1)

ai) hx = [1 0 –1]  
 [1, 0, -1]  
 [1, 0, -1]   
 hy = [1, 1, 1]  
 [0, 0, 0]  
 [-1, -,1, -1]

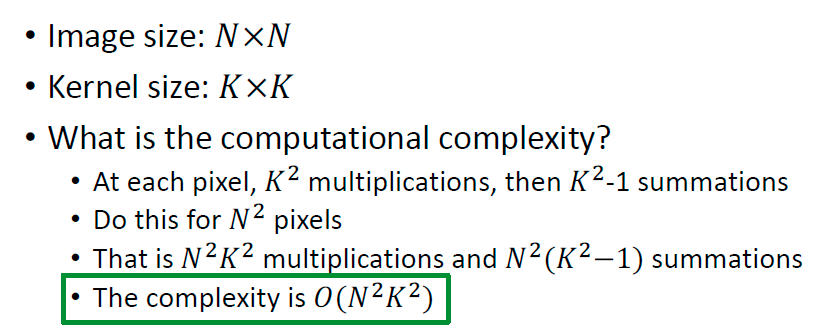
aii) Flip the kernels (horizontally and vertically) and perform ~~matrix multiplication~~ convolution  
  
gx = [3, 0, -3]

[6, 0, -6]  
 [9, 0, -9]  
  
gy = [4, 6, 4]  
 [4, 6, 4]  
 [4, 6, 4]  
  
aiii) It can be separated into two 1D filters such that we have smoothing followed by finite difference:  
  
[1  
 1  
 1]  
  
and then [1, 0, -1]  
  
Multiply them together and you basically get what you want  
  
aiv) Gaussian smoothing is performed prior to performing finite difference. This will smooth out a noisy signal so that when we calculate the derivative (i.e. using finite difference of a prewitt filter) it will produce a clearer peak signal and it will be easier to determine the maximum.

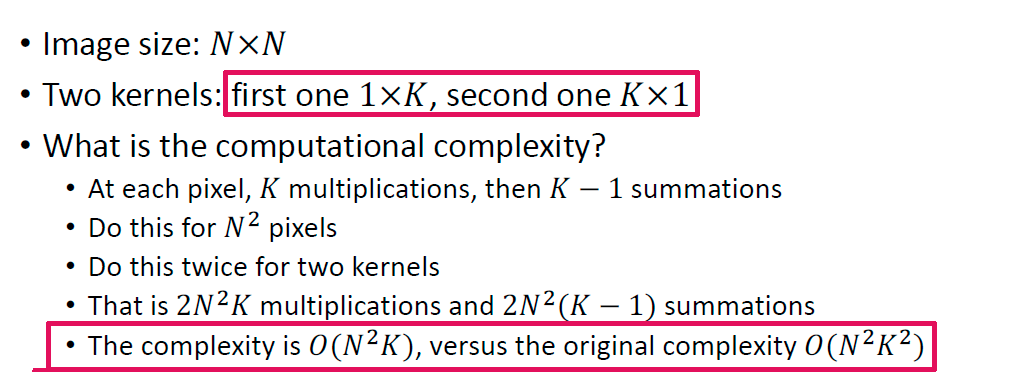
Also, since the perwitt filter contains a finite difference component (1 0 –1) this is the derivative part. This part is sensitive to noise, therefore we choose to smooth before due to the above reason.

bi) ~~K should be around 3 or 4 as beyond this the gaussian function gives values which are reallym small and in the grand scheme of things are insignifcant to contributing to the pixel values~~

[- sigma \* k, sigma \* k] where k is 3 or 4 from lecture slides, so in total it would be (2\*sigma\*k+1)x (2 \* sigma \* k + 1)

