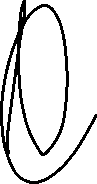


**Introduction to Computer Vision**



**Coursework**

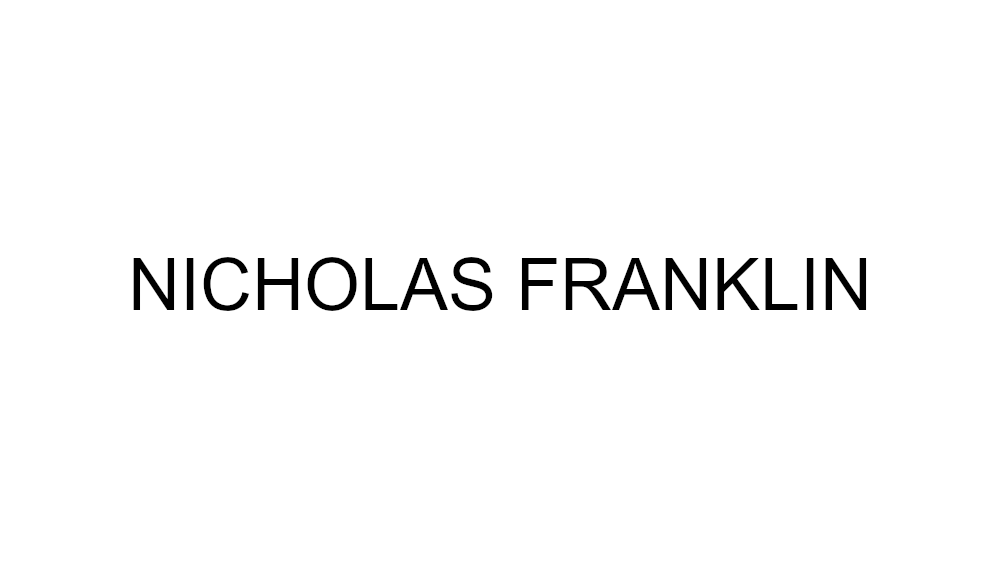
**Submission 1**



**Your name Nicholas Franklin**

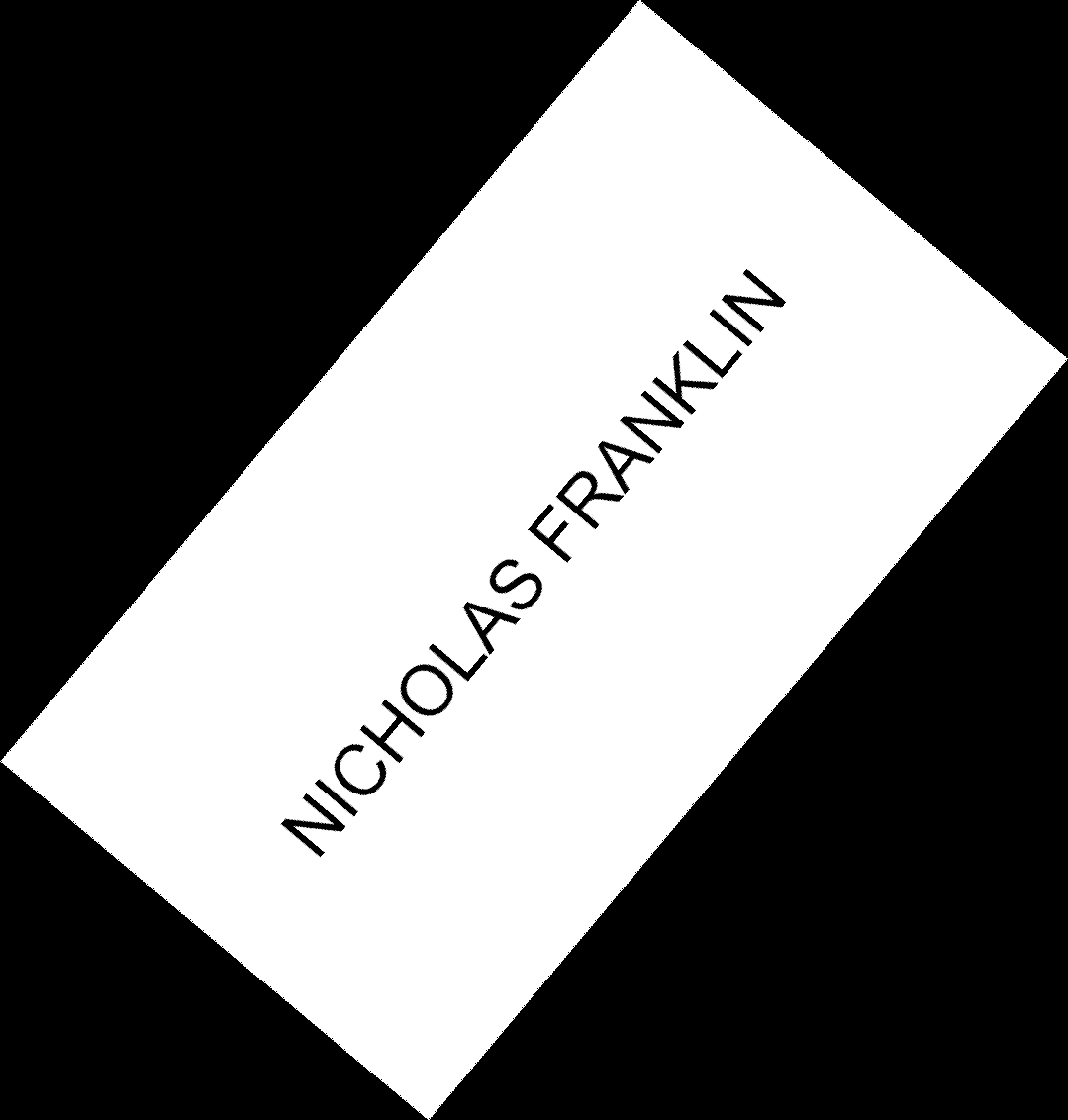
**Student number 150402149**

**Question 1(a):**

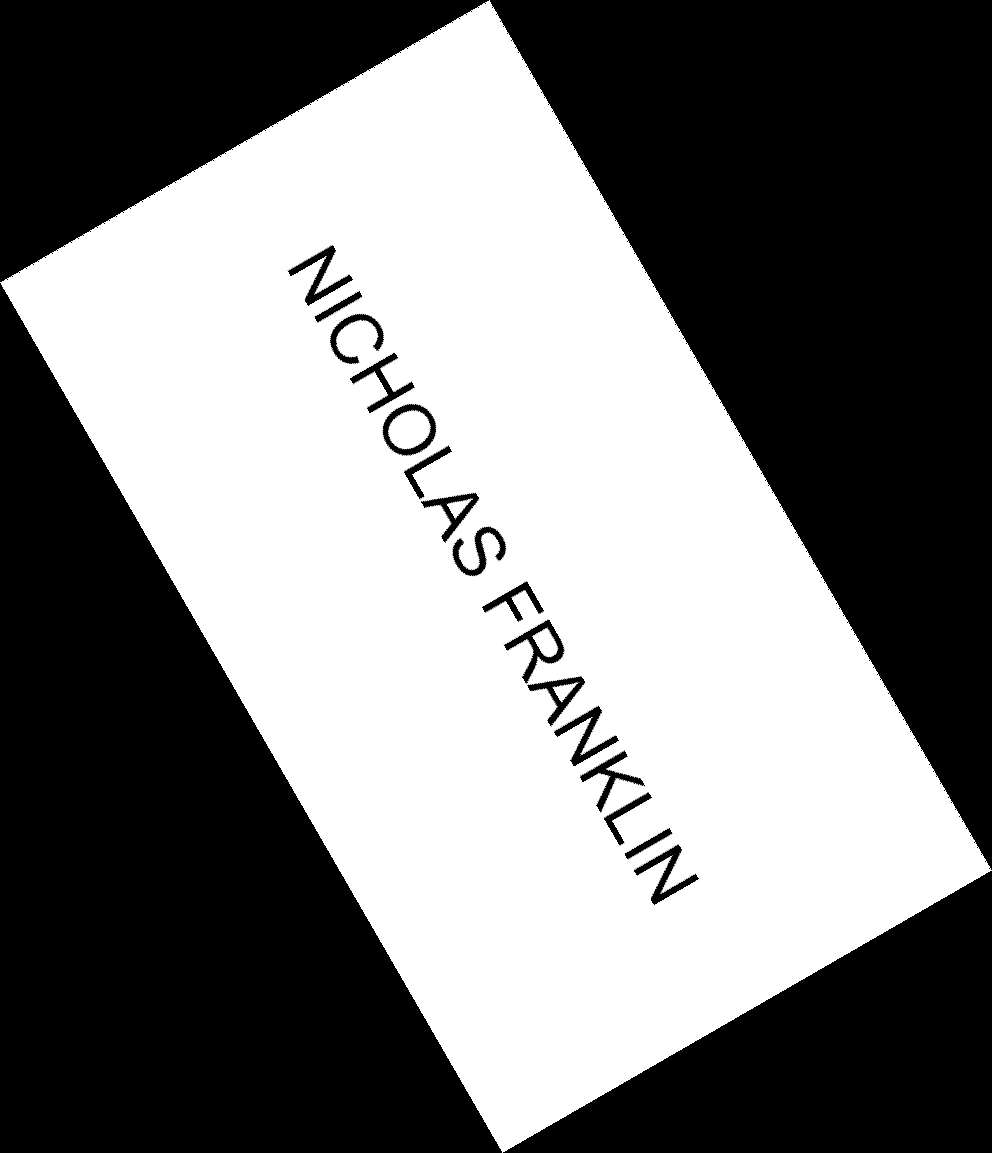


**Rotated images:**

θ = -50 deg



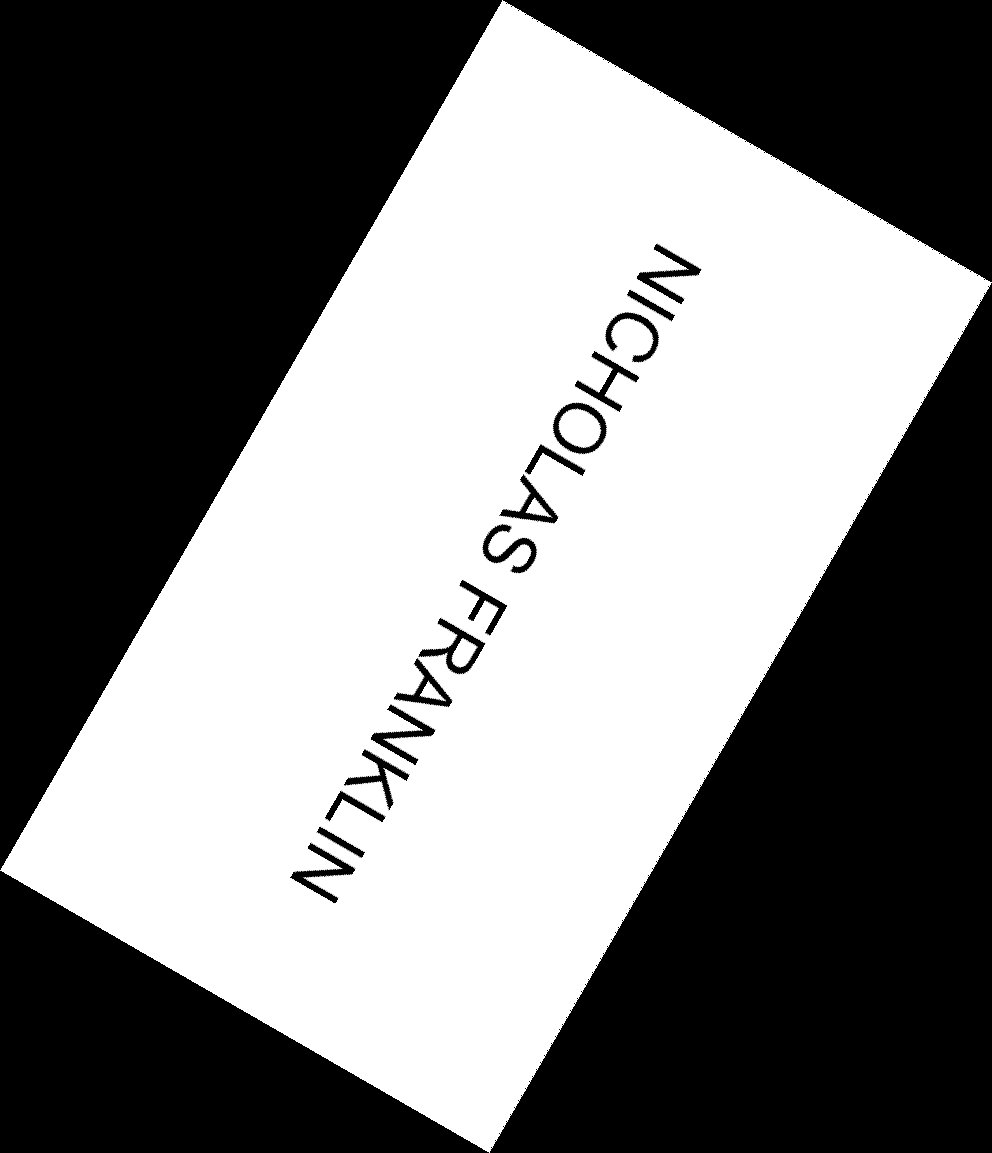
θ = 60 deg



θ = 30 deg



θ = 120 deg



**Skewed images:**



θ = 10 deg



θ = 40 deg



θ = 60 deg



**Your comments:**

I was skewing with the top left pixel at 1,1. Resulting in errors at small scale. I also decided that I wanted to skew from the bottom left. So, the angle skew by is added to -90 degrees to make skewing form the top left equivalent to skewing from the bottom left with the angle being that between the vertical and the transformed line.

