**Computer Vision**

Computer vision is the automatic interpretation of visual data using computers.

Image acquisition   
(hardware)

Image processing   
(low level vision)

Image analysis/feature extraction (mid-level vision)

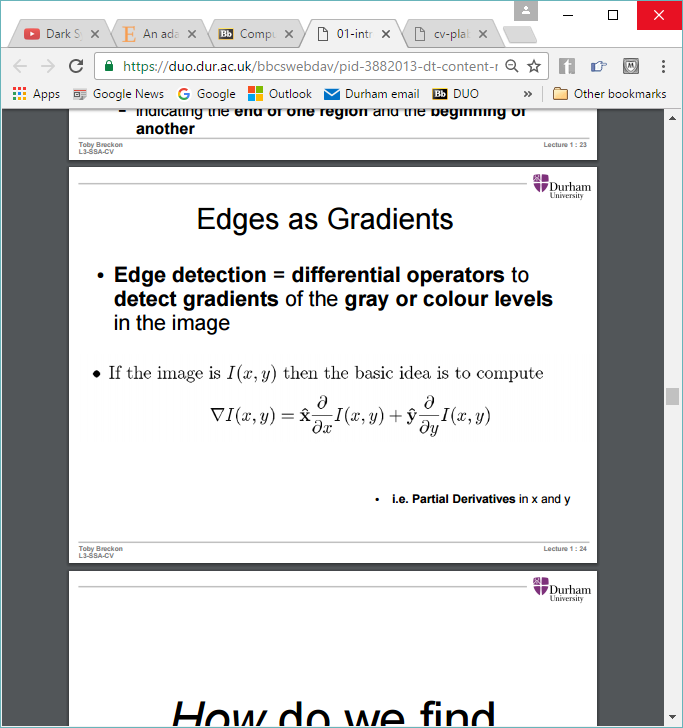
Image understanding/object recognition and reasoning   
(high level vision)

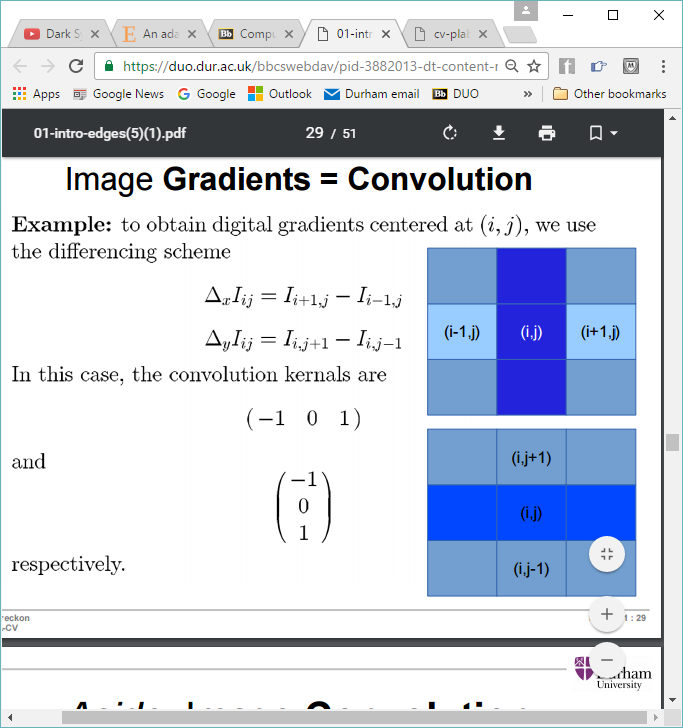
Further processing   
(Resulting action/behaviour of system)

Computer vision pipeline:

**Edges**  
Edges are useful in feature extraction. These outlines of objects can be used in rocognition or perception of distance and orientation.

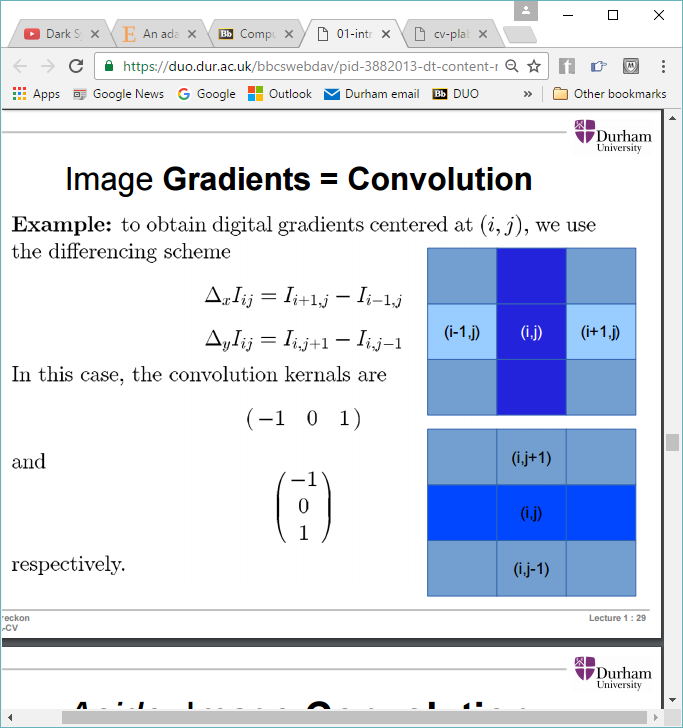
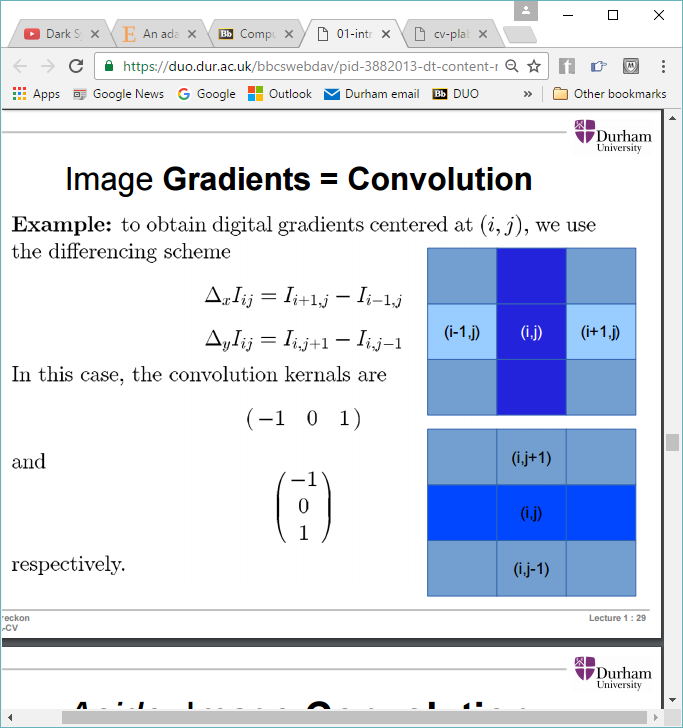
An edge can be identified as an image feature where there is an abrupt change in intensity, indiccating the end of one region and the beginning of another.