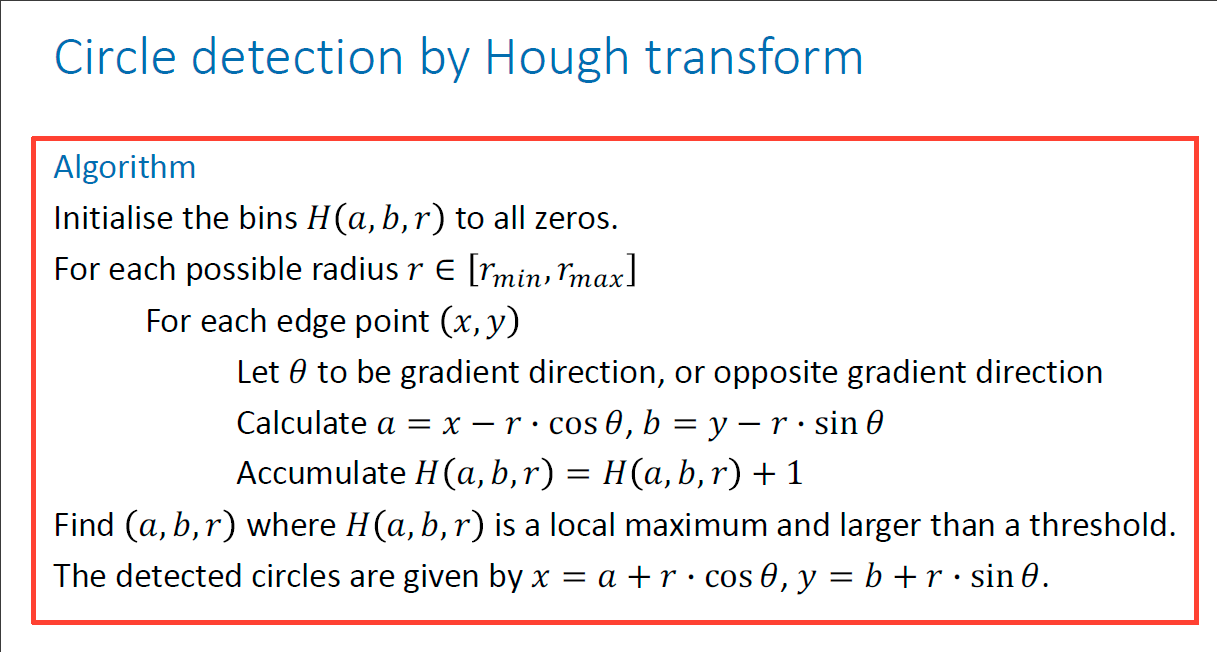
bii) For 2D convolution 

For 1D convolutions one after the other



ci)

* Using points detected by an edge map
* Attaching linear equation to match as accurately as possible to the edge of an image
* Each point can vote for a parametric line which fits the best and represents it
* The one with the largest vote “wins” and is considered the parametric edge line

cii) 

2)

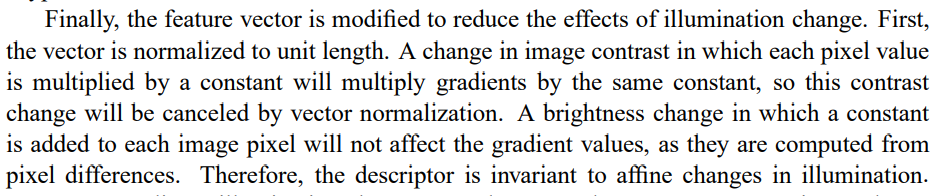
a) First three steps of the algorithm.

* Detection of scale-space extrema using DoG filter across many scales
* Keypoint localisation by fitting a quadratic function to the DoG response of neighbouring pixels and finding a refined extrema
* Orientation assignment by calculating the gradient orientation of the pixels in the neighbourhood (pixels that correspond inside the blob detected by the first two steps) and creating an orientation histogram of 36 bins (10 degrees per bin) and the orientation is assigned by the majority bin
* Keypoint descriptor – HOG

b)