In addition, the skew and the rotation matrix I am using are the reflections of the standard versions of these matrixes because later in development I found that I had programming mistake were width and height were inverted. I mistakenly thought that this need to invert the standards matrixes was because MATLAB indexed from the top left.

I also came across problems with rotation as some of the transformed values were negative. I solved this by sizing the image based on the difference in location of the max and min points in the image. Then shifting the image into positive space to display the image.

I had no problems with rotation until I got to gap filling. The main problem I had was differentiating between a hole and a piece of the extension to the image, so the rotation is not cut off. I made an array of -1’s with dimension large enough to fit the whole output image. Then I output the new pixels into their required positions after averaging any overlayed pixels. Next, I loop thought the output image and reverse transformed the target -1 pixel if its new location was outside off the dimensions of the original image it is assign a value of 0. If it is in the bounds of the image, then it is assigned a colour based on it 1 NN.

I learnt how cell arrays worked so that I could find overlay pixels. I filled a cell array with the transformed pixels locations. When an overlay was encountered the depth of the cell array was increased and all the values were stored. After this any cell with a depth of greater than 1 are averaged together.



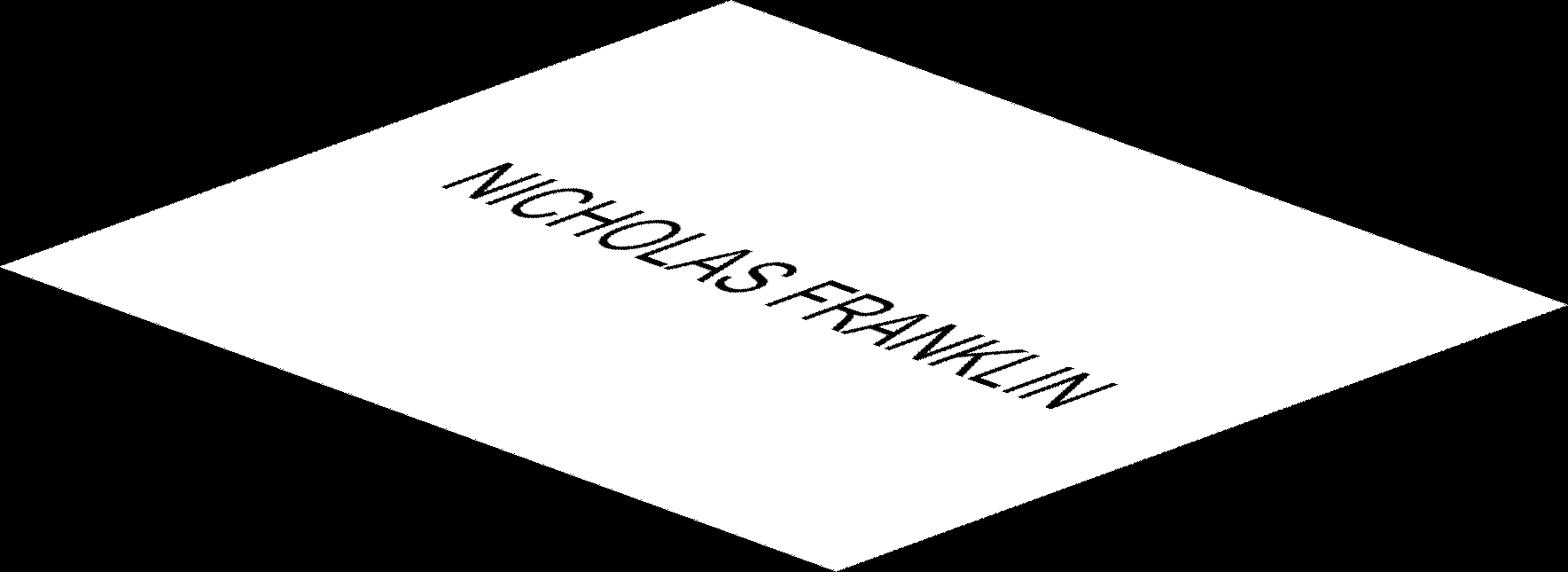
The final error I had was forgetting that I need to convert an image to uint8 before writing wasting lots of time trying to work out why alternative images didn’t work in my transformation.

