決定? (slide 51~55)

  理論上應該要去 sample 整個 space，但是這樣子太沒有效率，因此我們就要在效率和完整性之間做一個 tradeoff。作者在這邊拿了 32 張 real image，去做一些 transformation，然後設定不同的 s 和 ground truth 相比，看哪個的 repeatable rate(所有的 feature 裡面，找到了幾個)最高，越高越好，根據實驗 s=3 最好，比 3 大的時候會找到一些不 stable 的 feature，而且 s 越大的時候，選到的 feature 點數就越多。

  另外一個實驗就是把原來的 image 256*256 做 linear interpolation 放大到 512*512，再 apply σ=1.6 的 Gaussian filter 做 pre-smoothing，這樣子的結過會比較 robust。

  由實驗的結果，我們可以看到 Laplacian of Gaussian 的效過其實還是最好的，但是 cost 相對較高，不過我們也可以把 SIFT 的 DoG filter 看作是 Laplacian of Gaussian 的一種 approximation。不過不管如何， 至少 SIFT 是比 Harris corner detector 更具有 scale-invariant。

■ 2. Accurate keypoint localization (slide 56)

第一個步驟我們找出了可能是 feature 的點，接下來我們就要想辦法刪除一些不 stable 的 feature point，像是 contrast 比較低的點不要，可能是 edge 的點也不要。

- Accurate keypoint localization (slide 57)

  對於 low contrast 的點，基本上是用一個 3D quadratic function 去 fit，來找出 sub-pixel maxima。這也是一個泰勒展開式，D 是 DoG 的結果，x 是剛剛覺得可能是 feature 的點，根據泰勒展開式，藉由 D 和 x 我們可以找到一個 offset: $\hat{\mathbf{x}}$。這個 $\hat{\mathbf{x}}$ 可以看做是真正 local extrema 的 sub-pixel 的位置，然後再把 $\hat{\mathbf{x}}$ 帶進去泰勒展開式，如果算出來的絕對值小於 0.03，我們就說這個點是 low contrast。

$$D(\mathbf{x}) = D + \frac{\partial D}{\partial \mathbf{x}}^T \mathbf{x} + \frac{1}{2}\mathbf{x}^\mathbf{T}\frac{\partial^2 D}{\partial \mathbf{x}^2}\mathbf{x}$$

$$\hat{\mathbf{x}} = -\frac{\partial^2 D}{\partial \mathbf{x}^2}^{-1}\frac{\partial D}{\partial \mathbf{x}}.$$

- Eliminating edge responses (slide 58)

  跟之前在算 Harris Corner Detector 的 trace 和 det matrix 的方法

類似，這裡如果不滿足下面這個不等式，我們就說這個點可能是 edge，這樣我們就可以把它剔除。

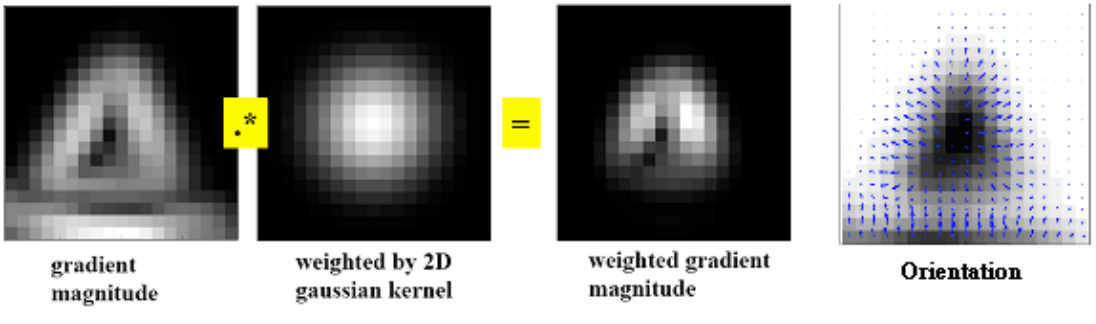
$$\frac{\mathrm{Tr}(\mathbf{H})^2}{\mathrm{Det}(\mathbf{H})} < \frac{(r+1)^2}{r}$$

CYY: 這邊你們聽的懂嗎? 聽的懂嗎?... 你們都沒有回應，我也不知道你們有沒有聽懂，我自己是聽的很懂啦...

