# CS4243 Cheatsheet

Ng Xuan Jun

**Image Formation Model**:

$$f(x,y) = i(x,y) \cdot r(x,y)$$

$i(x,y)$ – Amount of source illumination incident on the scene, $0 \le i(x,y) < \infty$

$r(x,y)$ – Amount of illumination reflected by objects in the scene, $0 \le r(x,y) \le 1$

- 0 for total absorption, 1 for total reflectance (All energy is reflected)

**Resolution**:

1. **Spatial Resolution**

Higher Resolution → Will have a detailed look of what we are looking at

Lower Resolution → Very pixeled and we will not know what we are looking at

2. **Intensity Resolution**

More Bits → More grey levels and can discern between different levels

**File Size**: Spatial and Intensity resolution determines the file size

$$b = M \cdot N \cdot k$$

$M \cdot N$ – Size of the image, $k$ – Bit depth of the image

**Maximum number of gray scale levels**: $(N-1) \times n + 1$, Where $N$ is the number of grey levels and $n$ is the number of channels that it is split equally by

## Filtering

**Image Normalization (Whitening)**: Resulting image pixels are zero-mean, unit variance.

- Allows for removal of contrast and constant additive luminance variations

$$\mu = \frac{\sum_{i=1}^{I}\sum_{j=1}^{J} p_{i,j}}{IJ}, \qquad \sigma^2 = \frac{\sum_{i=1}^{I}\sum_{j=1}^{J}(p_{i,j}-\mu)^2}{IJ}, \qquad x_{ij} = \frac{p_{ij}-\mu}{\sigma}$$

**Gamma Mapping**: Mappings can be non-linear

$$x_{ij} = 255 \cdot \left(\frac{p_{ij}}{255}\right)^{\gamma}, \qquad \gamma > 0$$

- Small $\gamma$ → Increases the range for the dark areas, makes the dark areas become brighter.
- Large $\gamma$ → Increases the range for the bright areas, makes the bright areas become darker.

**Histogram Stretching**: Stretch the boundaries to [0, K]

$$x_{ij} = \frac{(p_{ij} - f_{\min})}{(f_{\max} - f_{\min})} \times K$$

$f_{\min}$ – Minimum value observed, $f_{\max}$ – Maximum value observed, $p_{ij}$ - Value at pixel $ij$, K – maximum value that is possible in that image

**Histogram Equalisation**: Forces the cumulative distribution of the intensity to be linear

$$x_{ij} = K \cdot c_{p_{ij}}$$

$K$ – Maximum value observed, $c_{p_{ij}}$ – Cumulative distribution value at pixel $ij$

**Note**: If the images have a lot of boundaries vs another that does not have much have the same histogram originally. If we do blurring, the one with more boundaries will have more counts for in between values and therefore their histograms will no longer be the same.

**Image Normalisation (Whitening) vs. Histogram Equalisation**:

- Equalisation boosts contrast more
- However, equalisation is much more expensive

**Otsu's Method**: Minimise the sum of weighted variance of object and background by choosing the optimal threshold.

$$T^* = \arg\min_T w_1(T)\sigma_1^2(T) + w_2(T)\sigma_2^2(T)$$

$\sigma_1^2(T), \sigma_2^2(T)$ – Variance of pixels less than or equal to and greater than threshold

$w_1(T), w_2(T)$ – Number of pixels less than or equal to and greater than threshold

**Normalised Cross-Correlation**: Normalize cross correlation to reduce effect of white patch

$$x_{ij} = \frac{1}{|F||w_{ij}|}\sum_{u=-k}^{k}\sum_{v=-k}^{k} f_{uv} \cdot p_{i+u,j+v}$$

$|F|$ – Magnitude of the filter, $|w_{ij}|$ – Magnitude of the input widow size. The remaining part is the normal cross correlation term. The front part is a scaling factor.

**Noise in Image**:

| Noise | Description | Solution |
|---|---|---|
| Impulse & salt and pepper noise | Random occurrences of white (and black) pixels, usually due to defective sensor elements | Median Filter |
| Gaussian Noise | Variations in intensity drawn from a Gaussian distribution due to inherent noise of the sensor | Gaussian Filter, Box Filter |