I also had problems with large areas of black in the output image that were added to the parts of the original image didn’t get cropped. I solved this with the ICV\_trim method. In this method I look at the max value in any given column or row if the max value was 0 then there was only black in that row so I trimmed it.

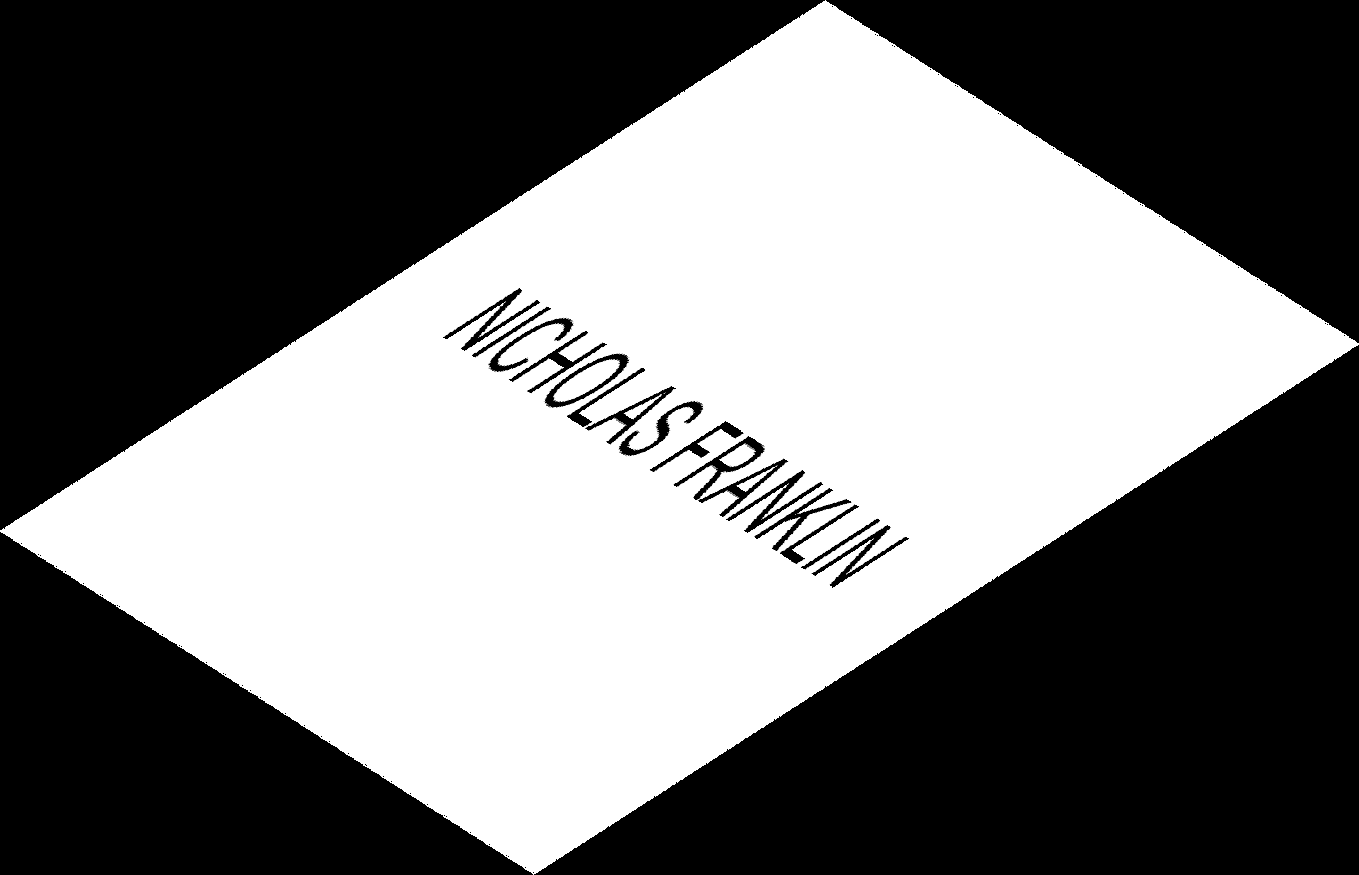
**Question 1(b):**



θ2=50 and θ1=20 clockwise



θ1=20 clockwise and θ2=50





**Your comments:**

The results are not the same if you skew then rotate vs rotate then skew.

This is because transformations are the application of a transformation matrix on a matrix that is the pixel position in the image. Therefore, applying multiple transformations to the same pixel is equivalent to multiplying one transformation by another then applying it to the points in the image and because in matrix multiplication:

they are not equal. This is because matrix multiplication is not commutative.

**Question 2(a)**:

**Designed kernel:**

**A box blur.**

Averaged image



Original image

**Your comments:**

This box blur kernel averages the value of the target pixel by the pixels surrounding it. The matrix constant (1/9) reduces the floating-point maths to a multiplication with the output of the mask on a target pixel. Therefore improving efficiency as less floating point maths is performed.

The pixel is added together by this mask and divided by 9 (the weight of the mask) so that the pixels are in the range 0-255.

I chose to convert the image to grey scale because of the comment in the instruction stating that the input array is normally a grey level image.

**Question 2(b):**

**Filtered image with kernel A**

**Filtered image with kernel B**

**Your comments:**

Kernel A is a gaussian blur approximation. It has blurred the high frequency noise from the image while reasonable painting the lower frequencies.

This can be seen in the number plate and the car manufacturer logon on the boot. The letters on the number plate are reasonably crisp because they are a large area of the same colour and gaussian blur removes high frequency noise. Whereas the logo is made up of a lot of pixel of extremely different values to there neighbours. Therefore, is similar to high frequency noise and this has been blur to an approximate single colour.

Kernel B is an edge detection kernel. It has made any vertical or horizontal edge in the image white while the rest is black.