If an image is I(x, y) then the basic idea is to compute:

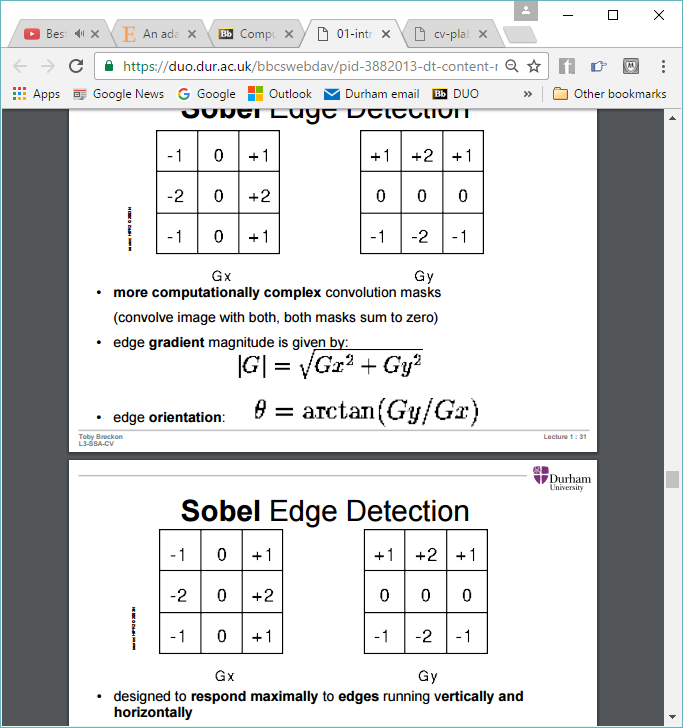
i.e. how much change in the row/column at this point.

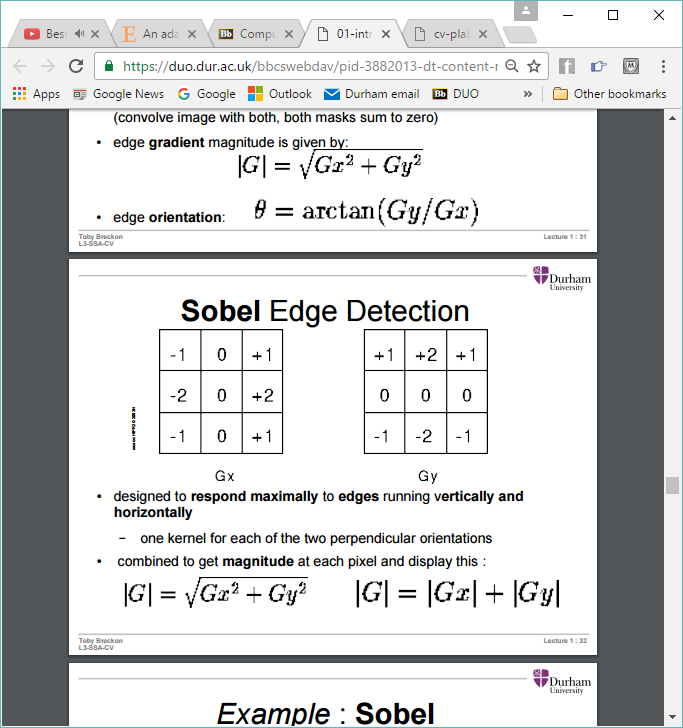
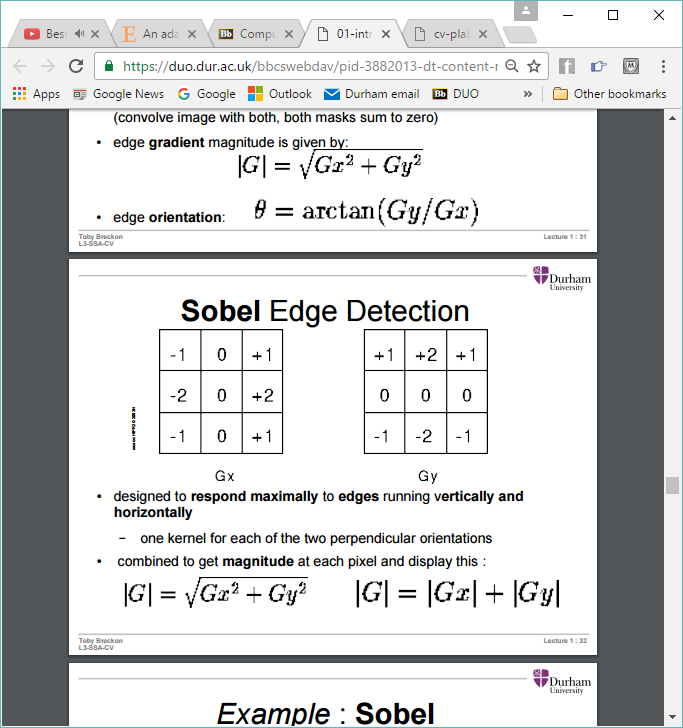
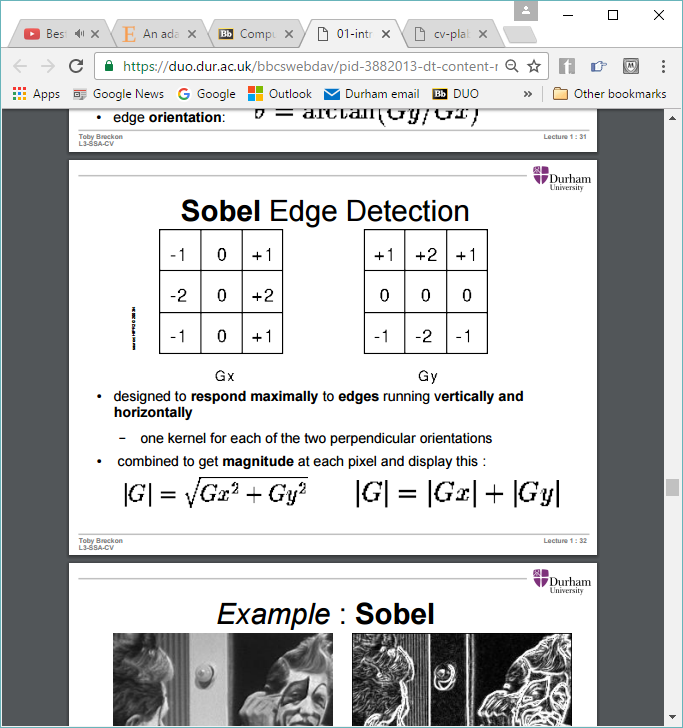
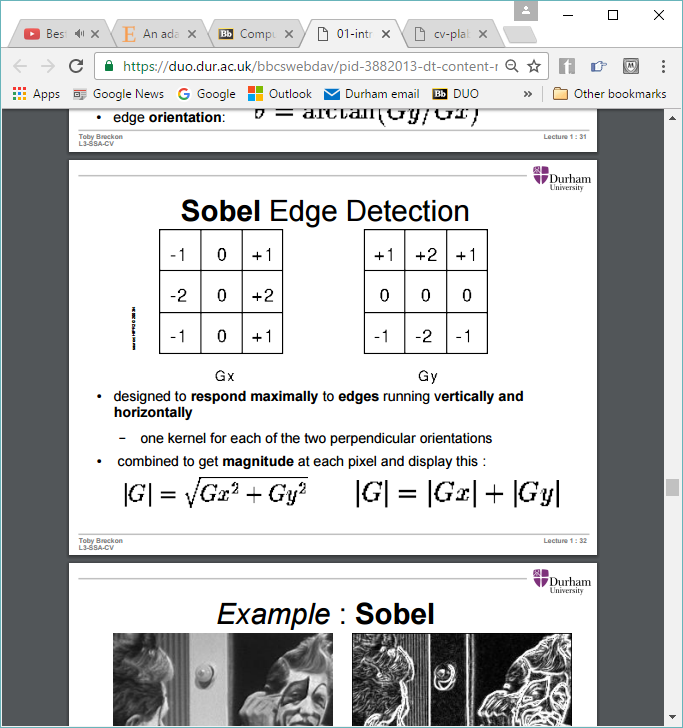
Based on the above formula, image gradients centred at (i, j) can be found by replacing the partial derivatives with differences:

Which uses the convolution kernels:

