Queen Mary University of London

Introduction to Computer Vision

Coursework

Submission 1

Your name **Houssem El Fekih**

Student number **171031393**

Question 1(a):

HOUSSEM EL FEKIH

Rotated images:

30 deg)

60 deg)

120 deg)

Comments:

There were many unexpected challenges with performing the rotation:

First I had to construct the correct method for applying the matrix and using backwards mapping instead of forward mapping to avoid blank patches in output image as explained in lecture.

After applying the transformation correctly we noticed that a part of the image was outside the bounds of the output frame, this required us to correct the transformation by first aligning the image center with the origin in original format (with original height and width) and then re-center the image using new size (new_height and new_width after transformation).
The most challenging step was to actually calculate the correct transformed height and width in the output image as this required a case disjunction which part of the unit circle. (negative or > 90 deg etc.)
Although there were no difficulties combining R and S into one matrix It was not the preferred solution to combine the re-centering transformations (back_to_base and new_center_to_origin) in one transformation matrix due to a slowness limitations in matlab.
This is explained in the code and the alternative and slower solution (~1-2 mins vs 10 sec), is provided since it is more mathematically elegant, and in theory should have been faster.
It can be called with ICV_Rotate_alt(path, angle_rot_rad, skew_angle). My best understanding is that this is quite a strange artefact of matlab inv and / operations, one can test this strange behaviour by trying to map a reverse translation to a point using Point/M (had to use inv(M)*P instead), with a translation matrix M.

One noteworthy observation relates to the convention between angles in mathematics (anti-clockwise) and the question asking us to rotate clockwise.

Skewed images:

10 deg) **40 deg)** **60 deg)**

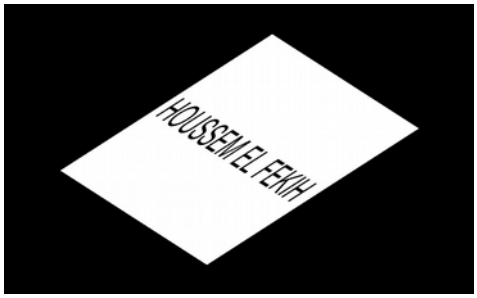
Your comments:

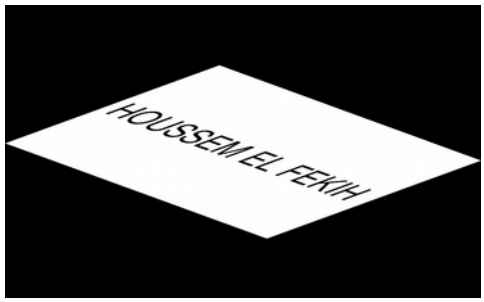
Following the convention for rotation It was presumed that skewing was clockwise (positive angle skews the top to right).

This horizontal skewing the matrix encoding x' = x - tan(theta) * y. this has the effect of increasing the width to w + tan(theta)* h. when we are testing our function ICV_Rotate it we add a new parameter for horizontal skew angle and set the rotation angle to 0.

As it can be checked in our code, this means that the process for the transformation is exactly the same we just multiply our Skew matrix S with R (or premultiply for other order around) to obtain the correct transformation.

Question 1(b):





Your comments:

It is not the same, these operations are not commutative.
Here's the mathematical proof:

Mapping of a point with order 1 R o S (skew then rotate)
after skew:
x' = (x - tan(theta))
y' = y

after rotate
x'' = cos(theta)*x' + sin(theta)*y' = cos(theta) * (x – tan(theta)) + sin(theta) * y = cos(theta)x + sin(theta)*y – tan(theta)* cos(theta)
y'' = -sin(theta)*x' + cos(theta)*y' = -sin(theta)*x' + cos(theta)*y'

In the other order S o R (rotate then skew)
after rotate:
x' = cos(theta)*x + sin(theta)*y
y' = -sin(theta)*x + cos(theta)*y

after skew:
x'' = x' - tan(theta) = cos(theta)*x + sin(theta)*y - tan(theta)
x'' = (-sin(theta)*x) + cos(theta)*y

x R o S – x S o R = -tan(theta)* cos(theta)) – tan(theta) = -tan(theta) * (1 + cos(theta)) != 0
y RoS – y SoR = 0

This shows that the same point has the same y component but the x component is not the same and is off. If the same point maps to two different coordinates then it is not the same transformation.