Its design means that it responds to edges on the horizontal and the vertical. The total average output of the kernel is 0 hence most of the image being black as the kernel outputs 0 when all the pixels in the masked area are the same value. Where is if there is an edge there is a change in intensity which this filter response to creating bright areas.

The response on the vertical and horizontal is best seen on the 5 on the number place the top horizontal line is bright white as is the vertical down stroke.



The lack of response on the diagonals is seen on the bumper as is appears in the transformation image to be made up of several discounted white horizontal lines as the edge is slightly angled so only parts of the line are responding to the filter.

**Question 2(c):**

A followed by A

**A followed by B**

**B followed by A:**

**Your comments:**

i) Kernel A then kernel A further blurs the image compared to a single application of the blur. The glare on the top of the car shows this best as you see the circle of the glare growing compared to a single application of the blur and a further loss of detail in place like the car name which is no longer readable as ‘Quick’.

ii) Kernel A then kernel B. Blurs the original image then applies the edge detection filter. This results in the response on the edges being reduced. This can be seen by less intense lines on the edges. However, the edges detected are more continues comparted to just applying B.

iii) Kernel B then kernel A. Applies an edge detection then gaussian blurring to the image. This results in stronger responding points in the image compared to A then B (they are further to white). However, the lines in this image are often not continues. This is because the edge detection responses best to large changes in the intensity between the target pixel and the neighbourhood which are more common before the blur. In addition, the responses in B then A are thicker because the blur averages some of the adjacent pixel from black closer to the value of the detected edge.

**Question 2(d):**

**Extended kernels of A and B (5x5):**

**Results obtained by applying 5x5 kernel:**

**B followed by A**

**A followed by B**

**A followed by A**

**Extended kernels of A and B (7x7):**

**Results obtained by applying 7x7 kernel:**

**A followed by A**

**B followed by A**

**A followed by B**

**Your comments:**

d) **Discussion on how to extend them to larger filter kernels.**

I identified kernel A as a gaussian blur so my 5x5 kernel is what I remember a 5x5 gaussian blur to be. In additional a gaussian blur is approximately a bell curve so I model my 7x7 kernel A on a bell curve that can be seen to grow slowly near the edge growing faster as you get to the centre with a peak to try and mimic the properties of the 5x5 and 3x3 blurs.

I created a 5x5 and 7x7 of kernel B by identifying that the average output of the kernel would be 0 so I decided to maintain that. In addition, we wanted the effect to be the same therefore we wanted it to continue to respond to horizontal and vertical lines only. So, when I extended the kernel, I increased the value the centre pixel when I added additional 1s on the horizontal and vertical axis of the mask. Therefore, in all my kernel B extended masks, they result in a 0 if applied to an area of the same colour.

**Comment on the results.**

Kernel A 5x5 then kernel A 5x5 further blurs the image compared to the result in C. The car name is now nothing more than 3 lighter squares.

Kernel A 5x5 followed by kernel B 5x5 show the greatest improvement in clarity the stronger response on the lines results in some of the lines being white another grey giving greater clarity on the relative size of the different horizontal and vertical lines in the original image.

Kernel B 5x5 then kernel A 5x5 is a reduction in quality of the image compared the 3x3 filtered version from C the lines are getting muggy and the changing response strength within the lines on the bumper is getting more visible. It does a slightly better job at showing of the response of very thin lines compared to A followed by B 5x5 as they are blurred to cover a larger area. However, the lack of an initial blur has also resulted noise in the image being detected as edges this can be show by the grey blurring in large areas that used to be near black in the 3x3 version.

Kernel A by A 7x7 continues to do the same as the pervious versions but more so. Now the blur is so strong that the large letters on the number plat are no longer readable. In addition, most details have now been lost with little more than the image being a car being discernible.

Kernel A by B 7x7 further improves on the 5x5 for edge detection we can now clearly see the outline of the dashboard. Which would further help if we wished to segment the image including a segment for interior features of the card.

Kernel B by A 7x7 is consists of a lot of noise with some edges now merged into one another and not separable. This is best shown on the circular logo which in the A B 7x7 is clearly two circle and it also reasonably clearly two circle int eh B A 5x5. I don’t think this a useful image as the noise is now become an increasing factor in the output image.