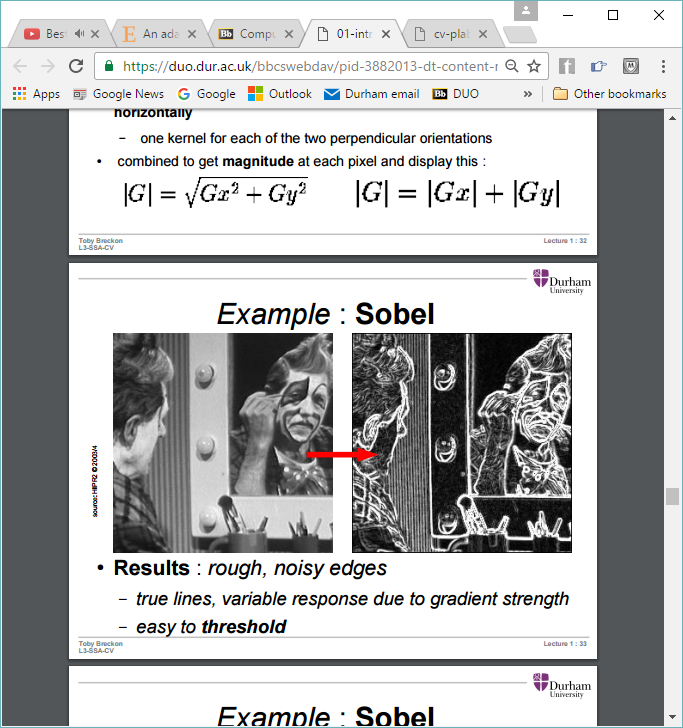


**Sobel edge detection**Uses more computationally complex convolution masks that together combine to make 0. This means that once both the masks have been applied, if there is a large difference in intensity (edge) in either direction then the result of the applied masks will be high. If the intensity is very similar then the result will be low. It is designed to respond maximally to edges running vertically and horizontally – one kernel for each direction.

  
Edge magnitude:



Edge orientation:

  
The resulting detected edges are rough and noisy. The edge width and strength can vary.

However it is a quick and easy edge detection method and allows for thresholding (although does not always benefit). It can form the basis for advances edge, shape, or object detection. Also good at identifying textures.

**Canny edge detector**The first modern edge detector still used today. It is based on the first derivative of Gaussian. The four main stages are:

Gaussian smoothing – noise/small edge removal

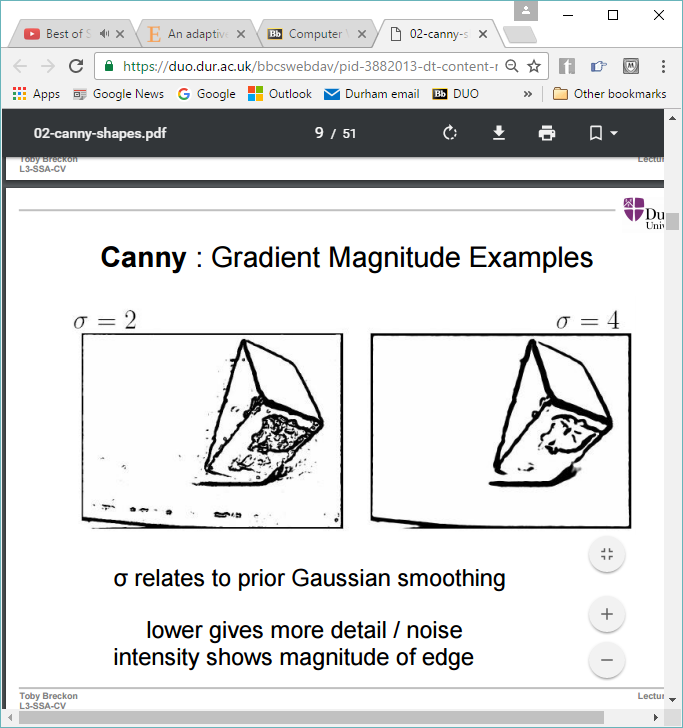
Gradient calculation – locate potential edges

Non-maximal suppression – locate “best” edge points

Hysteresis edge tracking – locate reliable but weak edges

**Gaussian smoothing**This step smooths noise without blurring the edges too much.   
Take an image I(x, y), applying Gaussian smoothing gives G(x, y).

**Gradient magnitude calculation**  
Compute the local derivatives in the x and y image directions. In practice this can be done by Sobel edge detection. The detected edges are determined by the kernel size in the Gaussian step. The lower the kernel, the more detail and noise. The intensity shows the magnitude of the edge.

The direction at each edge is also calculated. This value is rounded to either 0o, 45o, 90o, or 135o – up, horizontal, vertical, or diagonal respectively.

**Non-maximal suppression**An edge thinning technique that identifies where exactly the edge is. This is done by suppressing lower gradient values by checking across the gradient itself, revealing the peak of the gradient. First the gradient of the current pixel is looked up, then that is compared to the gradient of the pixels in the positive and negative directions of this pixels gradient. If the gradient of the current pixel is greater than the other two, then it is left alone, otherwise it is suppressed.

This works because if you have a very thick edge line going in a certain direction, then the gradient of intensity change will be perpendicular to this line. So the centre of the line (what we want to find) will be the pixel that has the highest gradient magnitude.

So if the above is an edge segment, the direction of each intensity gradient will be vertical. So if there is a pixel in the centre line, it will be compared with the pixels above and below. Since the centre pixel has the greatest gradient magnitude, it will be kept. A pixel in the dark or light grey part of the edge will be made white, because it has a lower gradient magnitude than the black line.

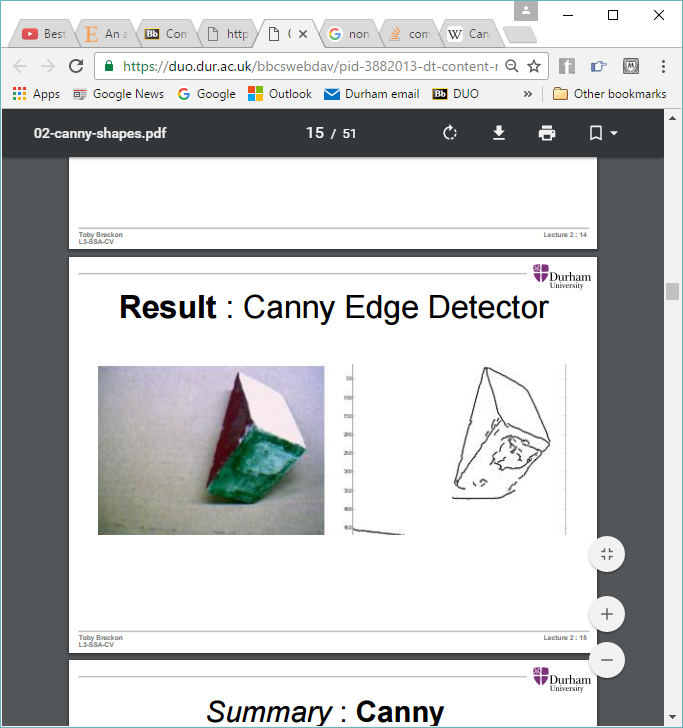
**Hysteresis tracking**identifies edge sections with components between two thresholds i.e. the strongest edges. It removes false edge fragments or texture noise.