**Filters**:

| Filter | Matrix (Example) | Effect of the Filter |
|---|---|---|
| Horizontal Sobel Filter | $\begin{bmatrix}1&0&-1\\2&0&-2\\1&0&-1\end{bmatrix}$ | Detects horizontal gradients and therefore can detect vertical lines. Takes $O(w \times h)$ |
| Vertical Sobel Filter | $\begin{bmatrix}1&2&1\\0&0&0\\-1&-2&-1\end{bmatrix}$ | Detects vertical gradients and can detect horizontal lines. Takes $O(w \times h)$ |
| Median Filter | 3 × 3 Median Filter. Just take the median value of the 3 × 3 area | Gives a boxy look. Helps to remove noise from the image. Note that the if the kernel is small enough, we won't have a boxy effect. Removes spikes in the image. |
| Box Filter | $\frac{1}{9}\begin{bmatrix}1&1&1\\1&1&1\\1&1&1\end{bmatrix}$ | It is technically just a Gaussian Blur. Need to look at the averaging term to see how much is being averaged |
| Gaussian Filter | $\frac{1}{16}\begin{bmatrix}1&2&1\\2&4&2\\1&2&1\end{bmatrix}$ | Makes use of the neighbouring values and therefore gives a smoothing effect. Could perform a better job than box filter. |
| Shift Filter | $\begin{bmatrix}15&0&0\\0&0&0\\0&0&0\end{bmatrix}$ | Shifts the image. Note that it will move in the opposite direction of the filter. (i.e. This filter is up and left, this will cause the image to move down and right) |
| Sharpen Filter | $-\frac{1}{9}\begin{bmatrix}1&1&1\\1&-17&1\\1&1&1\end{bmatrix}$ | Accentuates the details on the image. Accentuates difference with local average. Noise and edges will get emphasised but no effect on flat regions. |
| 1D Derivative Filter | $[-1 \quad 0 \quad 1]$ | Detects edges, in this case will be horizontal lines |
| 1D Laplace Filter | $[1 \quad -2 \quad 1]$ | Detects edges, in this case will be horizontal lines. Similar to derivative filter |
| 2D Laplace Filter | $\begin{bmatrix}0&1&0\\1&-4&1\\0&1&0\end{bmatrix}$ | Like sharpening filter but helps us to detect edges through accentuating the differences. |

**Template Matching**:

| Type | What would be detected | Issues |
|---|---|---|
| Cross Correlation | Bright patches will give the strongest response | Due to **bright spots having large intensity values**, it will give large response for the template match even if it doesn't match |
| Convolution | | |
| Normalised Cross Correlation | Allows for better detection with respect to the template. Not as susceptible to bright patches. | **Not really an issue**: If the patches that we try to detect is not the exact template. Then the one that looks most similar ("blob" that looks the most similar) will give the strongest response. |

**Possible Pre-Processing Steps**:

| Pre-Processing Step | Benefits | Cons |
|---|---|---|
| Sharpening | Can sharpen some of the details and helps with processing after | Too much could cause over emphasis on the noise |
| Intensity Thresholding - Use Ostu's Method for Automatic Thresholding | Useful for removing extreme background (Black or White) | If our image has bright spots / dark spots as the features then may not be optimal |
| Histogram Stretching | Could help to improve contrast | If there are already dark or bright spots (0 / 255 type) then it won't be able to stretch the range |
| Histogram Equalisation | Helps to map the distribution of the histogram to be linear. Helps will improving contrast, probably better than histogram stretching in most cases | May cause some pixels to darken / whiten due to higher proportion of brighter / darker patches overall in the image. |
| Canny Edge Detection | Helps to convert thick regions to thinner regions for better detection of lines afterwards | |

## Gradients

**Gradients and Edges**:

| Filter | Description | Benefits / Cons |
|---|---|---|
| Horizontal Sobel Filter | Type of derivative filter to detect vertical lines. | Susceptible to noise. However, detected edges will be very sharp. |
| Vertical Sobel Filter | Type of derivative filter to detect horizontal lines. | Susceptible to noise. However, detected edges will be very sharp. |
| Derivative of Gaussian (DoG) | Combines the Gaussian Filter with a derivative filter (Like Sobel) | Applies blurring to the image to help remove some of the noise. However, the detected image may not be as sharp. Edges are at peaks of white points. |
| Laplacian of Gaussian (LoG) | Combines the Gaussian Filter with a Laplacian filter | Applies blurring to the image to help remove some of the noise. However, the detected image may not be as sharp. Blob detector Edges are at zero-crossing but not a preferred choice since it is hard to find. Can be estimated by Difference of Gaussians |

**Canny Edge Detector**:

1. Filter image with Derivative of Gaussian (DoG)
2. Find magnitude and orientation of gradient
3. Non-maximum suppression

Thin wide "ridges" down to single pixel width

4. Linking and thresholding (hysteresis):

Use the high threshold to start edge curves and connect them with the low threshold edges

## Lines and Houghs

**Double Intercept Form**: $x/a + y/b = 1$

**Normal Form**: $x\cos\theta + y\sin\theta = \rho, \ a = \cos\theta/\rho, \ b = \sin\theta/\rho$

**Line (Normal Form) Detection with Hough Transform**:

1. Quantize Parameter Space $(\theta, \rho)$
2. Create Accumulator Array $A(\theta, \rho)$
3. Set $A(\theta, \rho) = 0, \forall\theta, \rho$
4. For each image edge point $(x_i, y_i)$

For each element $\theta$

Solve for $\rho = x_i\cos\theta + y_i\sin\theta$

Increment $A(\theta, \rho) = A(\theta, \rho) + 1$

**Hough Transform for Circle**:

$$(x-a)^2 + (y-b)^2 = r^2$$

Note that we will take $r$ to be a constant and the centre of the circles are the parameters.