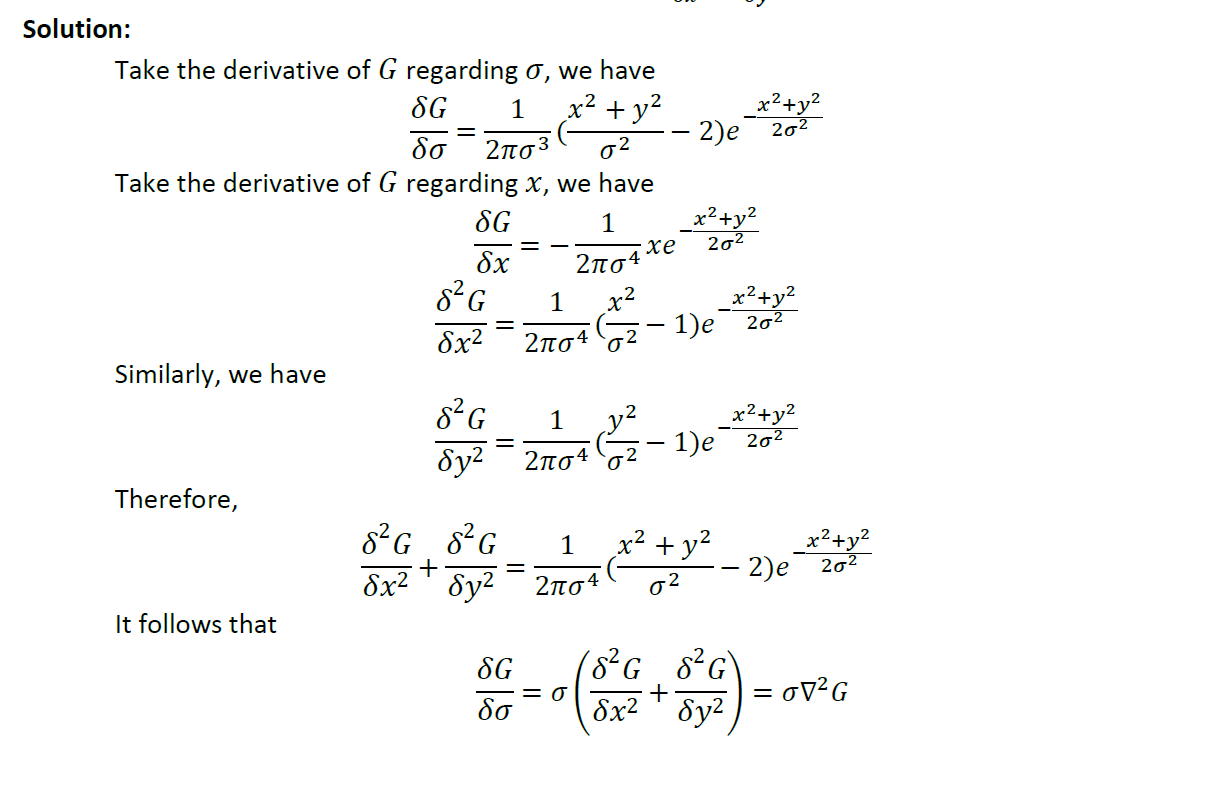
* Feature vector describes each of the subregions in detail
* Each subregion has 8 bins associated to it, which together describe the orientation histogram for that subregion
* Each bin represents an orientation and has a magnitude associated with it
* As such we have 16 x 8 values as a feature descriptor

The feature vector stands for the concatenated normalised (L2 normalisation on the concatenated histograms) histogram in SIFT