Question 2(a):

Designed kernel:

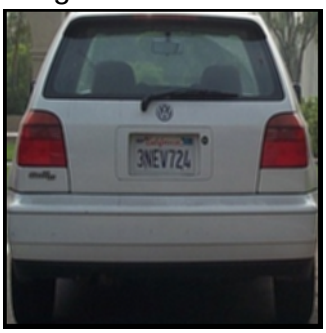
Median kernel (in matlab notation):

kernel = 1/9*[1,1,1;1,1,1;1,1,1];

As you can see this is a normalised square matrix of 1's

Original:

average filter:



Your comments:

I apply convolution on the derived mean kernel, one caveat is to carefully adjust the offsets since the kernels are not symmetric functions (in mathematical definition of convolution) but are indexed as a 2D matrix,
This means the the point at r-1, c -1 will map to first element in kernel which is indexed from 1 to 3 and not -1 to 1 in the range of kernel, this results in a slight deviation from the formulation in lecture slides where one needs to add +2 to the index. This is for a kernel of size 3X3. This is what's computed with the h_kernel_offset and v_kernel_offset in code.

This convolution operation is applied on each channel of the image, resulting in the above image.

As for handling corner, the filtering approach skips corner points since it is a possible strategy, leaving behind a blank border.

The result is a barely noticeable blur and effective removal of grainy noise, there are more blocky areas around curves such as rear lights or rear windshield.

Question 2(b):

Your comments:

Filter A has standard effect of a Gaussian kernel, this Gaussian has a standard deviation of approximately 1 which results in a slight smoothing effect, as you can see this has better defined regions where there was blockiness before, there is still blocking or aliasing but it's a much better than before.

As for Filter B it is an laplacian operator approximating a second derivative with a central difference at every point, This means that the absolute value will be high when there is a high variation in pixels, which is around edges. In our picture we can distinguish the borders where the lines go dark (at zero crossings).
However since the picture is noisy we can see that aside from the border the noise is present in the feature map.

Note that for both filters Normalisation was needed in order to not overflow any pixel. We also had to cast our values to a double image to allow negative values for the laplacian kernel. We then needed to normalise the output values since some were in negative range.

This is done by first computing the minimum c and maximum d and applying the formula
Pout = (Pin - c) x (256) / (d-c)

Question 2(c):

A followed by A:

A followed by B:

B followed by A:

Your comments:

filter A followed by A achieves a further blurring effect and big noise reduction, this illustrates the fact that the convolution of two Gaussians is a Gaussian, since convolution is associative.

Filter A followed by B obtains good results to extract the borders of the car, since the previously seen noise artefacts are no longer misclassified as edges. This is what's commonly called a Laplacian of Gaussian filter.

Filter B followed by A corresponds to a smoothing of the noise that the Laplacian was sensitive to is smoothed this achieves a similar result to the Laplacian of Gaussian with a feature map that is clearer in tone. This eliminates some of the less pronounced borders , but this loss of information is barely noticeable and might be desirable in some applications.

Question 2(d):

Extended kernels of A and B (5x5):

I have identified in (2) that the 3x3 kernels A and B are respectively a gaussian (bilinear) filter and a Laplacian of Gaussian filter. The natural extension for these filter would be a gaussian filter and a LoG of 5x5 and 7x7.

In order to generate a discrete Gaussian or Laplacian of gaussian kernel one could define bounds and sample the continuous functions at middle pixel of defined bounds coarsely but since the function is non-linear this would introduce an unacceptable level of error, of which the effect will be magnified for a larger kernel instead we resort to an approximation by integration equally spaced regions of the continuous function at predetermined bounds. The functions involved are:

Gaussian = 1 /(sqrt(2 * pi)* stddev) * exp (- (x^2 + y^2) / 2 * stddev^2))
Laplacian of Gaussian= -1/(pi * stddev^4) .* (1 - ((x .^2 + y .^ 2)/ (2 * stddev^2))) .* exp(-(x.^2 + y.^2)/ (2* stddev^2));

But first we need to decide what variance our Gaussian has, the provided example (normalised) has a variance of 1 , the bounding box is calibrated in the [-2.5 ,2.5] range because the values are vanishingly small beyond that range which skew the values for our filter. Check out the code in (code)