**Question 3(a):**

**Two non-consecutive frames:**

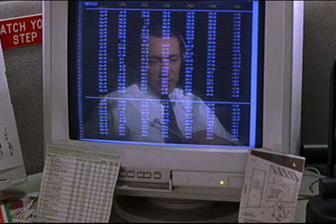
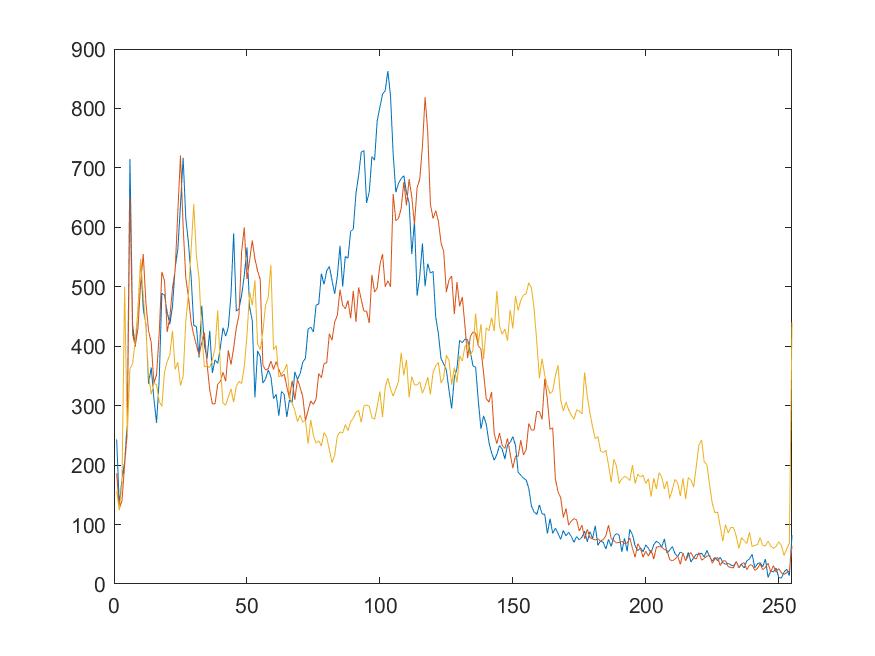
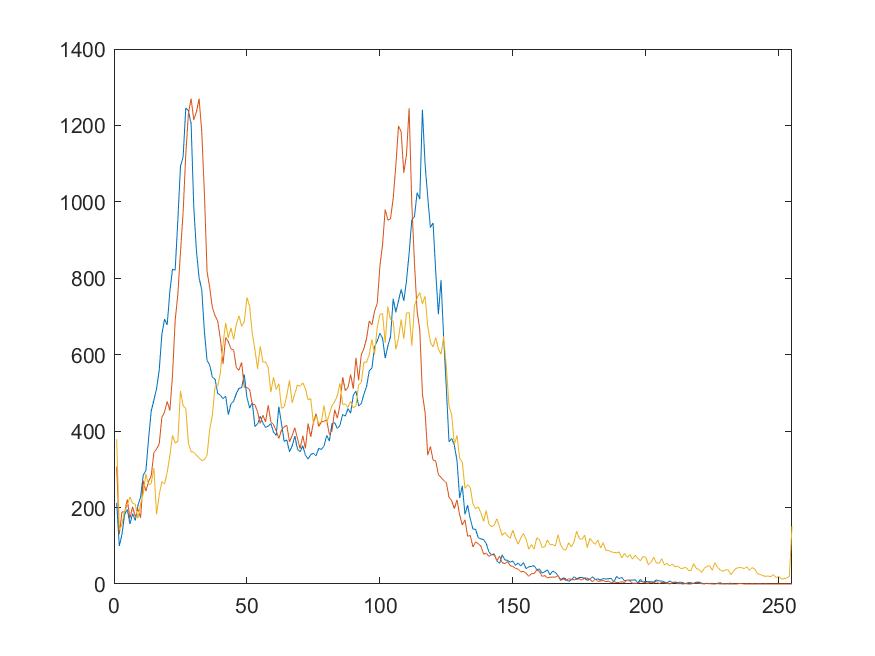
Image 1

Image 2

**Corresponding colour histograms:**

Histogram 2

Histogram 1

**Your comments:**

Due to MATLAB indexing from 1 I had to add one to every colour to find its location in an array tracking the counter. With the index of the array being the colour value plus 1.

Therefore, I found I had to convert from double to uint16 or I would loss the final value as it would overflow uint8 resulting in a peak at the end of the histogram

I created histograms for every frame in the video and store them in the cell array ‘histo’ indexed by their frame number therefore can visualise any frames histogram.