■ 3. Orientation assignment (slide 60~69)

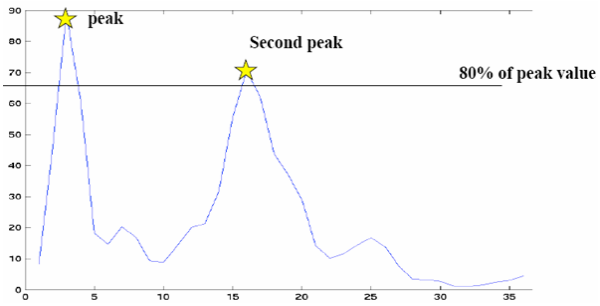
接下來的第三個步驟就是 assignment description，這是爲了接下來要拿來做 matching 用的。第一我們想要找出 major orientation，找到了 major orientation，之後如果我們要做 matching 的話，就可以把所有的 image 轉到這個 local frame，這樣就可以達到 rotation-invariant 的目的。

因此對於每一個 keypoint 我們要去計算它 gradient 的大小和方向。這裡所採用的方法是 orientation histogram，基本上的 concept 是說，對於每一個 keypoint，我們去考慮它臨近周圍一個 window 內的點的 gradient 的方向，最多人投票的方向就當作 major orientation。而每個鄰近的點對中央這個 pixel 的 weight，就是一個 Gaussian distribution 再乘上該點的 gradient 的大小來決定。



如果 orientation histogram 的 peak(超過 80%的點支持這個方向)有好幾個的話，就可能會是 multiple orientation 的情形，那我們就會複製同一個 feature point 在使他們分別朝向不同的方向。不過這種情況並不多見，大概只會有 15%的點會有這種情形。

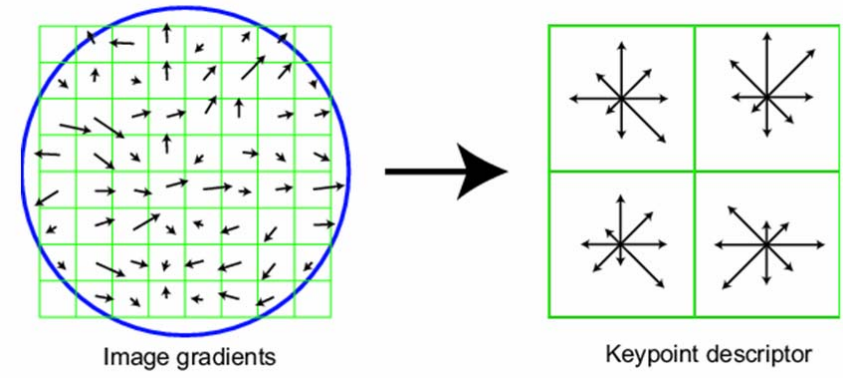
P.S. keypoint ≡ feature point ≡ interesting point



■ 4. Local image descriptor (slide 70~72)

在第三個步驟我們已經找出 feature point 的位置、scale、orientation，接下來第四步驟就是要找出 description。最簡單的方法就是把這個 image block 存起來，但是這樣子太沒有邏輯了。所以這裡所採用的方法基本上還是根據 orientation histogram。

我們把一個 pixel 附近 8*8 這個 window 切成 2*2 的 sub-window，然後去統計每個 sub-window 的 orientation histogram，因此這每個 sub-window 的 orientation 就由 4*4 的 orientations 用之前的方法來決定。每個 orientation 是 8 個 bits，這樣每個 pixel 就會有 4*8=32 個 dimentions。

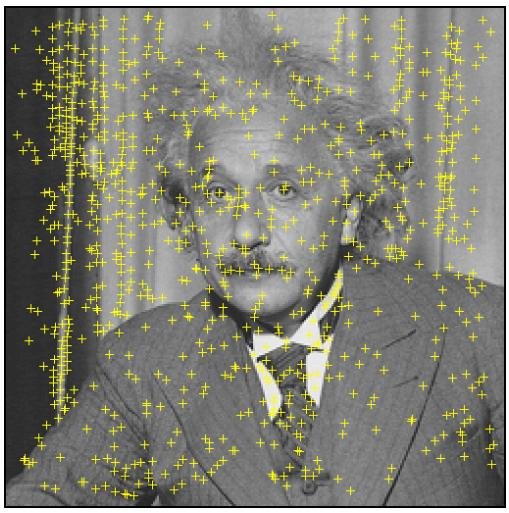


至於爲什麼會用 orientation histogram 來當做 feature point descriptor 呢? 這是因爲用 gradient 來做，會對 illumination 比較 robust，再加上有考慮 orientation，所以對 rotation 也會比較 robust。

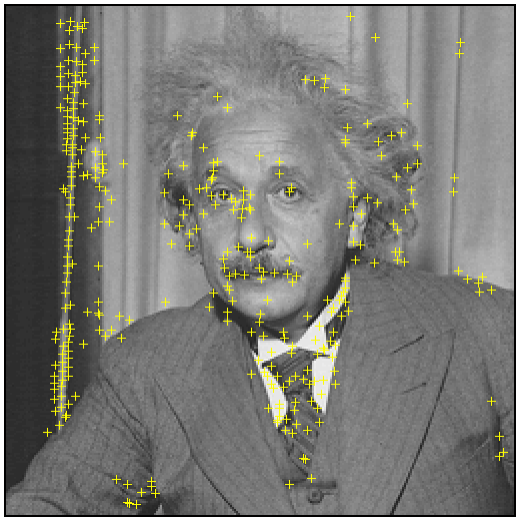
不過實際上我們是用 8 orientations 在加上 4x4 histogram array 總共 128 dimensions 來做。那爲什麼用 4*4*8 呢? 這也是實驗得來的。理論上 SIFT 並不適用於 affine transform，不過根據實驗的結果，SIFT 對 affine transform 還滿 robust 的。