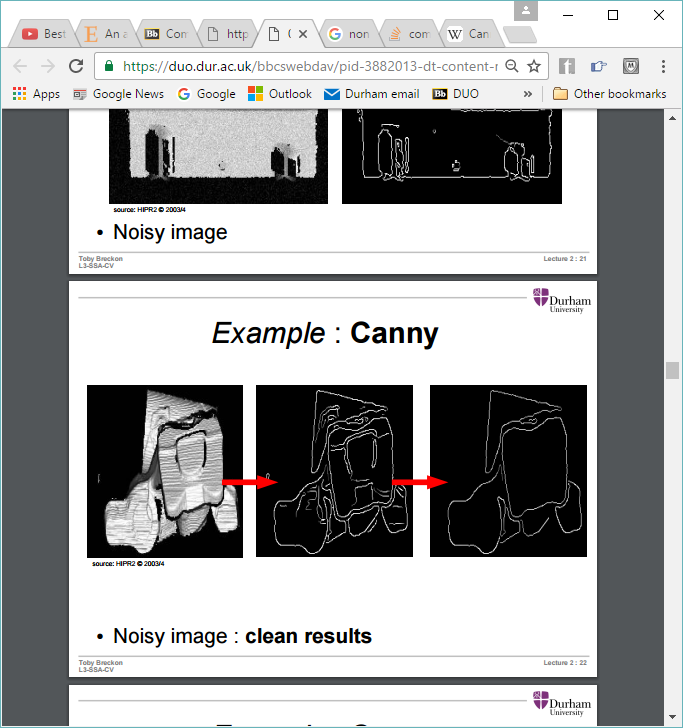
Identifies a pixel with a gradient magnitude H > Tstart+ (a strong edge)

Find any neighbours with H > Tcontinue (lower than Tstart)

Connect these edge points

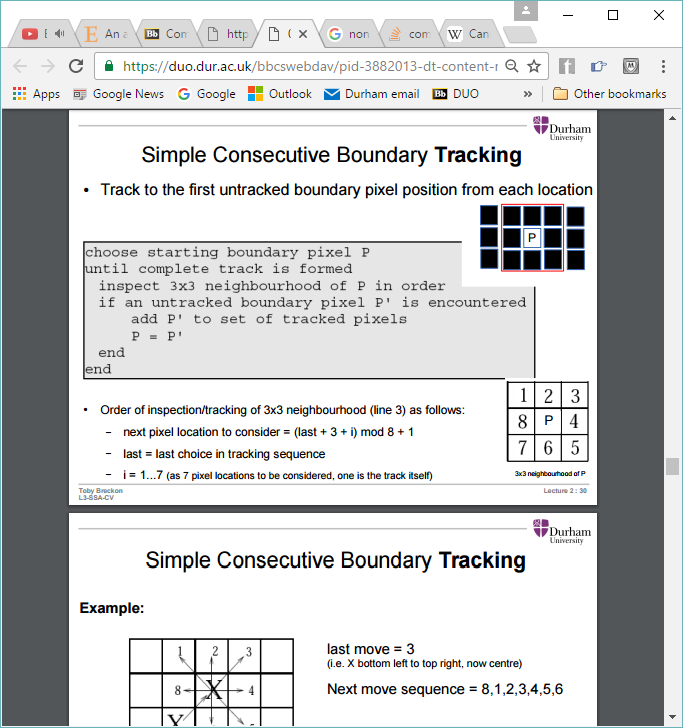
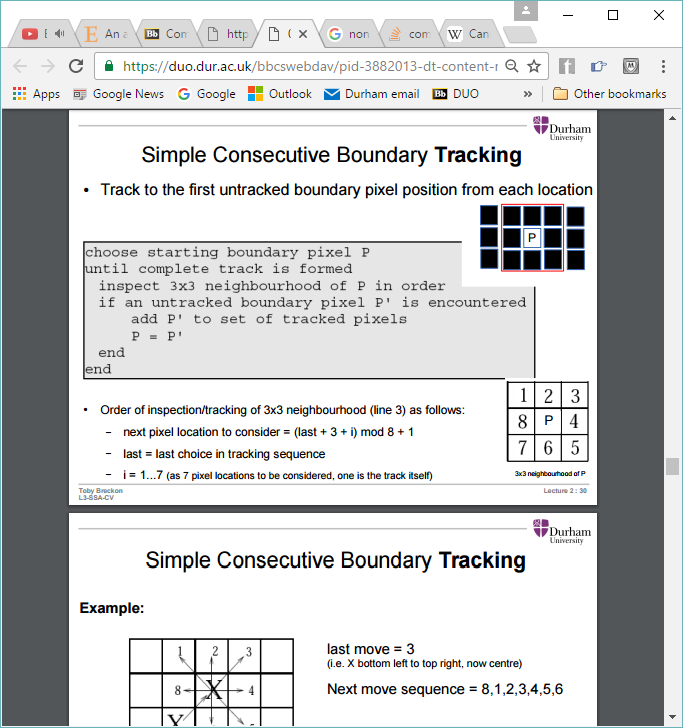
Any edge point < TContinue becomes 0.

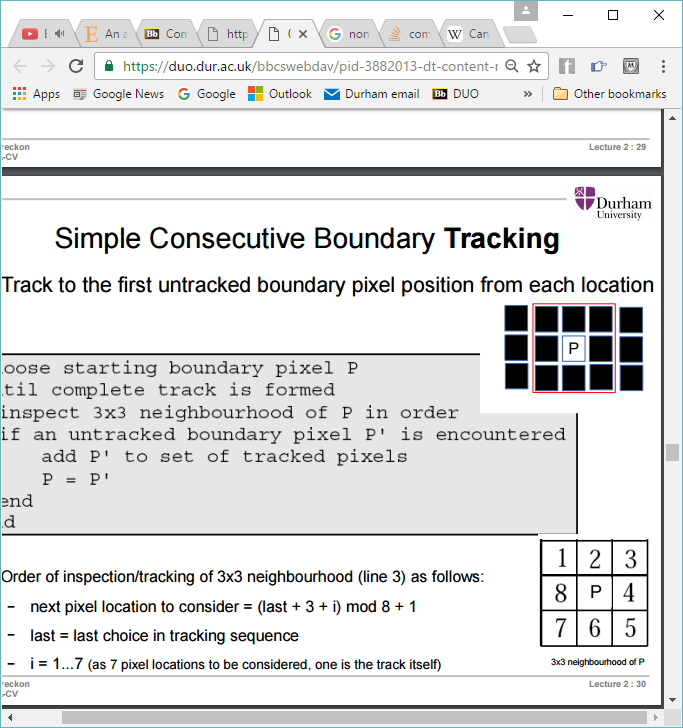
If Tstart is too low, end up with lots of small noisy edges.  
If Tcontinue is too high, noisy edges break up.

Produces clean edges from noisy images:

**Boundary detection**Edge operators detect local edge elements, not necessarily contours/boundaries. Boundary detection links the edges to meaningful boundaries.

**Simple consecutive boundary tracking**Track to the first untracked boundary pixel position from each location



So take p to be the first boundary pixel. Check around using the following pattern:   
i.e. check the pixel in position 1, then position 2…

If a pixel is found to also be an unidentified edge pixel (and hence part of this current boundary), that pixel becomes the centre pixel, and the search starts again.

The local search around the most recent boundary pixel starts from the pixel location based on this formula: *(last + 3 + i) mod 8 + 1*

where - last is the number that the new centre pixel was in the previous local search, and

- i = 1..7 (i.e. 7 pixels to check, not 8 because one of them is the pixel that we just came from)

For example, if the next pixel in the boundary is at position 3, p moves to 3, and we start the local search from (3 + 3 + 1) mod 8 + 1 = 8. Then increase i until another appropriate pixel is found.