We will create a different accumulator for each radius.

**Leveraging Gradient Information**:

- If we have a lot of points, it could cause a lot of votes to be distributed across the accumulator and making it hard to discern the actual maxima.
- We can make use of the gradient information to **minimise the votes** and **reduce computation time**.
- Make use of only the 2 directions of the orientation to cast the votes rather than all directions.

**Generalized Hough Transform**:

**Assumption**: Translation is the only transformation. (i.e. orientation and scale of the arbitrary shape are fixed)

1. Define a model shape by its boundary points $p_i$ and reference point $a$ that we will try to vote towards.
2. For each edge point:
   a. Use its gradient orientation $\phi$ to index into stored table.
   b. Compute some averaging statistics for each of the points (e.g. mean, mode)
   c. Use retrieved $r$ vectors to vote for reference point.

**Hough Voting for Object Detection**:

**Idea**: Instead of indexing displacements by gradient orientation, index by matched local patterns (visual codewords)

**Note**: This visual codewords will be pre-generated and it should be able to discern between different directions. (i.e. For car tyres, we can point to 2 sides and when 2 tyres vote for the same centre, it will be centre of a car)

**Hough Transform**:

| Pros | Cons |
|---|---|
| All points are processed independently so can cope with occlusions and gaps | Search time complexity increases exponentially with the # model parameters |
| Some robustness to noise: Noise points unlikely to contribute consistently | Non-target shapes can produce spurious peaks in parameter space |
| Can detect multiple instances of a model in a single pass For instance, if there are multiple lines in the same image, they can be detected at once | Quantization: Can be tricky to pick a good grid size |

## Segmentation

**K-Means Clustering**: Minimises average squared distance within-cluster. It does not try to find the actual cluster means but works on minimising the objective function instead

1. Given $K$, randomly initialize the cluster centers, $c_1, \cdots, c_k$
2. Given clusters centers, determine points in each cluster

For each point $p_i$, find the closest $c_j$. Put $p_i$ into cluster $j$

3. Given points in each cluster, solve for $c_j$

Set $c_j$ to be the mean of points in cluster $j$

4. If $c_j$ have changed (up to some threshold), repeat Steps 2, 3.

| Pros | Cons |
|---|---|
| Simple, fast to compute | Clusters based on intensity don't have to be spatially coherent |
| Converges to local minimum of within-cluster squared error | Requires sufficient number of points to ensure convergence to the correct clusters |
| | Sensitive to initial centers |
| | Sensitive to outliers (since we are taking mean) |
| | Detects only spherical clusters |
| | Assumes that means can be computed (efficient, meaningful) |
| | Hard to determine the number of $k$ we need initially |

**SLIC Superpixeling**: A group of pixels that share common characteristics locally

**Computation**:

$n_{sp}$ – Number of superpixels, parameter that we set

$n_{tp}$ – Total number of pixels, determined by $W \times H$

$s = \left(\frac{n_{tp}}{n_{sp}}\right)^{1/2}$ – Initial spacing of each superpixel.

$n_W = W/s$ – Number of pixels in each superpixel's width

$n_H = H/s$ – Number of pixels each superpixel's height

$D = \left[\left(\frac{d_c}{d_{cm}}\right)^2 + \left(\frac{d_s}{d_{sm}}\right)^2\right]^{1/2} = \left[d_c^2 + \left(\frac{d_s}{s}\right)^2 c^2\right]^{1/2}$ – Compose distance and note that $s$ and $c$ will determine how much weightage is placed on each of the terms.

**Changing the value of $c$**:

If $c$ large → Spatial information more important. We will have more regularly shaped segments. It will be a hexagonal shape which has the shortest distance to the center. Shifts in the centers can be computed through the perpendicular bisector.

If $c$ small → Colour information more important. We will have more irregularly shaped segments.

$c$ – Maximum colour value. $s$ – Maximum spatial value is the spacing between 2 superpixels.

**Algorithm**: Note that we can do segmentation with this after the super-pixeling → Make use of K-Means as well.

1. Compute the initial super pixel clusters,

$$m_i = [r_i, g_i, b_i, x_i, y_i]^T, \qquad i = 1, 2, \cdots, n_{sp}$$

Sample the image at regular grid steps, $s$. Move the cluster centers to the lowest gradient position in a 3 × 3 neighbourhood. For each pixel location, $p$, in the image, set a label $L(p) = -1$ and a distance $d(p) = \infty$

- Note that we do not want to initialise at an edge or noise so that it will be similar to the points around it