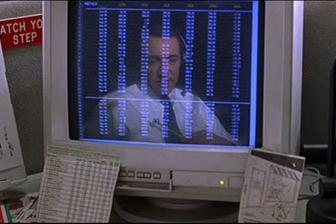
**Question 3(b):**

**Example 1:**

It Frame 88

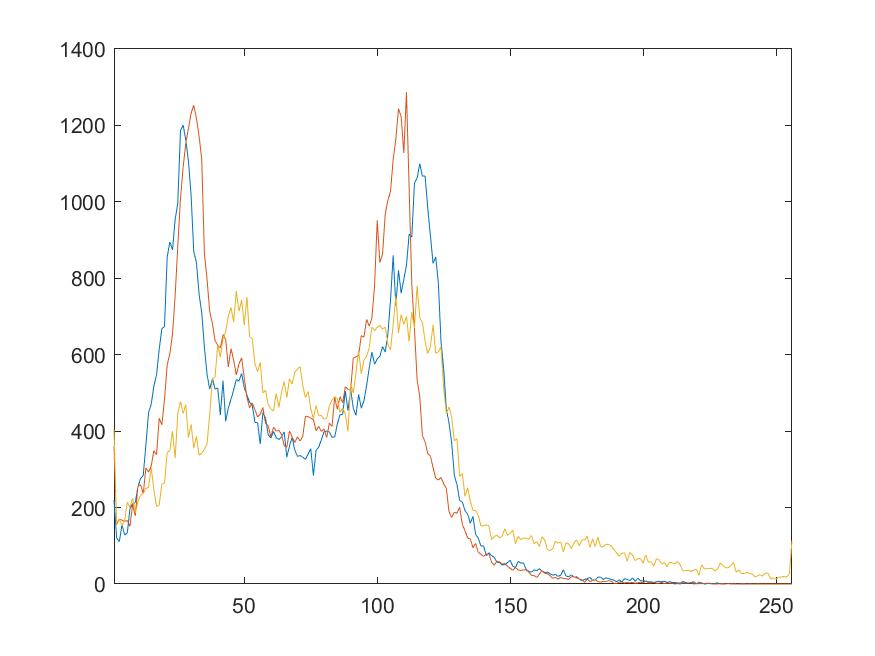


It+1 Frame 89

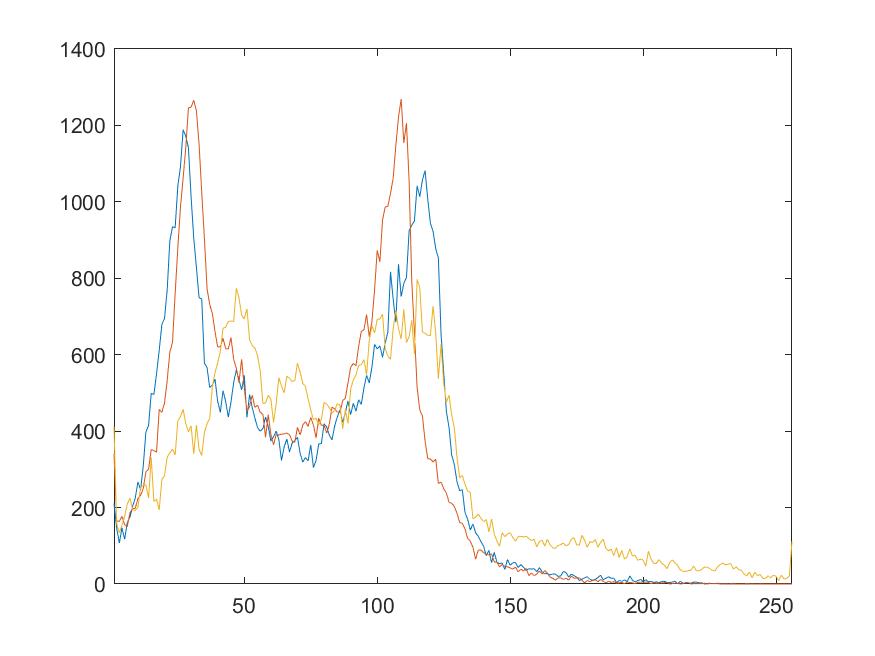


**Histograms:**

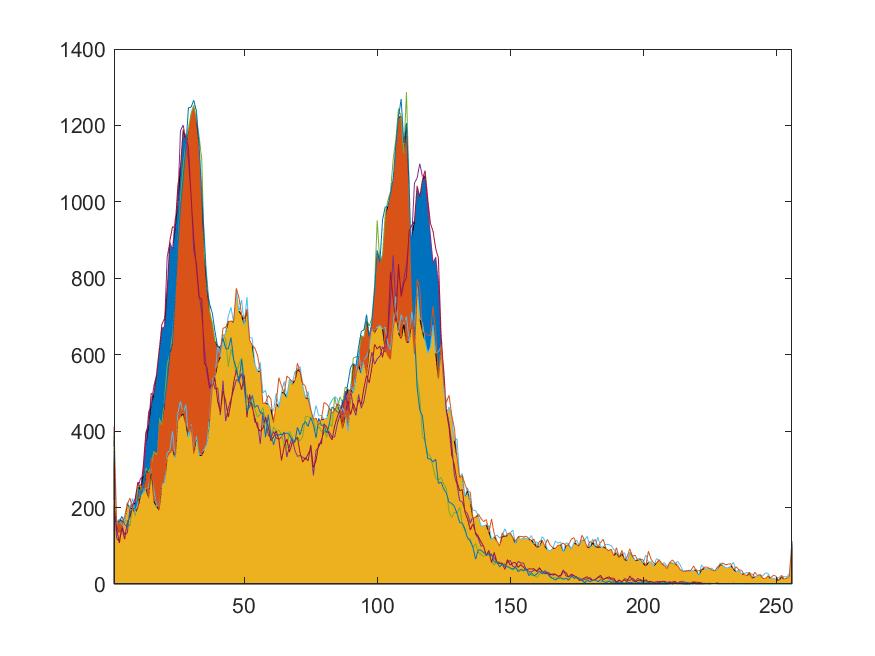
Histogram of It



Histogram of It+1



Intersection result



% Intersection = 97.45%

**Example 2:**

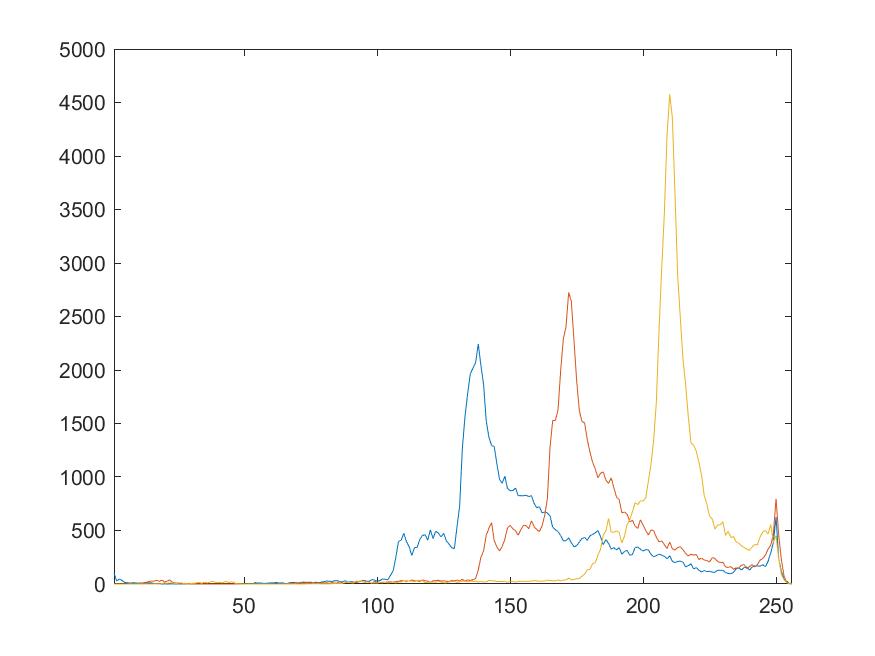
It+1 Frame 10

It Frame 9

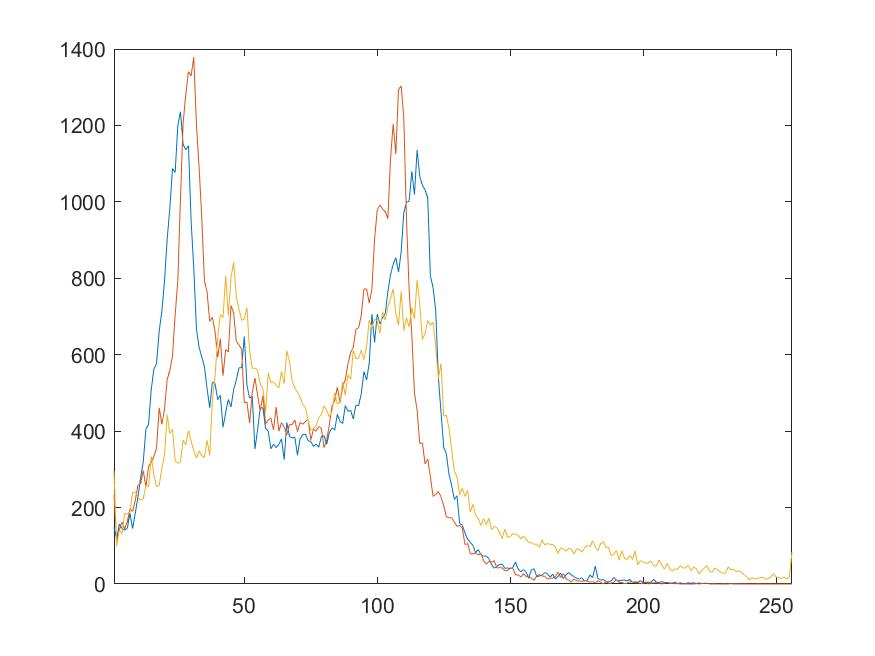


**Histograms:**

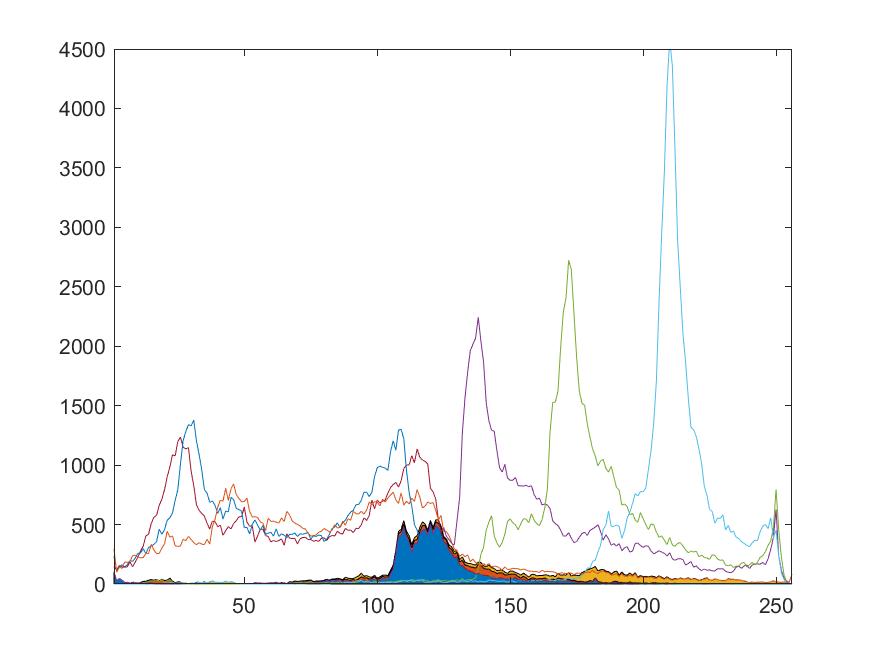
Histogram of It



Histogram of It+1



Intersection result



% Intersection = 10.04 %

**Your Comments:**

I normalised the intersection value by taking the number of pixels that exist in the

Visualizing the histograms was hard as most frames were very similar to the previous frame so weren’t very clear. Frame 9 and 10 gave clear results as they have minimum intersection, so the shaded intersection area is very clear. For frames 88 and 89 the intersection is extremely small and only seen on close inspection as areas of white under parts of the curve.

**Does normalizing the histogram change the results?**

Normalizing the results into a single percentage intersection value changed the results greatly. It was extremely hard to tell in the 88 and 99 intersection how much they intersected by with the visualisation I would have assume near 1%.

In however didn’t make it clearer how little 9 and 10 intersected as the intersection was clearly low from my visualisation. However, it does allow a much easier comparison between the comparative intersection between different frame intersections.



**Question 3(c):**

**Comments:**

**What does the intersection value represent for a given video input?**

The intersection value represents the number of pixels values in an input frame that also exist the in model frame. The total intersection value divided by the number of pixels in the model frame is the percentage of pixels that occur in both images a useful value to quickly compare scene difference.

**Can you make decisions about where the scene in the video changes?**

You can make decisions about scene change in a video with histogram intersection because scene change typically come with a location change. Therefore, a large change in the colour in the image. This means that if a histogram changes beyond a threshold then we can identify that as a scene change. In media scene are not always static but if they move there won’t be a sudden change in the histogram it will change over time. Therefore, can be identified as probably a moving in a scene not a scene change. The percentage intersection value is probably easier to use as a comparison but less table the comparison particular parts of the histogram intersection.