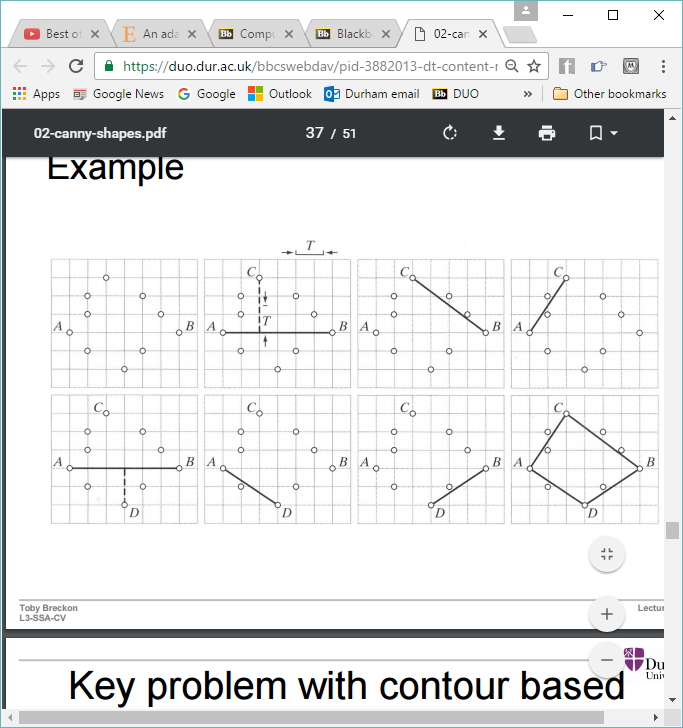
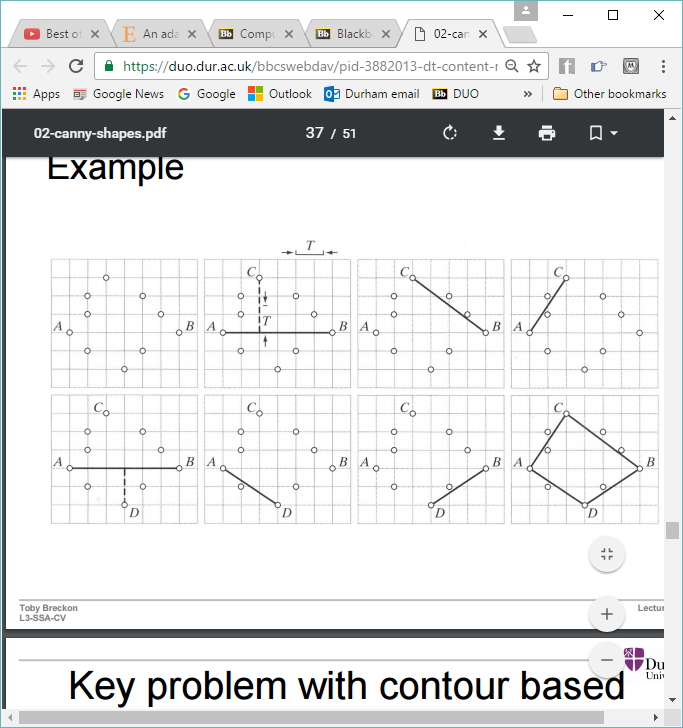
**Edge linking algorithm**Another algorithm for taking edges and identifying the ones that are contours. It assumes the edge points on the region’s boundary are known and ordered e.g. clockwise or anti-clockwise. From this a polygon approximating the boundary is computed, defined by the set of its vertices, a subset of the input edge points. The algorithm works as follows:

1. Specify two starting points A and B, a threshold T, and two empty stacks OPEN and CLOSED. All other points belong to P.
2. Identify whether the curve is an open curve or closed curve. An open curve is indicated by a large distance between two consecutive points A and B relative to the distance between other consecutive points, where A and B are end points   
     
     
     
     
     
   If the curve is open, push A into OPEN and B into CLOSED  
   If curve is closed, push A into CLOSED and B into OPEN

Closed Open

1. Compute the line passing from the top point in CLOSED and the top in OPEN
2. Compute the distance from that line to all the points in P whose points lie in between the two points at either end of the line. Select the point Vmax with the maximum distance Dmax
3. If Dmax > T, push Vmax at the OPEN stack. Go to step 3.
4. Else, pop from OPEN and PUSH it into CLOSED.
5. If OPEN is not empty, go to step 3.
6. End. The vertices in CLOSED are the vertices of the polygon.

Example



Steps 1 & 2. In reality, would be a lot more points

Steps 3, 4 & 5

Steps 3, 4 & 6

Steps 3, 4 & 6

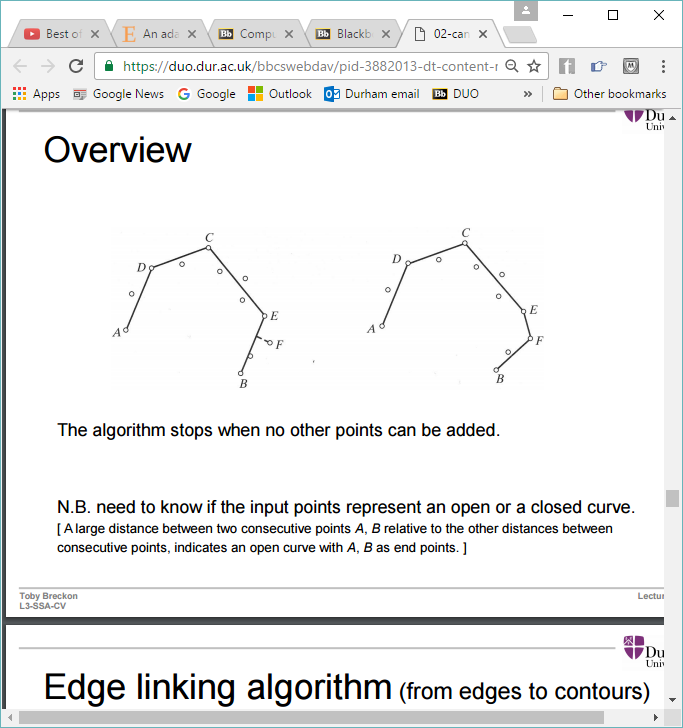
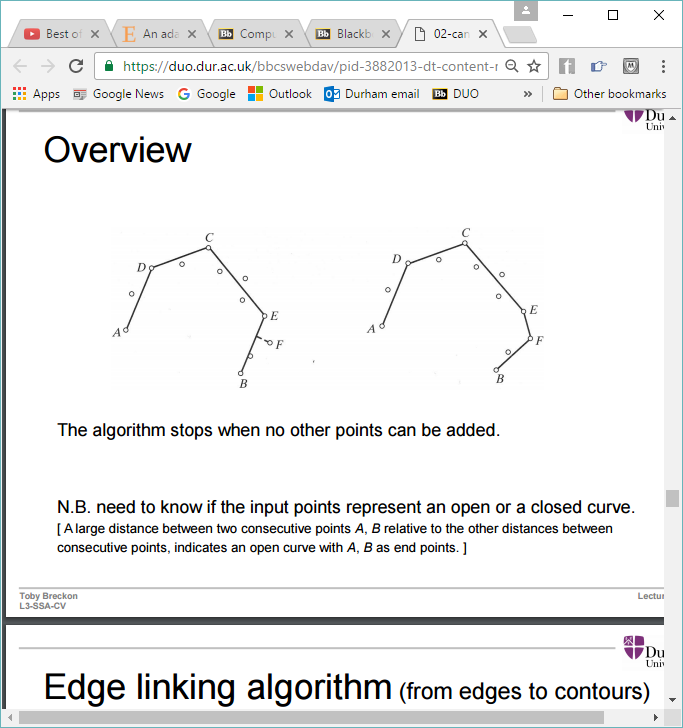
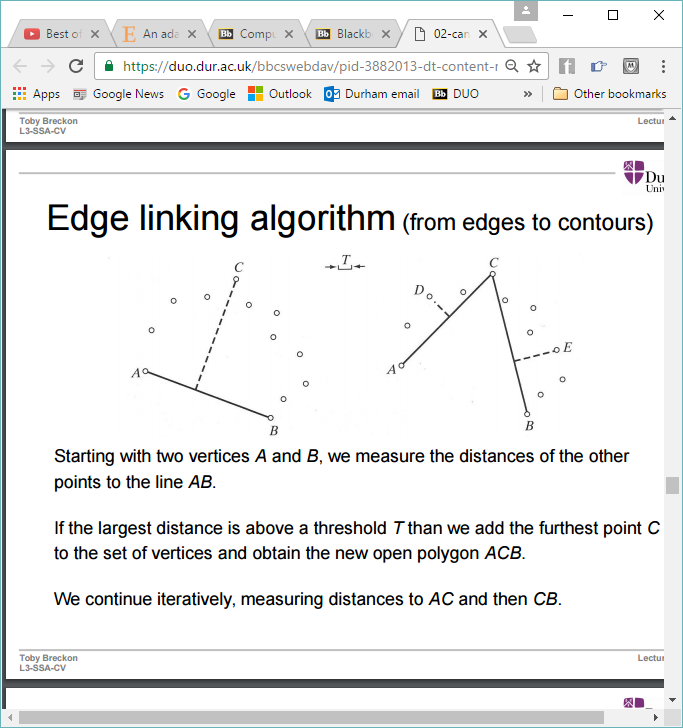
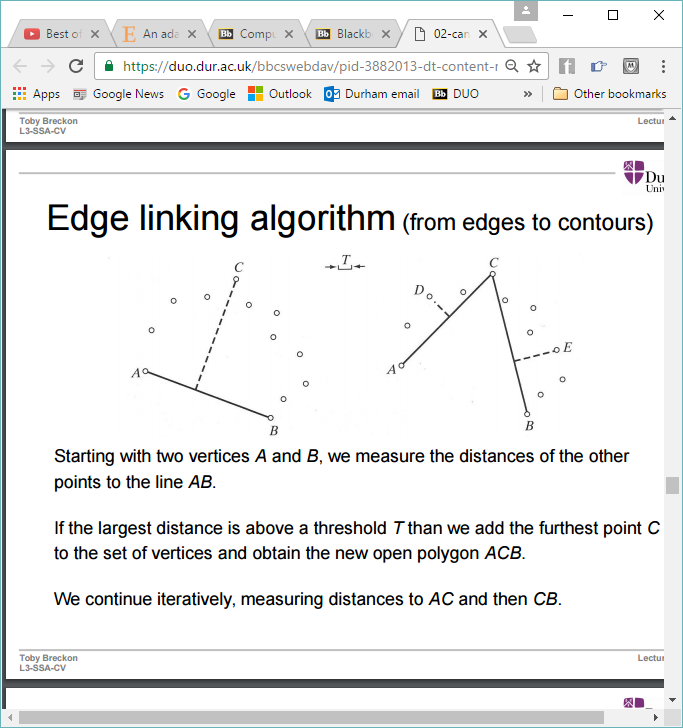
Steps 3, 4 & 5