This yields the following results:

G5 =1/264*[1,4,6,4,1;
4,16,25,16,4;
6,25,40,25,6;
4,16,25,16,4;
1,4,6,4,1];

l5 =[0.1684,0.6205,0.8425,0.6205,0.1684;
0.6205,0.5278,1.3114,0.5278,0.6205;
0.8425,1.3114,-6.8730,1.3114,0.8425;
0.6205,0.5278,1.3114,0.5278,0.6205;
0.1684,0.6205,0.8425,0.6205,0.1684];

The relationship between values corresponds intuitively to the underlying function but the ratio between elements in our kernel (which is what matters since the filter is re-normalised) is on the right trend broadly but a different than the standard 5x5 Gaussian or LoG filters. There are manifold reasons for this deviation. These are that the choice of our bounding box is relatively arbitrary and will lead to imprecision, the other is that we have opted for a simple box numerical integration approximation which is not the state of the art in terms of precision (simpson's or trapezoid method is more precise for example).

Results obtained by applying 5x5 kernel:



Extended kernels of A and B (7x7):

The exact same discrete kernel implementation process is applied for 7x7 , see code in ICV_compute_filters for details:
Although our gaussian is still relatively well behaved with our approximation at this size, we suspect that our LoG approximation is not representative of a LoG filter, we attribute this imprecision to the fact that our bounding box needs to be a function of the size of the kernel and we will need to experiment with it to obtain the right results. Unfortunately there is no time to do that experimentation the result is that the 7x7 LoG is incorrect, instead to get some meaninful result we will use a commonly used approximation, please be aware that we have use the output of our handwritten ICV_compute_filter in all other instances...

we obtain:
g7=1/1035*[1,3,7,9,7,3,1;
3,12,24,31,24,12,3;
7,24,51,65,51,25,7;
9,31,65,83,65,31,9;
7,24,51,65,51,25,7;
3,12,25,31,25,12,3;
1,3,7,9,7,3,1];

l7 (obtained through our code):
[0.0374,0.1426,0.2850,0.3471,0.2858,0.1436,0.0379;0.1426,0.4097,0.4809,0.3606,0.4801,0.4115,0.1441;0.2850,0.4809,-0.5676,-1.7102,-0.5793,0.4792,0.2875;0.3471,0.3606,-1.7102,-3.7035,-1.7313,0.3551,0.3498;0.2858,0.4801,-0.5793,-1.7313,-0.5912,0.4783,0.2883;0.1436,0.4115,0.4792,0.3551,0.4783,0.4132,0.1451;0.0379,0.1441,0.2875,0.3498,0.2883,0.1451,0.0383];

real 7x7 Laplacian (from http://homepages.inf.ed.ac.uk/rbf/HIPR2/log.htm)
l7 = [0,1,1,2,2,2,1,1,0;
1,2,4,5,5,5,4,2,1;
1,4,5,3,0,3,5,4,1;
2,5,3,-12,-24,-12,3,5,2;
2,5,0,-24,-40,-24,0,5,2;
2,5,3,-12,-24,-12,3,5,2; **(continued next page)**

1,4,5,3,0,3,5,4,1;
1,2,4,5,5,5,4,2,1;
0,1,1,2,2,2,1,1,0];

Results obtained by applying 7x7 kernel:



Your comments:

Overall the same effects were observed in 5x5 than in 3x3 for the Gaussian and LoG filters. There is a noticeable improvement in the detection of borders in LoG filter at 5x5 since we can see borders inside the rear lights (and this is a border that is tending towards green as well as gray) because this 'border' is the result of the variation in tone of red in rear light which is unusual and not present in the rest of the image.

The 7x7 LoG (from previously mentioned source) is not responding well to this image.. not sure why..