**How robust is the histogram intersection technique to changes?**

This technique is not very robust. There is no consideration to anything beyond the total pixel count of the scene so some scene changes could maintain the same histograms resulting in no identified scene change. This method removes a dimension of information to get an easier to compare values. The location of pixels is useful information that is not used to a change in scene that maintains the same colours frequencies would go unidentified even if the locations changed completely.

**Where does it fail?**

Histogram intersection fail on scenes that have large colour changes within them while continuing to be the same scene and scene changes that maintain the same colour frequencies. The best example of this would be a nightclub with flashing coloured lights. Each change in the lighting would likely be detected as a scene change as the histogram of that image would be dramatically different to the pervious image in the video sequence. Therefore, it would produce false positives.

In addition, this method fails when adjacent frames in a scene change have very similar histograms so produces a false negatives as the scene change is not detected.

**What would be other applications areas where histogram calculations and histogram intersection can be used?**

Histogram calculation can be used to segment image. If you can identify two peaks in an image, then a point selected between them is a good point to segment an image at to get a binary image with too distinct separate areas.

Histogram calculation can also be used to equalise an image so that the image uses the whole range of colours within an image. You can create a look up table using the histogram of an image which though point processing will level the histogram to that all the colour space is effectively used increasing the dynamic range in the image.

Histogram intersection can be used to create an extreme naive object classifier. If you have the histogram for some known objects you can compare the intersection of an unknown object histogram to the histogram of the known objects. Then the histogram intersection with the greater matching pixels would be identified as that known object.