2. Assign samples to cluster centers. For each cluster center $m_i, i = 1, 2, \cdots, n_{sp}$, compute the distance $D_i(p)$ between $m_i$ and each pixel $p$ in a $2s \times 2s$ neighbourhood about $m_i$. Then for each $p$ and $i = 1, 2, \cdots, n_{sp}$, if $D_i < d(p)$ let $d(p) = D_i$ and $L(p) = i$. (Only compute distance to closest set of cluster centers)
3. Update the cluster centers by computing the new mean (as per K means)
4. **Test for convergence**: Compute Euclidean norms of the differences between the mean vectors in the current and previous steps. Check the residual errors for the super pixels, if the sum is less than $T$, a specific threshold then go to Step 5 else repeat Step 2.
5. Post processing (Optional to create "Stained-Glass" Effect) : Replace all the super pixels in each region, $C_i$ by their average value $m_i$

| Pros | Cons |
|---|---|
| Could be faster than K-Means since we are only considering neighbours in a range of $2s \times 2s$ | No provision is made to enforce connectivity of super pixels. Could have isolated pixels to remain after convergence |
| Can be used in other color spaces and distance can be other values that we can derive meaningful distance measure | If we do segmentation, we may have not get the exact shape back. (i.e. circle) |

**Mean Shift**: Find modes or local density maxima in feature space

**Algorithm**: Do the following for each data point

For each point $x_j$, where $j = 1, \cdots, n$

1. Initialise the density window $x: x = x_j$
2. Compute the mean shift vector $m$

$$m_{h,G}(x) = \left[\frac{\sum_{i=1}^{n} x_i g\left(\left\|\frac{x - x_i}{h}\right\|^2\right)}{\sum_{i=1}^{n} g\left(\left\|\frac{x - x_i}{h}\right\|^2\right)}\right] - x$$

3. Shift the density window and update

$$x' = x + m(x), \qquad x = x'$$

4. Repeat until the centroid stops moving (convergence)

**Segmentation with Mean Shift**:

1. Find features (colour, gradients, textures, etc) to represent each pixel
2. Initialize density windows at individual feature points
3. Perform mean shift for each point until convergence
4. Merge pixels whose feature points that end up near the same mode or "peak" into the same segment

**Increasing bandwidth, $h$ / window size**: When bandwidth increases, we will be including more points in the computation and makes the mean shift vector more accurate and will try to converge towards the actual mean. **Number of segments generated decreases**.

**Note**: We need a large enough amount of neighbours to ensure that the algo can progress.

| Pros | Cons |
|---|---|
| Finds variable number of modes (don't have to pre-specify) | Output depends on bandwidth, $h$ and selecting it needs trial and error |
| No prior assumptions on cluster shape | Computationally expensive and slow to run |
| 1 parameter only (window size/bandwidth) which represents the scale of local max we are intending to find | Scales poorly with feature space dimension since we need to repeat with all points |
| General algorithm for mode-finding | |
| Robust to outliers | |

**Speedups**:

1. Can assume that all points within radius $r$ of end point after mean shift will belong to the same cluster / mode. We can assign them to the same cluster and don't have to run on mean shift on them
2. Assume that points in the same path will be attracted to the same basin of attraction.Assign all points within radius $c$ of search path to mode. But it will most likely be smaller than $r$.

## Textures

**Choice of window scale for texture representation**: Perform scale selection by looking for window scale where statistic (texture representation) does not change much.

**Gabor Filters**: Can be used to create many different filters for the use of a filter bank

$$f_{mn} = \frac{1}{2\pi\sigma^2}\exp\left[-\frac{m^2+yn^2}{2\sigma^2}\right]\sin\left[\frac{2\pi(\cos[\omega]\,m + \sin[\omega]\,n)}{\lambda} + \phi\right]$$

$m, n$ – 2D coordinates of the kernel, $\sigma$ – Controls the spread of the Gaussian kernel, $\omega$ – Controls the orientation of the sinusoid in 2D, will influence the orientation of the detected feature, $\lambda$ – Wavelength of the sinusoid (Tells us the frequency of the sinusoid for us to see the rest of the peaks when merged with the Gaussian), $\phi$ – Controls the amount of shift from center of the kernel.

**Gabor Filter Estimations**: If scale is small compared to the frequency, Gabor Filters can be used as approximate derivative operators.

- Gaussian Derivative: Can be estimated by odd Gabor Filter
- LoG: Can be estimated by even Gabor Filter

**Computing Textons**: Replace each pixel with integer representing texture "type"

1. Apply filter bank to (training) images
2. Cluster in feature space and store cluster centers. This forms a texton dictionary. Note that this is done using the training images.
3. For new (test) image, filter image with same filter bank to get feature representations for each pixel based on feature vectors at each pixel, assign to the nearest cluster. The cluster ID is the texton ID.

**Histogram of Textons**:

For a given region, compute a histogram of textons as representation: vector storing number of occurrences of each texton.

**Note**: When we have relative counts, it could help if the scale of the images are different

# CS4243 Cheatsheet

Ng Xuan Jun

**Make use of Texton Histogram for Classification**: If a new region is most similar to a texton histogram → Assign to the label
**Drawbacks**: Effectiveness is dependent on the size of the window as the window size could make it hard to localise.