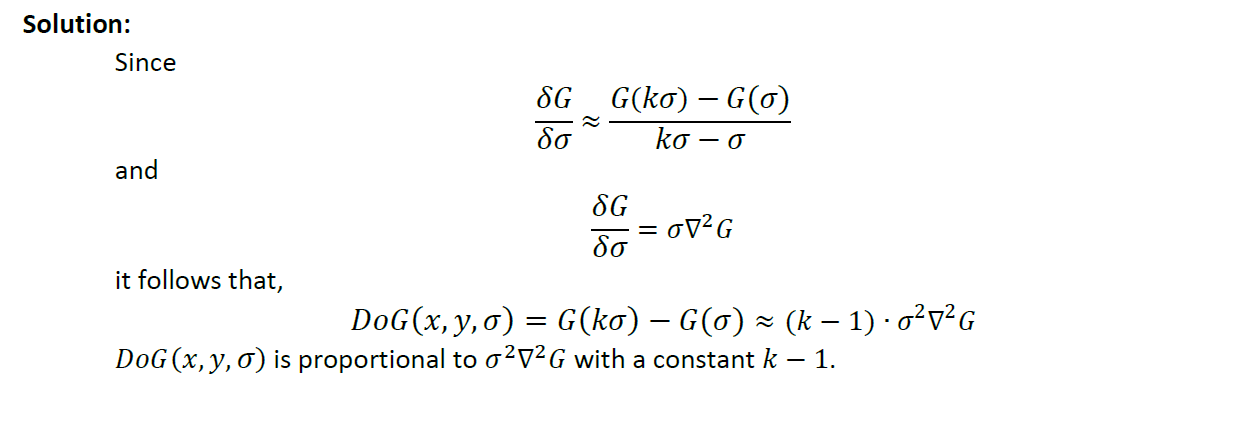
(From SIFT paper)

ci) Perform calculations to find nearest neighbour. Involved calculating Euclidean distance between SIFT operators and finding the minimum distance.  
  
cii)

* For the nearest neighbour we must evaluate distance between each point in original image and other image resulting in NM distance computations. Each computation uses a D dimensional descriptive vector and hence can be
* said to have computational complexity of O(D). Hence computational complexity of the whole nearest neighbour algorithm is O(NMD). Computational complexity can be decreased by using approximate nearest neighbour or by decreasing the dimensionality of the descriptive vector D.
* Use RANSAC which aims to maximise the number of inliers and avoids the mapping overfitting to outliers

di) 

dii)

  
3)

A)

Convert images to grayscale

Ensure the tower is centered in the image, to properly detect features

remove noise by performing Gaussian smoothing

Normalise the size of the images to same size (Block normalization to reduce lighting effects?)

Make sure scales are comparable, since if the tower is very small or very big, likely that training will not be very good => the idea is for HOG to be able to infer the most general features, which is where the line will be fitted

Rotate image (similar to “Slant Correction”) to ensure tower can be detected from different angles

B) pixel intensities are sensitive to absolute intensity changes, reducing their usefulness for classification compared to HOG which is robust to rotation and intensity changes. HOG is rotation invariant as it uses histograms which have this property.

C) Support vector machine. (description from slides)

D) build image pyramid from test image consisting of multiple scaled versions of the test image. Process all pyramid levels using the SVM and if the Queen’s Tower is detected in one level, we return an overall true result.

Alternative solution: use different size blocks when building histograms, e.g. first use cells of KxK, where each block you just have a longer feature vector. Train on all these features.

Need to be consists of 4 cells, then use 2Kx2K etc. Concatenate all histograms in a normal way, but no careful, however, to ensure that the vector doesn't become too long as it may impact performance (curse of dimensionality). **Any thoughts guys?**

(From EdStem:)

“One idea to solve this problem is using a RPN (Region Proposal Network) along with the SVM classifier. RPNs can generate region proposals of different sizes and aspect ratios, which allows the SVM to detect objects at different scales in the image. The proposed regions are then passed through the SVM classifier to verify whether they contain the Queen's Tower or not. Another benefit of this modification is that it reduces the number of windows that need to be processed by the SVM and therefore the detection speed is increased.”

E) use non-max suppression by considering the objectness scores of each overlapping region and only retaining the result with the highest score.