■ Results:

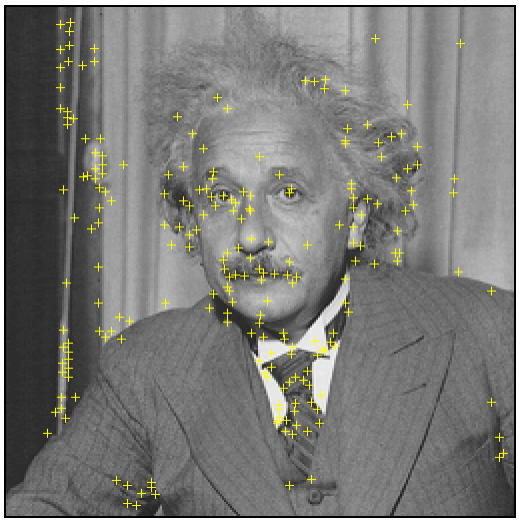
- Local extrema



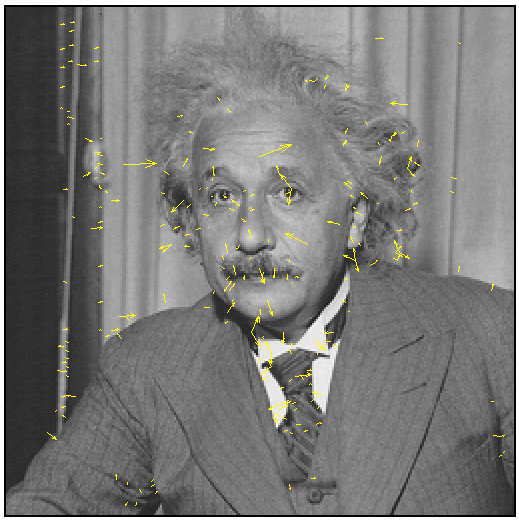
- Remove low contrast



- Remove edges



- SIFT descriptor



◦ SIFT extensions

底下介紹兩個 extensions，都自稱比原本的 SIFT 更好。兩者皆用到 PCA (Principle Component Analysis) 的概念。以人臉爲例，一張 64x64 人臉照片理論上應該是一個 64x64 維的 vector，因此描述該影像時，就得用如此高維度的資料來表示。但是由於“人臉”具有一定的特性，因此並不是任意 64x64 維的資料皆可形成人臉，而那些之所以形成人臉的資訊，便是在此高維度資料中真正重要的資訊。PCA 就是一種計算如何用較低維度的 representation 來表達一具有高維資訊的方法。

PCA：把 high dimensional vector 投影成 low dimensional vector，並可把此 low dimensional vector 還原回可逼近原來的 high dimensional vector。

◦ PCA-SIFT

改變原來 SIFT 中的步驟四。在 feature (gradient space) 周圍 41x41 的 block 中去計算它的 principle components (feature 之所以爲 feature 就表示它具有特殊的性質，不是任意 41x41 block 皆可形成 feature，因此存在有更精簡的表示方法)。PCA-SIFT 將原來的 2x39x39 = 3042 維的 vector (block 的上下左右邊界不看，所以是 39)，降成 20 維 (SIFT 要 128 維) 以達到更精簡的表示方式。作者並宣稱可以有比 SIFT 好的效果。

◦ GLOH (Gradient Location-Orientation Histogram)

作者說它這方法比 PCA-SIFT 更好。方法是把原來 SIFT 中 4x4 棋盤格的 location bins 改成用放射狀同心圓的 17 location bins 來表示，並計算其中的 gradient orientation histogram (orientation 的方向分類爲 16 種)，因此總共是 16x17 = 272 維度的表示方式，之後再做 PCA 將之降維成 128 維的資訊，因此保有跟 SIFT 一樣精簡的表示方法，但比 SIFT 有更佳的 performance。

◦ Application

- Recognition：例如先拍出玩具車各個角度的照片，算出各張圖的 features 存在 database 中，之後進來新的圖片若想知道裡面有沒有玩具車時，就去計算這張新進來的圖片的 features 並將這些 features 拿去之前建好的 database 做 search，若有 match 則表示裡面存在有玩具車。
- 3D object recognition：道理同上。Robustness 在於既使物體部分被遮擋住，仍可辨識出來。
- Office of the past：認出桌上的文件跑到什麼地方去。
- Image retrieval：用 feature match 在大量 database 中找出想要的照片。因爲是 feature-based 所以視角改變並不會影響結果。
- Robot location：利用房間內的 features 所建立的 database 可幫助 robot 知道自己在什麼位置。Sony 的 Aibo 就有把 SIFT 做進去，可以讓 Aibo 在快沒電時能夠認出可以充電的地方自己走過去，或是可以踢足球等等的應用。
- Structure from Motion：一種做 matchmove 的方法。根據 2D feature matching 來估計 3D 的參數。
- Automatic image stitching：自動把一堆 pictures 接成 panoramas。