**Texture Boundaries**:

1. Consider a disc, split into two halves by a diameter of a particular orientation
2. Measure the difference in texture between the two halves by comparing texton histograms
3. Try all possible orientations → If we get a strong difference then it should be a texture boundary

## Keypoints

**Equivariance**: If there is a transformation on the image, there should be the same transformation on the keypoint. (Location of the keypoint should be transformed in the same manner as the image)
**Invariance**: Image is transformed and detection (score) does not change.

**Harris Corners**: We have a fast approximation for the estimation of the values

$$R = \lambda_1\lambda_2 - \kappa(\lambda_1 + \lambda_2)^2, \quad \det(H) = \lambda_1\lambda_2, \quad trace(H) = \lambda_1 + \lambda_2$$

$x_{\max}$ – Eigenvector with direction of larger increase in error → Minor Axis ($\lambda_{\max}^{1/2}$)
$x_{\min}$ – Eigenvector with direction of smaller increase in error → Major Axis ($\lambda_{\min}^{1/2}$)
**Determinant of $2 \times 2$**: $ad - bc$, Trace of matrix is just the diagonals
**Algorithm**:

1. Compute gradient at each point in the image
2. Compute H matrix for a window centered at every pixel in the image

$$H = \sum_{(x,y)\in W} w_{x,y} \begin{bmatrix} I_x^2 & I_xI_y \\ I_xI_y & I_y^2 \end{bmatrix}$$

3. Compute cornerness score: $\det(H) - \kappa(tr(H))^2$
4. Find points whose surrounding window gave large corner response ($R >$ threshold)
5. Take the points of local maxima, i.e. perform non-max suppression

**Note**: We cannot just find corners through those with high gradients in x & y because there are other image patches that have high gradients in both. (e.g. diagonal lines)

**Non-Maximum Suppression**: Iteratively search for max values, then zero out everything else
**Hyperparameter: Size of window**
**Size of window too large** → We may unintentionally remove some corners
**Size of window is too small** → We may include too many local maximum where in fact they are supposed to be from the same region

**Adaptive Non-Maximum Suppression**: Pick corners which are both local maxima and whose response is significantly greater (e.g. 10%) than all neighbouring local maxima within some radius $r$
**Rationale**: Simple search of maxima will lead to mostly interest points with high contrast (White to Black) which may not be optimal since we may want to have corners from low contrast areas as well. By doing this, we will have a better spread of corners around the image

| Algorithm / Method | | Invariance/Equivariance | | | | | |
|---|---|---|---|---|---|---|---|
| | | Translation | Image Flip (Mirror x/y-axis) | Rotation | Scaling (proportionate) | Additive Intensity (Photometric) | Scaling Intensity (Photometric) |
| Detectors | Harris Corners | ✓ | ✓ | ✓ | | ✓ | |
| | Laplacian of Gaussian | ✓ | ✓ | ✓ | | ✓ | ✓ |
| | SIFT | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| Descriptors | MOPS | ✓ | | | | | |
| | GIST | | | | | ✓ | |
| | SIFT | ✓ | | ✓ | ✓ | ✓ | |
| | Weighted Intensity | ✓ | ✓ | ✓ | | | |
| | Weighted Gradients | ✓ | ✓ | ✓ | | ✓ | |

**Equivariance/Invariance of Harris Corners**

| Transformation | Equivariant | Invariant |
|---|---|---|
| Translation | True, because derivatives $H$ is obtained through convolution which is translation equivariant | True, eigenvalues for cornerness scores is also based on derivatives |
| Rotation | True, eigenvectors represent the direction of max/min change in appearance, so rotate with patch | True, eigenvalues represent the corresponding magnitude of max/min change so it is constant |
| Photometric Transformation | Additive Changes (Changes in overall brightness) – True | |
| | Scaling of Intensity (Changes in contrast) – False | |
| Scaling | False – Since if we scale the image up, we could detect many more points as keypoints instead | False |

**Laplacian of Gaussian**: Highest response when the signal has the same characteristic scale as the Gaussian. (i.e has similar shape and width of the Gaussian). It is a "blob" detector which responds strongly to circles.
**Note**: We can make use of difference of Gaussians to approximate this, i.e. take a bigger Gaussian to minus a smaller Gaussian as a cheaper approximation
**Size of the filter**: By changing the size of the filter, we will be able to detect key points of differing scale.