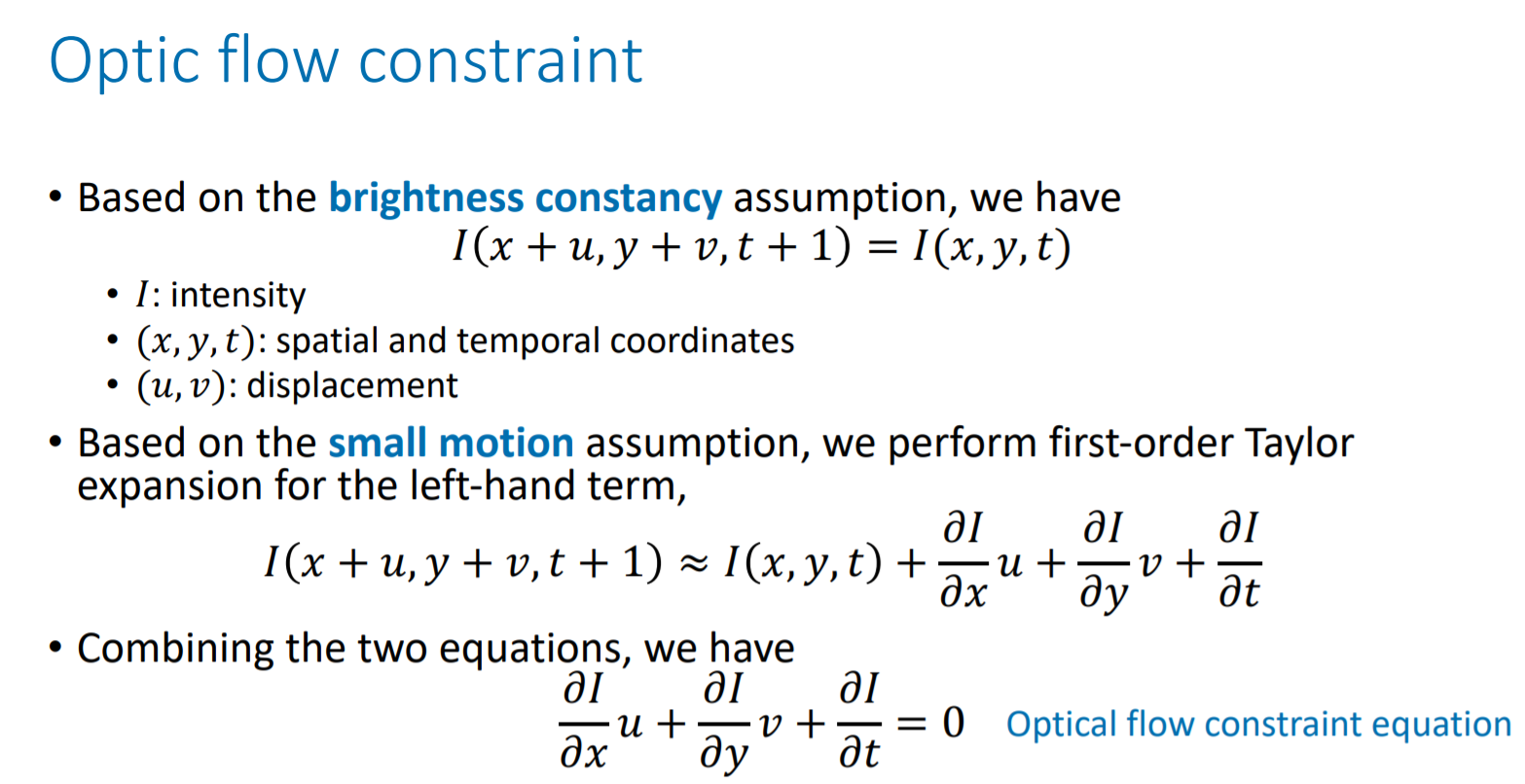
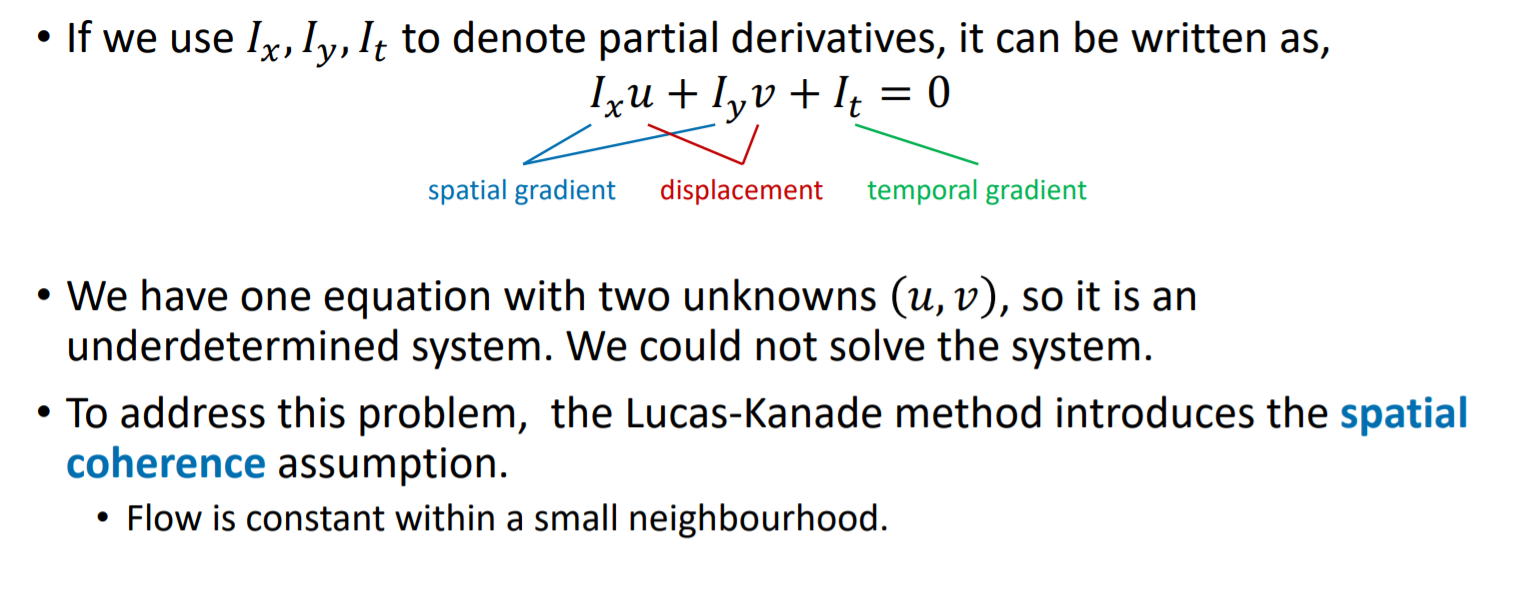
F)

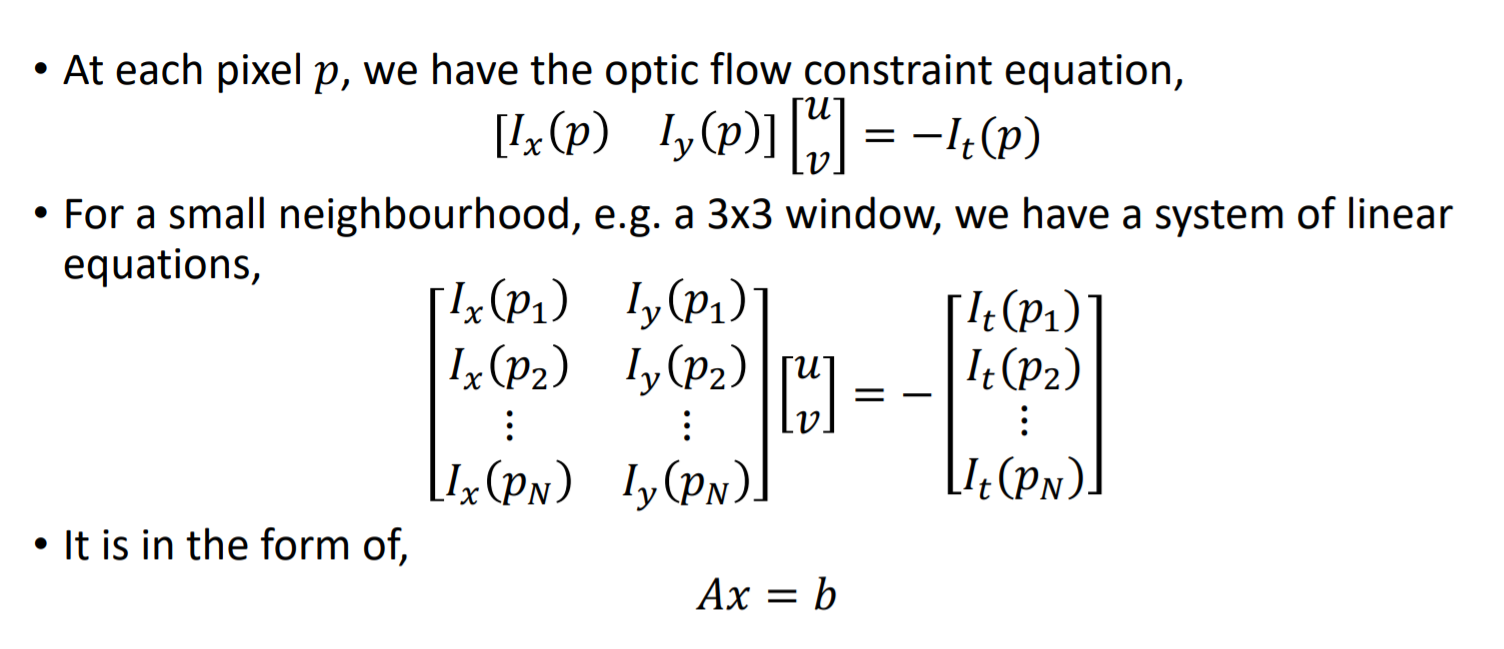
Classification accuracy: use proportion of true positives and true negatives out of all detector predictions. Higher proportion means higher accuracy.

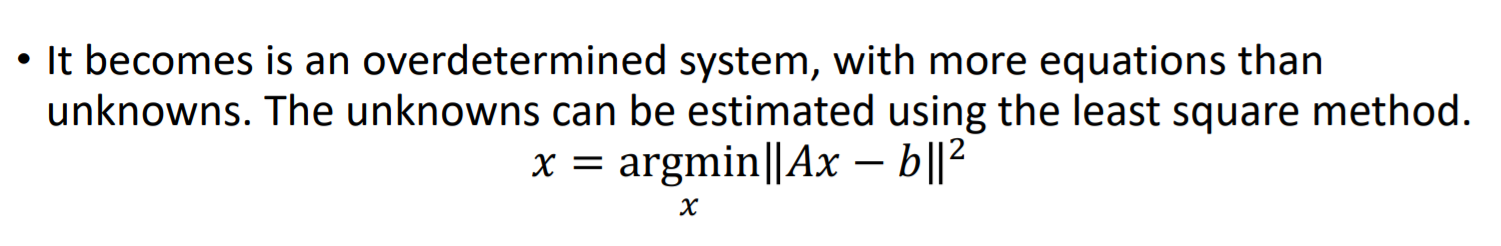
Localisation accuracy: for each prediction, compute the transformation from the anchor to the ground truth bounding box, t, and also the transformation from the anchor to the predicted bounding box, t\*. One metric we can use is the mean-squared error between t and t\*. Lower MSE means higher accuracy.