Steps 3, 4 & 5.

Steps 3, 4 & 6.

Steps 7 & 8.

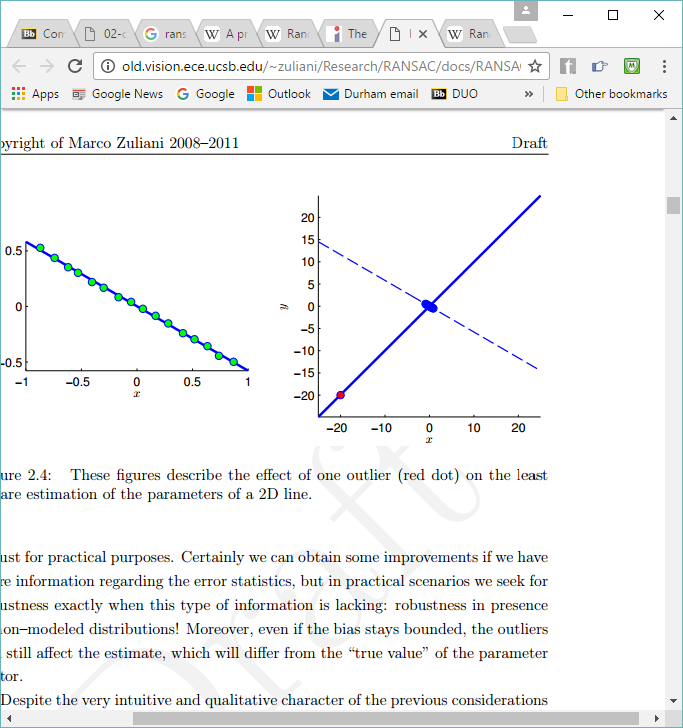
Another example



The main problem with contour based edge linking is that it doesn’t work well with noisy and complex scenes with noisy and complex edge patterns.

**RANSAC (random sample and consensus)**In general, RANSAC is a method to estimate parameters of a model from a set of observed data that contains outliers. So in the case of edges, it does not find the exact edges, but models them as straight lines. It is non-deterministic in that it produces a reasonable result only with a certain probability. Since it is an iterative method, the more iterations, the higher the probability. RANSAC is fairly robust to noise.

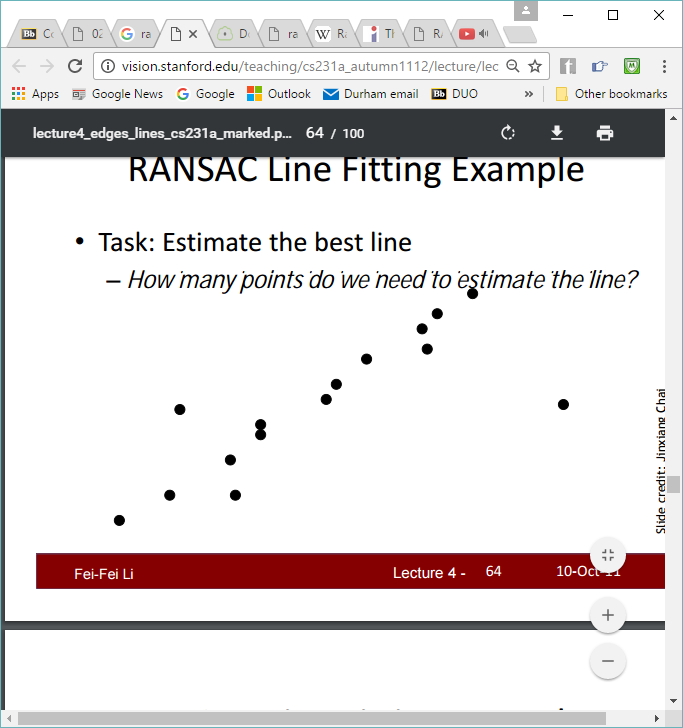
It is assumed that the data consists of “inliers” whose distribution can be explained by some set of model parameters, but may be subject to noise, and “outliers” who do not fit the model. The data must conform to a priori model i.e. the points that fit the model are true and not based on observation e.g. the edges of something are actually edges.

The effect of outliers can be seen in this graph. Red is the outlier, blue are the “true” values:

RANSAC can handle sets of data where the proportion of outliers is >50%. This is better that many other commonly used techniques for parameter estimation.