## Descriptors

| Possible Descriptor | Benefits | Issues |
|---|---|---|
| Intensity | Good if geometry and appearance unchanged | If lighting changes, then we will not be able to detect these points |
| Gradients | Invariant to absolute intensity values | Shifting it by a little through translation and it will make all the descriptors very different |
| Colour Histogram | Invariant to scale (assuming the histogram is normalised) and rotation | Not sensitive to spatial layout because there could another image that has similar colour histogram but different layout of the pixels |
| Spatial Histogram | Retains rough spatial layout and some invariance to deformations | It is invariant to slight rotation but since it's a grid, if we rotate it too much, we no longer can find the points anymore |

**Histograms**:
**Choosing of range**:

- Grids of entire domain → Fast → Only applicable when dimensionality is low
- Clustering → Slower → But can quantize data in higher dimensions

**Bin Size**:

- Few Bins → Less Data required → But coarser representation
- Many bins → More data required → But finer representation

**Comparison between histograms**: Histogram Intersection/ Euclidean could be faster but Chi-Squared often works better
**Issue with Histogramming**: No localisation and sensitivity to spatial layout

**Multi-Scale Oriented Patches (MOPS)**: Local Descriptor

1. Take a $40 \times 40$ window around keypoint (eg. Harris corner)
2. Further split the window into $5 \times 5$ cells (subsampling ⇒ absorbs localization errors/increase tolerance)
3. Rotate to horizontal (Find dominant gradient and rotate all cells to same direction) ⇒ rotation invariant
4. Normalise intensity value within the window $\left(\frac{I-\mu_I}{\sigma_I}\right)$ ⇒ robust to photometric variations

Wavelet transform on $8 \times 8$ patch to get a 64-dimensonal descriptor (similar to PCA)

**Note**: Rotational Invariant, some photometric invariance but Not scale invariant
**GIST Descriptors**: Global Descriptor

1. Divide entire image (patch) into $4 \times 4$ cells
2. Apply Gabor filter (filter bank of directional edge detectors) ⇒ each pixel has $N$ descriptors
3. Compute filter response averages for each cell ⇒ each cell has $N$ descriptors
4. Size of descriptor is $4 \times 4 \times N$, where $N$ is size of filter bank

**Scale Invariant Feature Transform (SIFT)**: Note that the detector and descriptor can be used separately
**SIFT Detector**

1. Multi-scale extrema detection – Scale invariance
Makes use of the LoG to detect blobs at different scales. Makes use of difference between Gaussians to estimate LoG effect
2. Keypoint Localization
Refine location of keypoints to sub-pixel accuracy via Taylor series expansion across the octaves.
3. Orientation assignment / rotation – Rotation invariance
Note that edges are computed through the Hessian of the DoG response image, retaining maxima only if ratio of max to min eigenvalue of the Hessian is above some threshold

**Peak Threshold Parameter**: For the DoG response filtering (Higher is better)

- Increase in threshold → Lesser maxima will remain

**Edge Threshold Parameter**: For edge response image (Lower is better)

- Decrease in threshold → Lesser edges detected → This is good because we only want corners and the ratio of max/min for corners are small whereas it is large for edges

**SIFT Descriptor**

1. Take $16 \times 16$ window around detected keypoint from image at scale matching to keypoint. Partition window into a $4 \times 4$ grid of cells (gives some sensitivity to spatial layout)
2. Compute gradient orientations for each pixel, reweight magnitudes according to a Gaussian centered on keypoint and discard pixels with low magnitude
3. Of the remaining edge orientations, create a histogram with 8 orientation bins for each cell. To be rotation invariant, "shift" histogram binning by the dominant orientation. Collapse into vector ($6 \times 8 = 128$ dims)
4. Normalize vector to unit length, clamp values based on threshold, re-normalize again for final descriptor

| Pros | |
|---|---|
| Highly Robust | Invariant to Scale and Rotation (not strictly) |
| Quick and Efficient | Can handle changes in viewpoint (Up to 60 degree out of plane rotation) |
| | Can handle significant changes in illumination (Sometimes even day vs night) |

**Matching Features**:

1. **L2 Distance**: $||f_1 - f_2||$ – Can give small distances for ambiguous (incorrect matches)
2. **Ratio Distance**: $||f_1 - f_2||/||f_1 - f_2'||$ - Makes use of the ratio with the second best to ensure that we match feature pairs that are distinct

**Note**: We can threshold the ratio test as well to reduce the number of incorrect matches. Therefore, only those that are unique (small values) will be kept. If we increase threshold too much, number of correct matches that we get per increase will decrease because we are allowing for more ambiguous matches.

| Terms | Definition |
|---|---|
| True Positive (TP) | Detected keypoint is matched correctly with its corresponding keypoint. (Each match pair is considered 1 count) |
| False Positive (FP) | Detected keypoint is matched incorrectly with another keypoint (Each match pair is considered 1 count) |
| True Negative (TN) | Keypoints which should not be matched since the corresponding keypoints were not found in the other image |
| False Negative (FN) | Matches that we missed. Corresponding keypoints which are detected in both images but not matched. image (Each match pair is considered 1 count) |
| Precision | $TP/(TP + FP)$ – Tells us how accurate the feature pairs that are declared as matches<br>Increase threshold → Wont know whether will increase precision |
| Recall | $TP/(TP + FN)$ – Tells us whether the algorithm can identify all the actual pairs of features that can be found<br>Increase threshold → Increase recall because we will have more samples recognised as positive. |
| Specificity | $TN/(TN + FP)$ – Tells us whether the algorithm can correctly disregard the features which are not part of any pair<br>Increase threshold → Decrease specificity because we will have more samples recognised as positive → More FP → Denom ↑ |

**Solving Least Squares Conditions**:

1. $A^TA$ should be invertible
2. $\det(A^TA)$ should not be too small, $A^{-1} = adjunct(A)/\det(A)$ → $\lambda_1$ and $\lambda_2$ should not be too small
3. $A^TA$ should be well conditioned → $\lambda_1/\lambda_2$ should not be too large ($\lambda_1$ is the larger value)

**Automatic Scale Selection**
When looking for keypoints, find the scale that give local maximum of $f$:

- Harris Operator ($R$): For the same keypoint, go through varying scale of window and calculate $R$. Use the scale that maximises $R$ value.
- Laplacian of Gaussians/Blob Detector: Similar idea. Alternatively, instead of scaling filter size, we can scale the image and fix the filter size but also iterate through the various sigma value.

## Homography

**Applying a Homography**:

1. Convert to homogeneous coordinates. $p = [x \quad y]^T \Rightarrow P = [x \quad y \quad 1]^T$
2. Multiply by the homography matrix $P' = H \cdot P$
3. Convert back to heterogeneous coordinates $P' = [x' \quad y' \quad w']^T \Rightarrow p' = \left[\frac{x'}{w'} \quad \frac{y'}{w'}\right]^T$

**Solving for the H matrix**: Note that we need **4 pairs of points** (8 points total) to solve the homography and we only have 8 degrees of freedom because we can divide by $h_9$ throughout and it gets absorbed.

$$A_i h = 0$$
$$h_7xx' + h_8yx' + h_9x' - h_1x - h_2y - h_3 = 0 \cdots (1)$$
$$h_4xy' + h_5yy' + h_6y' - h_1x - h_2y - h_3 = 0 \cdots (2)$$
$$A_i = \begin{bmatrix} -x & -y & -1 & 0 & 0 & 0 & xx' & yx' & x' \\ 0 & 0 & 0 & -x & -y & -1 & xy' & yy' & y' \end{bmatrix}$$
$$h = [h_1 \quad h_2 \quad h_3 \quad h_4 \quad h_5 \quad h_6 \quad h_7 \quad h_8 \quad h_9]^T$$

**Solve for $H$ using DLT**:
Given $\{p_i, p_i'\}$ solve for $H$ such that $P_i' = H \cdot P_i \ \forall i$

1. For each correspondence, create $2 \times 9$ matrix $A_i$
2. Concatenate into single $2n \times 9$ matrix A
3. Compute SVD of $A = U\Sigma V^T$
4. Store singular vector of smallest singular value $h = v_i$
5. Reshape $h$ to get $H$

**Cons of using DLT**: Note that the points here works with applicability of Homography

1. Only applicable when the actual transformation between the two images is linear. If that is the case then this may not work out well
2. Sensitive to scaling in pixel space. (We can add a normalization step)
3. Does not deal well with outliers (i.e. bad matches). Can be more robust with RANSAC

**Applicability of Homography**:

1. The scene is **planar or approximately planar** (i.e. scene is very far away or has small (relative) depth variation.
2. Scene is captured under camera rotation only (no translation or pose change)

**Normalised DLT**:

1. **Normalisation of $p$**: Compute a similarity transformation $T$, consisting of a translation and scaling, that takes points $p_i$ to a new set of points $\tilde{x}_i$ such that the centroid of the points $\tilde{x}_i$ is $(0, 0)$ and their average distance from the origin is $\sqrt{2}$
2. **Normalisation of $p'$**: Compute a similar transformation $T'$ for the points in the second image, transforming points $p_i'$ to $\tilde{p}_i'$
3. **DLT**: Apply standard DLT to the correspondences $\tilde{p}_i \leftrightarrow \tilde{p}_i'$ to obtain a homography $\tilde{H}$
4. **Denormalization**: Set $H = T'^{-1}\tilde{H}T$

**RANSAC**
**Algorithm**:
Loop until total number of iterations reached

1. Randomly get (min 4) correspondences
2. Compute $H$ using DLT
3. Count inliers. Note that we may have to consider squaring the distance to ensure that we are getting inliers from both sides.
4. Keep $H$ if largest numbers of inliers

For the best $H$ with the most inliers, recompute $H$ again using all the new inliers.
**Parameters**:

1. $\delta$ – Distance threshold to consider as an inlier
Should be large enough so that noisy inlier points will still lie within the threshold
2. $N$ – Number of iterations to run RANSAC loop

$$N = \frac{\log(1-p)}{\log(1-(1-e)^s)}$$

$e$ – Prob that point is an outlier, $s$ – Number of points we draw per iteration for fitting parameters, $N$ – Number of iterations in loop, $p$ – probability that we want to ensure that at least one set of points which we sampled does not contain any outliers

## Optical Flow

**Flow Methods**

| | Lucas-Kande Flow | Horn-Shunck Flow |
|---|---|---|
| **Applicability** | Can be used when image motion is large (Pyramid approach) | Suitable for video when image motion is small |
| **Assumptions** | Brightness Constancy – Colour Constancy and Small Motion<br>$I_xu + I_yv + I_t = 0$<br>Make use of $\partial x = [-1 \quad 0 \quad 1], \partial y = [-1 \quad 0 \quad 1]^T$ | |
| | **Constant Flow** – Assume surrounding pixels (e.g. $5 \times 5$) have the same displacement | **Smooth Flow Field** – Neighbouring pixels should have similar flow to the current pixel (don't have to be same) |
| **Method** | Featured based approach. Will only track visual features (corners, textured areas) | Recovers image motions at every pixel based on spatio-temporal image brightness variation |
| **Type of Flow Field** | Sparse motion fields due to the flow fields only detected at visual features but it is more robust at tracking | Dense motion fields due to flow fields detected at every pixel but it is sensitive to appearance variations |
| **Issues** | Changes in brightness across image. (i.e. Histogram stretching/equalisation will cause violation of brightness constancy). Therefore, cause the flow fields to change. | |
| | **Aperture Problem** – If we only have a small visible patch and a line, we wont be able to know what is the exact motion.<br>**Aliasing** – If there are many pixels with the same intensity, when the motion is too big, there could be wrong "correspondence". Solution: Coarse-to-Fine Estimation.<br>**Edges** – Edges that are detected will have perpendicular flows | $\lambda$ small → Care more about smoothness<br>$\lambda$ large → Care more about brightness<br>**Swirl Patterns** –Due to strong smoothing constraints<br>"**Apparent Motions**" – Could be due to shadows |

## Tracking

**Lucas Kanade (Additive) Alignment**:

1. Warp Image, $I(W(x;p))$
2. Compute error image $[T(x) - I(W(x;p))]$
3. Compute gradient of the warped image $\nabla I(x')$
4. Evaluate Jacobian $\partial W/\partial p$
5. Compute Hessian approximation $H = \sum_x [\nabla I \ \partial W/\partial p]^T \ [\nabla I \ \partial W/\partial p]$
6. Compute $\Delta p = H^{-1} \sum_x [\nabla I \ \partial W/\partial p]^T \ [T(x) - I(W(x;p))]$
7. Update parameters $p \leftarrow p + \Delta p$

**KLT Algorithm for Feature Tracking**:

1. Find corners satisfying $\min(\lambda_1, \lambda_2) > \lambda$ for the Hessian
2. Loop over corners
Compute displacement to next frame using the Lucas-Kanade alignment method.
Store displacement of each corner, update corner position.
(optional) Add more corner points every $M$ frames using step 1
3. Returns long trajectories for each corner point

**Tracking Parameters**:
$\eta$ – Controls how much we want the new value to affect our template. When $\eta$ is large, it means that we want more of the new values.
**Tracking Algorithms differ by**:

| Property | Approaches |
|---|---|
| Representing Candidates | **Naïve**: Grey scale/ RGB<br>**Better**: Gradient Features, Histogramming, Deep Features |
| Search for Candidates | **Naïve**: Exhaustive set of locations. (Computation and efficiency concerns)<br>**Better**: Limit search space based on previous results. |
| Solving for mode-finding problem | **Naïve**: Take a simple global max (But could be hard to distinguish sometimes)<br>**Better**: Leverage previous locations to help with local max |
| Updating target's template | **Naïve**: Keep fixed, don't update but there could be appearance changes<br>**Better**: Update as the track progresses and using $\eta$ as a tuning parameter |

**Issues with Tracking**:

1. **Initialisation** – Often done manually. We can make use of background subtraction, detection can also be used
2. **Catastrophic errors** – Occlusions & clutter, exit of frame, multiple objects
3. **Drifting** - Due to accumulation of small errors over time

**MOSSE Filter for Tracking**:

1. Initialise on frame $I_0$ with marked ROI $x_0$ (DFT → Optimise kernel $g$)
2. Apply kernel $g$ to frame $I_k$ (DFT)
3. Find the local maxima to locate target bounding box $x_k$ (iDFT)
4. Update kernel with new bounding box $x_k$ (DFT → Optimise kernel $g$)

**Note**: Initialisation is done on several perturbed version of initial ROI to add robustness to first kernel $g$. Kernel is solved in Fourier domain via discrete Fourier transform (DFT). Localization is in the image domain, so convert cross-correlation results back with inverse DFT (iDFT).

**Evaluation Metric for Tracking**:

1. **Accuracy** – Talks about how well the tracker's bounding box overlap with ground truth bounding box
Can make use of $|B_{gt} \cap B_p|/|B_{gt} \cup B_p|$ as one of the metric. We can average it over frames in sequence for per-sequence accuracy. [0, 1] where 1 is the best
2. **Robustness** – Talks about how many times the tracker fails (i.e. lose the target) (0 is ideal)
We can re-initialise the tracker (manually) each time it has failed to resume tracking. We will average over multiple tracking runes on a sequence to see how many times it fails on average.