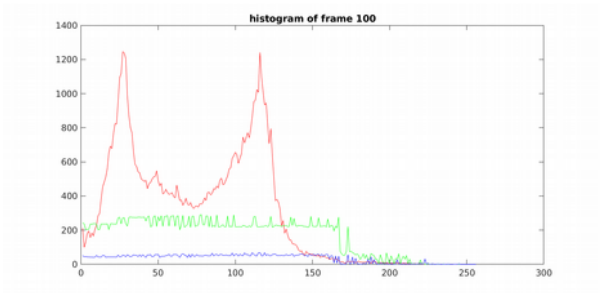
Question 3(a):

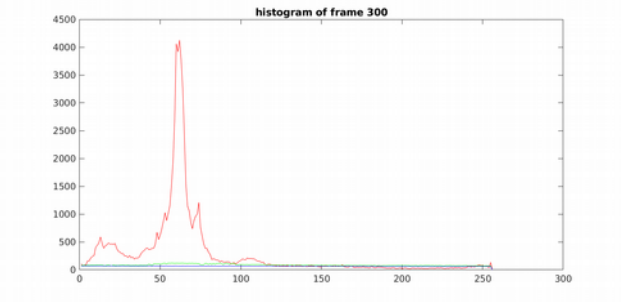
Two non-consecutive frames:

f100:

f300:

Corresponding colour histograms:





Your comments:

The histogram for frame 100 has 2 main peaks of colour in red corresponding to the red values of some dominant colour (the high one probably the whiteness in screen and background) and the other one maybe the red in the blue component of the screen since it is more purple than a pure blue.

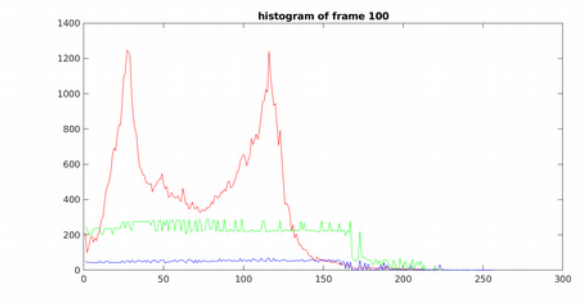
The green values are also relatively elevated compared to blue.

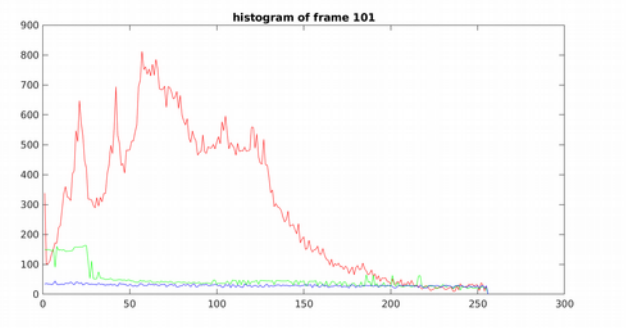
The other histogram of frame 300 is completely different and has one peak in red. The only thing in common between them is that the red value is generally much higher than other values. Which may be because of the lighting conditions which are similar since they are a part of a the same scene.

Question 3(b):

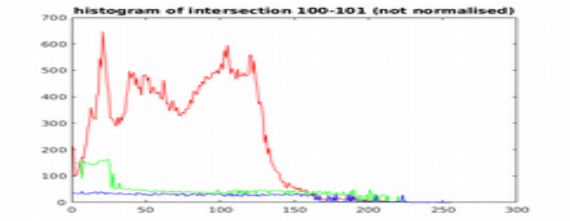
Example 1:

Histograms:

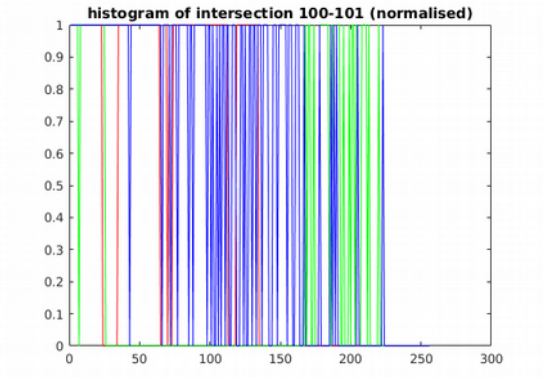




Intersection (Not Normalised):



Intersection (Normalised):