RANSAC method:

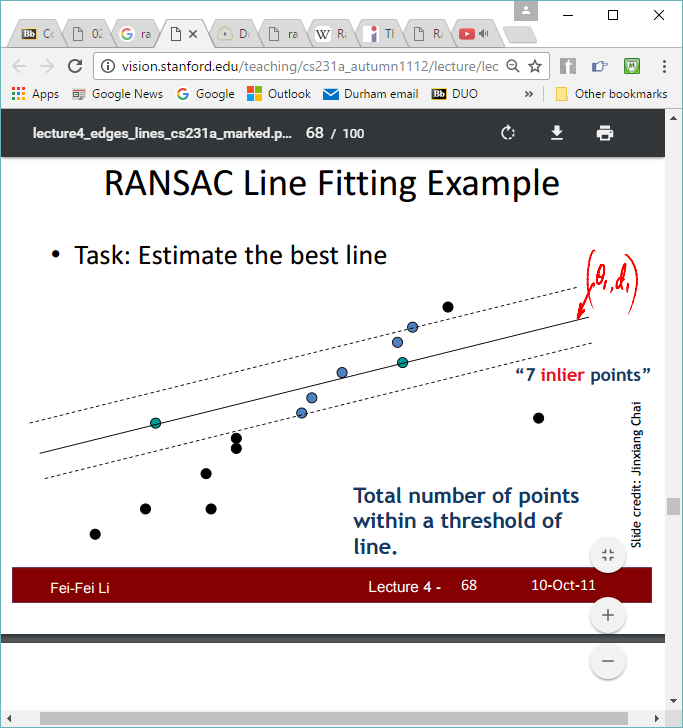
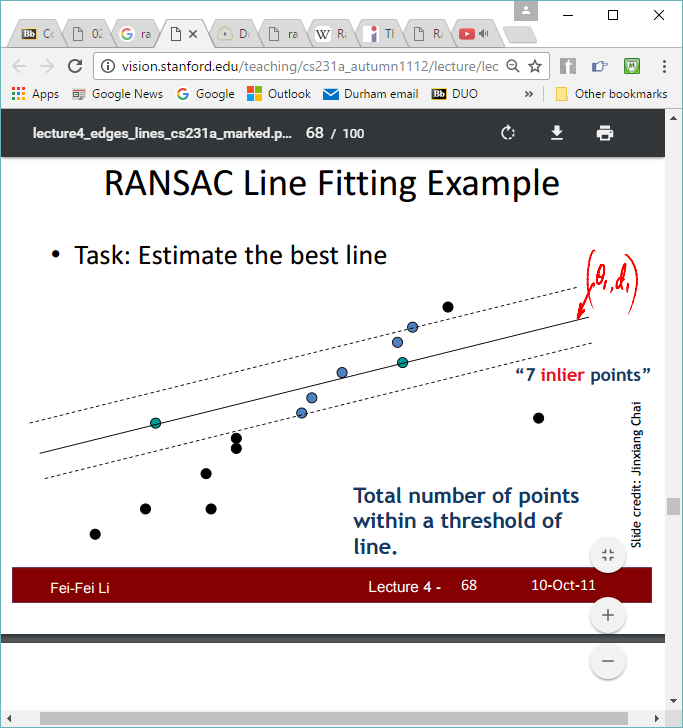
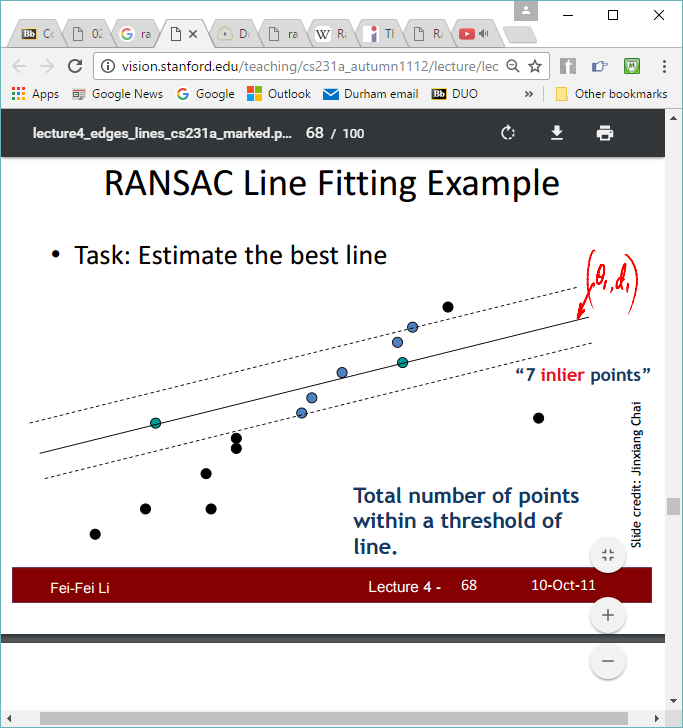
1. Select randomly the smallest subset of the original data required enable the model to be fitted
2. Fit a model M to this data
3. Determine how many points from the set of all points fit the model M. This is checked using a threshold value. This set of points that fit is known as the consensus set.
4. If enough points fit the model M well (determined by whether the ratio of the number of inliers over the total number of points in the set exceeds a threshold) i.e. the probability of finding a better ranked consensus set drops below a threshold. The model may then be re-estimated using the whole consensus set. Then terminate.
5. If not, go back to 1. This should only be done a certain number of times.

Example for points on a line:

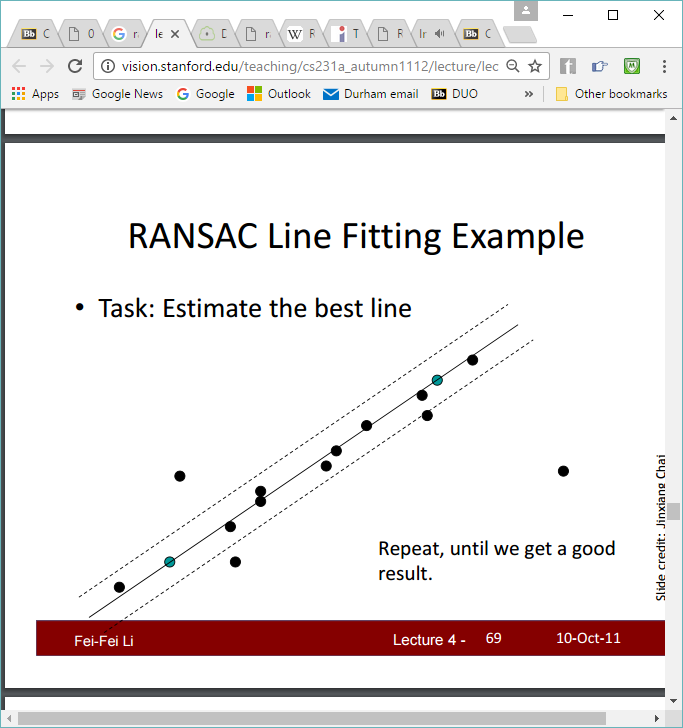
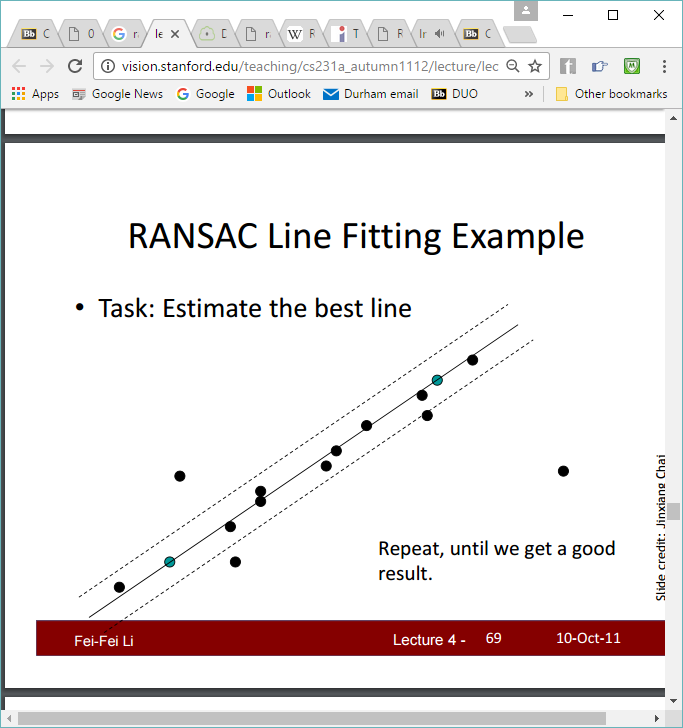