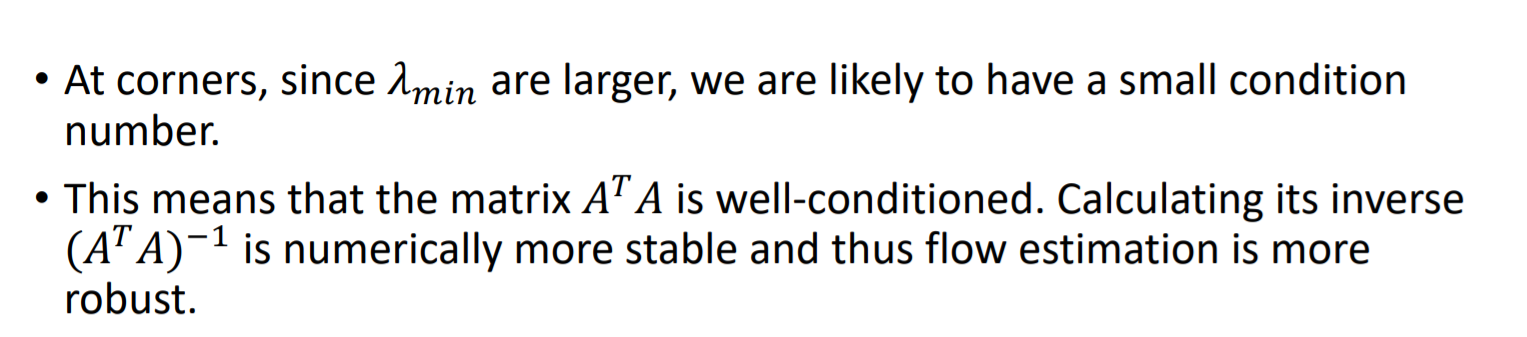
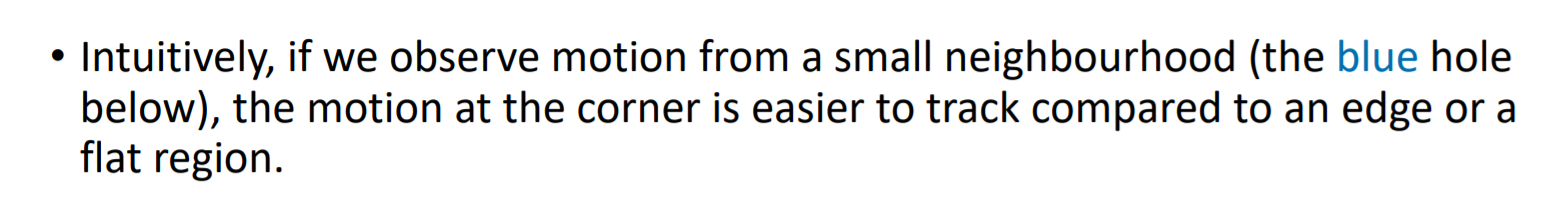
Alternative: Calculate the MSE between the overlapping area of the predicted bounding box and the actual bounding box (ground truth)

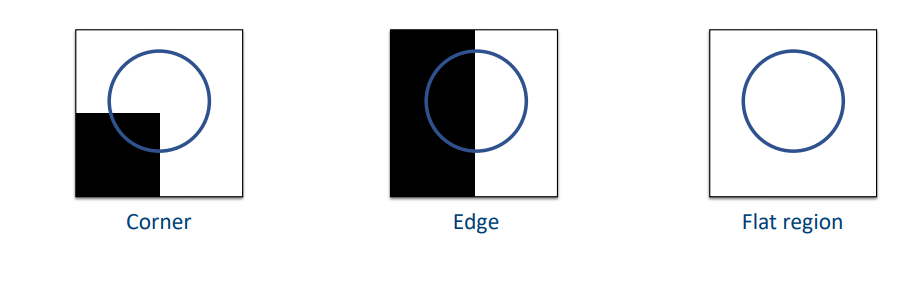
It does not ask for a *loss* as instead for an *accuracy* score, so a percentage of the overlapping area from the total predicted/actual bounding boxes would work?

4) a. 1) Brightness Constancy (Pixel has constant brightness across time)  
 2) Small Motion (Between frames motion is small)  
 3) Spatial Coherence (Pixels move like their neighbours)  
  
  
b)  




  
c) The corner because:

  
  
z

**ALTERNATIVE**: is the question not clearly referring to the ”Aperture Problem”? Yes corner, but should reference and describe the ”Aperture Problem” imo

d)

We can define a fixed size window on the video timeline and slide this window down the timeline. For each window, we calculate the optic flow field between adjacent frames using the Lucas-Kanade method, for example. Then we calculate the average of all flow fields in the window. Starting from the second frame in the window, we warp each frame in turn from the previous frame using the averaged field.

The window size should be kept small so that the overall motion within the window is small and hence more likely to be close to the smoothed version.

Alternative:

1. Estimate the optic flow of consecutive frames using the Lucas Kanade method
2. Warp each adjacent frame to the current image respective to the current images' optic flow fields
3. Denoise by calculating the average between the current and warped adjacent frames