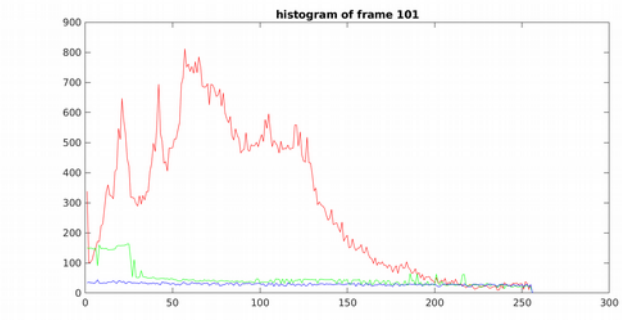


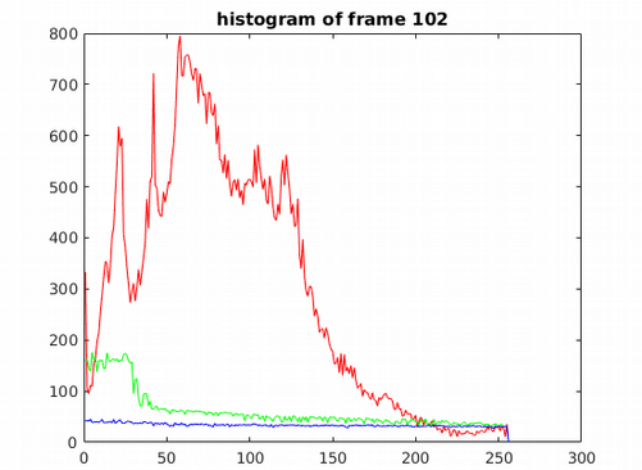
Example 2:

f101:

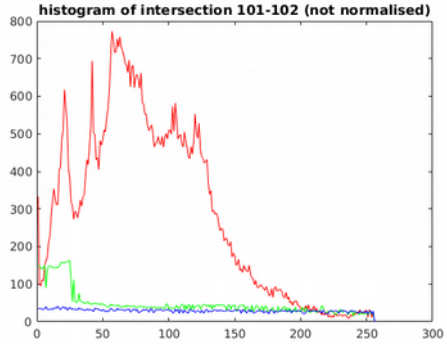
f102:

Histograms:



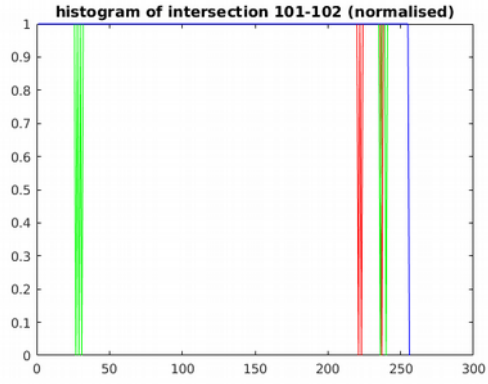


Intersection (Not Normalised):



Intersection (Not normalised):

Intersection (Normalised):



Your Comments:

In order for the intersection to be a meaningful measure of similarity between the frames I have chosen to normalise the values to [0,1] where 1 indicates the exact same colour frequencies for a particular bin over the total number of colours in that bin.
This means that after computing Min(Ij,Mj) it is divided by Max(Ij,Mj) this is more indicative of how similar that colour is than some absolute value. There are more sophisticated methods of comparing histograms, this is the simplest one I can think of.

Please do not be confused about the nature of the normalised graphs in part 2, while there seems to be a lot of values at 0, they are actually at the maximum 255, because of my inexperience with matlab plotting the blue line at the top is dominant. In fact the intersection ratio devised gave a near 1 to every value except in the small areas where the images differed obviously.

There is still less of a match between 100 and 101 since it corresponds to a change in scene, however the similarity measure is still relatively high, this shows how bar histograms are not sensitive to position and may represent completely different images but have similar colours since these images are part of a similar scene with similar lighting and types of objects present..

Question 3(c):

Comments:

This histogram similarity technique can be used for finding dominant colour in a picture, or as an unsophisticated detector of presence of particular object in general.

Histogram matching has some benefits in that it will find similarity in pictures that have been transformed affine transformations and projections. This invariance to position of pixels in the image is a loss of information and hence this method can mis-categorise two completely different images as being similar if not exactly the same.

As we have seen throughout the scene lighting conditions are a big influence over the color values, and if we take the representation in the colour space that we have our similarity measure will degrade in performance for recognising that an object is in the scene for example if there is a sudden change in luminescence. Noise is also a factor, as it can have an effect on the colour histogram.