Try 2 points:

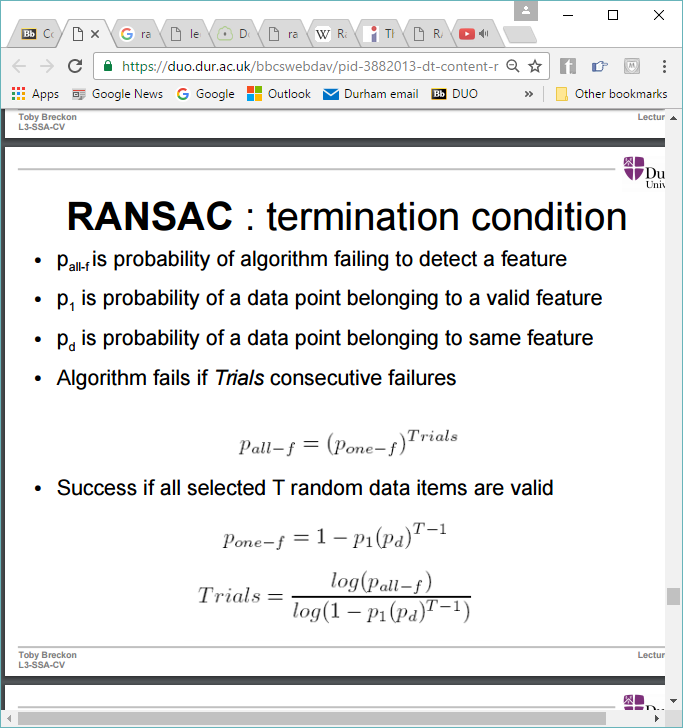
There are 7 points within the threshold of this line:



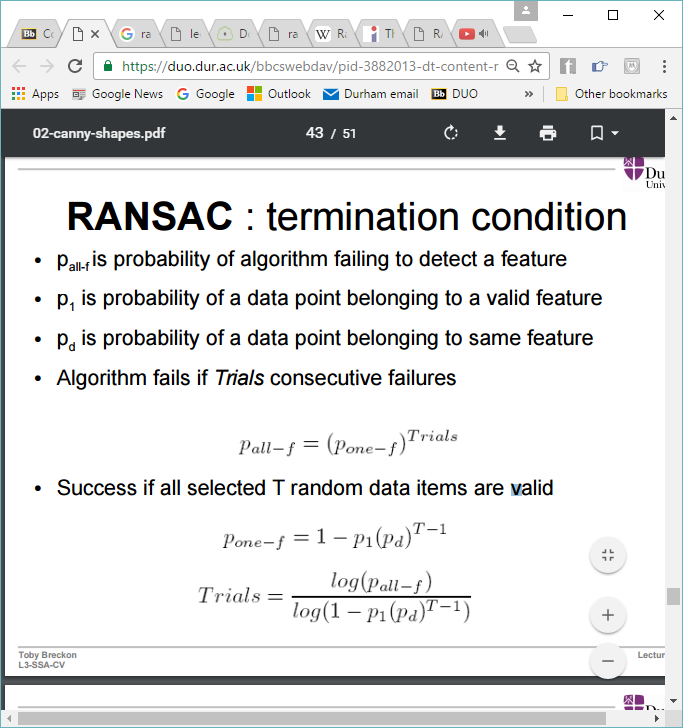
Repeat until we estimate the best line:



**RANSAC termination conditions**

* Pall-f is the probability of the algorithm failing to detect a feature
* Pone-f is the probability of one iteration of the algorithm failing
* P1 is the probability of a data point belonging to a valid feature
* Pd is the probability of a data point belonging to the same feature

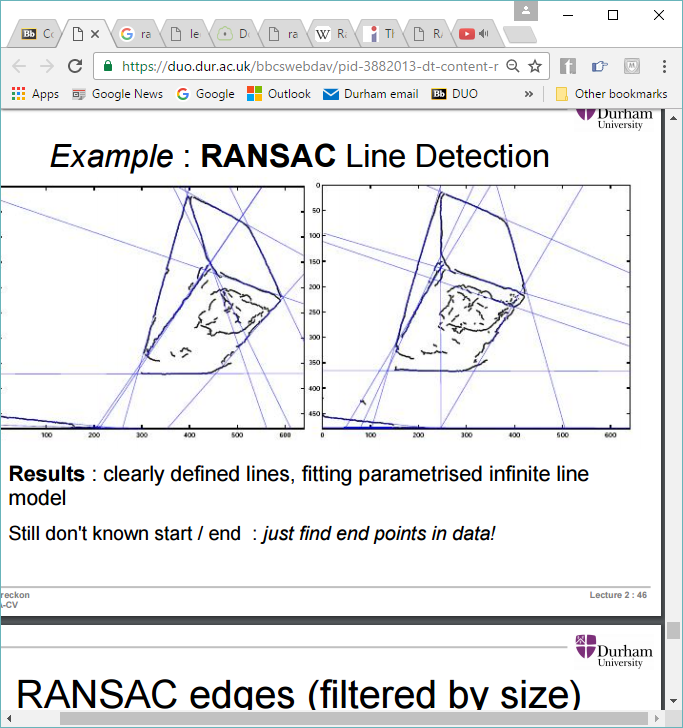
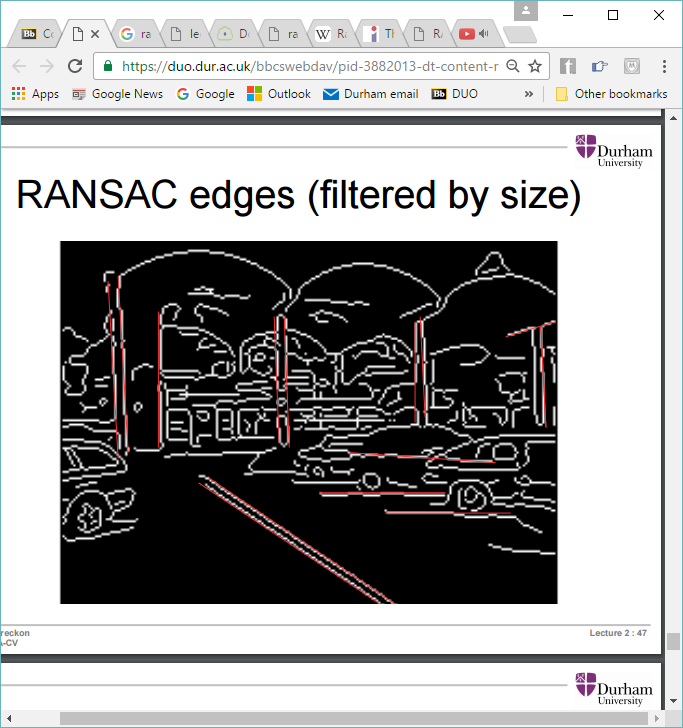
Algorithm fails if there are *Trials* consecutive failures:

Algorithm succeeds if all selected *T* random data items are valid:

**RANSAC for edge detection**

* The data used is a canny output image
* The model is an infinite line through 2 points
  + Need T = 2 edge points randomly chosen
* To check if the rest of the data supports it, check if it conforms to a line i.e. lies near a line
  + Accept if consensus set = 80 edge points within 3 pixels

If pall-f = 0.001, p1 = 0.1, pd = 0.01, then the number of trials = 688

Example of modelled lines found by RANSAC. Lines can be filtered by size to give just the major lines: