# 60006 Computer Vision (Term 2) 2021 Past Paper Solution

*Note: this is not an official solution but I'm trying to make it as accurate as possible. If you find any typos or incorrect answers and explainations, please email me at xz1919@ic.ac.uk with the title <course_number>-<course_name>-<year>-past-paper-enquiries or directly message me through Whatsapp. If you need explaination or reviews on the provided answer, please contact me as well. :) A copy of the exam feedback is provided at the end of the document as well*

---

1. a. Given a 5x5 image as shown below, perform Sobel filtering and calculate a 3x3 output image without considering the boundaries.

    i) Write down the 3x3 horizontal Sobel filter $h_x$ and vertical Sobel filter $h_y$

    $$h_x = \begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix} h_y = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$

    ii) Perform convolution between the image and the filters. Briefly explain the procedure and then write down the output.

    According to the definition of convolution, we need to flip the Sobel filters both horizontally and vertically. The question states we do not consider the boundaries, so we only focused on the 3x3 pixels in the middle

    The final output is shown below

    $$R_x = \begin{bmatrix} 8 & 6 & 0 \\ 6 & 0 & -6 \\ 0 & -6 & -8 \end{bmatrix} R_y = \begin{bmatrix} 8 & 6 & 0 \\ 6 & 0 & -6 \\ 0 & -6 & -8 \end{bmatrix}$$

    Surprisingly, the result of convolving with $h_x$ and $h_y$ are the same :)

    b. Gaussian filtering is commonly used in computer vision. The 2D Gaussian filter kernel is described by the following equation

    i) Gaussian filtering is combined with the Harris detector for interest point detection. Describe the motivation for introducing Gaussian filtering here.

    Gaussian filter is used to smooth the image and scale the image (not to scale the actual dimension but to magnify the relative impact of different edges) by changing the value of $\sigma$ in the equation. Since the Harris detector is not invariant to scale, a corner could be detected as an edge if it's too big/wide. Hence, we need to use the Gaussian filter to scale the edges at different levels (thus search on both spatial and scale domain) and find the response at the most suitable scale.

    ii) Show that the 2D Gaussian filter is a separate filter.

    The proof is shown below, just as the same one shown on the slide (Edge Detection I slide page 40)

$$
\begin{aligned}
f[x,y] * h[x,y] &= \sum_i \sum_j f[x-i, y-j] \cdot h[i,j] \\
&= \sum_i \sum_j f[x-i, y-j] \cdot \frac{1}{2\pi\sigma^2} e^{-\frac{i^2+j^2}{2\sigma^2}} \\
&= \sum_i (\sum_j f[x-i, y-j] \cdot \frac{1}{\sqrt{2\pi}\,\sigma} e^{-\frac{j^2}{2\sigma^2}}) \cdot \frac{1}{\sqrt{2\pi}\,\sigma} e^{-\frac{i^2}{2\sigma}} \\
&= \sum_i f * h_y[x-i] \cdot \frac{1}{\sqrt{2\pi}\,\sigma} e^{-\frac{i^2}{2\sigma^2}} \\
&= (f * h_y) * h_x
\end{aligned}
$$

iii) Suppose the 2D Gaussian kernel size is N x N, the input image size is N x N and Gaussian filtering is performed. Each multiplication is defined as one operation. Each addition is also defined as one operation, the same as multiplication. If we treat boundary pixels in the same way as inner pixels, how many operations do we need for direct 2D Gaussian filtering and separable filtering respectively?

We first discuss the complexity of a direct 2D Gaussain filtering

For each pixel, we perform $K^2$ multiplications and $K^2 - 1$ additions, and if we treat the boundary pixels the same was as inner pixels, then we have $N^2(2K^2 - 1)$ as the number of operations we need to perform

We then discuss the complexity of a separable filtering of Gaussian

For both the horizontal and vertical filters, for each pixel we need to perform $K$ multiplications and $K - 1$ additions. For an N x N image the total number of operations needed for a single 1D filter is thus $N^2(2K - 1)$. For two 1D filters it's just double the amount: $2N^2(2K - 1)$

c. Median filtering is often used for image denoising. Suppose a uniform region of an image is corrupted by salt and pepper noise (salt: random white pixels; pepper: random black pixels). Show that provided less than half of the pixels in a neighbourhood are corrupted, a 3x3 median filter will almost perfectly restore the uniform region.

From the question statement, we assume the region we are dealing with is uniform. (e.g. the pixel in the region are all about the same intensity value) With this assumption, let's assume a region of 3 x 3 with the intensity values below

$$
\begin{bmatrix} x_1 & x_2 & x_3 \\ x_4 & x_5 & x_6 \\ x_7 & x_8 & x_9 \end{bmatrix} \text{ where } \mathrm{x_1} \approx \ \mathrm{x_2} \approx \ \mathrm{x_3} \approx \ \ldots \ \approx \ \mathrm{x_9} = \mathrm{n}
$$

If we randomly change four of the pixels to either 0 or 255 (less than half of the pixels in the neighbourhood), then we still have other five values staying at the same/similar pixel intensity value $n$. For calculating the median of those nine values, when we rank the numbers in ascending/descending order, we will still get median near the value $n$ because the median is defined to be the number in the middle. In the worst case, when we have all four corrupted pixels to be equal to 0 or 255 (i.e. they are either the lowest four pixels or the highest four pixels), we will still have median near $n$. See the illustration below

$$
[0, 0, 0, 0, x_5, x_6, x_7, x_8, x_9] \\
[x_1, x_2, x_3, x_4, x_5, 255, 255, 255, 255]
$$

We can observe that the midian will be $x_5$ for both sequences, where $x_5 \approx n$ from the assumption

2. a. Interest point detection

   i) Let I[x, y] denotes an 2D image, G[x, y] denotes a 2D Gaussian filter and $*$ denotes convolution. Show the property in the paper

   Using the property that the differentiation of convolution is the same as applying differentiation to one function and then convolve, we have:

$$\begin{aligned}\frac{\partial^2(I*G)}{\partial x^2}+\frac{\partial^2(I*G)}{\partial y^2} &= I*\frac{\partial^2 G}{\partial x^2}+I*\frac{\partial^2 G}{\partial y^2}\\ &= I*(\frac{\partial^2 G}{\partial x^2}+\frac{\partial^2 G}{\partial y^2})\end{aligned}$$

   ii) Explain what the difference of Gaussian (DoG) filter is and why it is used for interest point detection.

   The difference of Gaussian filter is defined below

$$DoG(x,y,\sigma)=I*G(k\sigma)-I*G(\sigma)$$

   where k is a suitable value such as $\sqrt{2}$

   It basically takes an image, smoothing it with Gaussian kernels of scale $k\sigma$ and $\sigma$ separately. Then we make a difference between them to calculate the DoG response at scale $\sigma$

   Difference of Gaussian filter is a very good approximation of Laplacian of Gaussian filter, which detects interest points as local extrama across both scale and space above a given threshold. The shape of DoG is very similar to LoG, which is an inverted Mexican hat shape

   b. Feature description: After interest points are detected, we use SIFT to describe the feature for each interest point. Suppose 4x4 subregions are used, centred at the interest point (shown on the right). Explain how the SIFT descriptor is calculated for this interest point (we already know its location and scale).

   (We don't need to explaine interest point detection and keypoint localisation because we already know the location of the interest point. If you are unsure during the exam just write "assuming we have already performed interest point detection and keypoint localisation")

   We first perform orientation assignment by calculating the gradient oeriantetion for pixels in the nieghbourhood and vote for the dominant orientation. The peak direction will be assigned as the dominant direction of this interest point.

   We then calculate the SIFT descriptor for each of the 4x4 subregions: calculate the gradient magnitude and orientations for each of the pixels in each subregion, and then create a histogram of gradient orientations separated into 8 discrete intervals/bins. The height/length of bars in the histogram represents the gradient magnitude in that direction. Notice that the gradient orientation needs to be rotated by the amount of dominant direction we have assigned to the interest point in the orientation assignment step.

   c. Image matching:

   i) After feature description, we match interest points between Images A and B (shown below). Suppose we know the two images are related by translation and isotropic scaling only. Let (x, y) denote an interest point in A and (u, v) denote an interest point in B. Write down the equation that maps (x, y) to (u, v). How many pairs of points are needed to uniquely determine the mapping?

   The equation of mapping between (x, y) and (u, v) is shown below

$$\begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} m_1 & m_2 \\ m_3 & m_4 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} t_x \\ t_y \end{bmatrix}$$

Since there are 6 unknown variables to solve and all of the 6 appear in the equation above. We need at least 6 equations, or 3 pairs of points to uniquely determine the mapping

ii) We have obtained a number of point pairs between Images A and B. However, since some of these pairs are outliers, we decide to use RANSAC to fit the transformation model. Suppose 50% of the point pairs are outliers and in RANSAC each time we sample 3 pairs. How many times do we need to sample to assure that with 95% probability, we have at least once that all 3 point pairs are inliers? You can keep the logarithm in the answer.

(Personally I think this question is a bit irrelevant -.-)

We first use the geometric distribution to calculate the probability that all 3 sample pairs are inliers for the first time. Notice that the probability that all three point pairs are inliners is $(\frac{1}{2})^3 = \frac{1}{8}$

$$p(X) = \frac{1}{8} \cdot (\frac{7}{8})^{x-1}$$

Then we take integral over this function to form the cumulative density function and then solve for t in the equation below

$$\begin{aligned}
\int_1^t p(x)dx &= 0.95 \\
\frac{1}{8} \cdot (\frac{7}{8})^{t-1} \ln \frac{7}{8} - \frac{1}{8} \cdot (\frac{7}{8})^{1-1} \ln \frac{7}{8} &= \frac{19}{20} \\
\frac{1}{8} \ln \frac{7}{8} ((\frac{7}{8})^{t-1} - 1) &= \frac{19}{20} \\
(\frac{7}{8})^{t-1} &= \frac{19}{20} \cdot 8 \cdot (\ln \frac{7}{8})^{-1} + 1 \\
(t-1) \ln \frac{7}{8} &= \frac{19}{20} \cdot 8 \cdot (\ln \frac{7}{8})^{-1} + 1 \\
t &= \frac{38}{5} \cdot (\ln \frac{7}{8})^{-2} + (\ln \frac{7}{8})^{-1} + 1
\end{aligned}$$

We then take the floor of $t$ for the result because if, say $t$ is equal to 3.5, then that means we have to throw at least 4 times to get above 95% probability of all three being inliers.

Hence, the minimum number of times we haev to sample is equal to $t = \lceil \frac{38}{5} \cdot (\ln \frac{7}{8})^{-2} + (\ln \frac{7}{8})^{-1} + 1 \rceil$

3. a. You are given a dataset that consists of the Big Ben and other landmark buildings in London (some examples shown below).

   i) Your first task is to develop a classifier to detect the Big Ben in new images. You downsample the images to reduce dimensionality and then train a classifier on these images to differentiate images that contain Big Ben versus those not. However, you find the classifier performs poorly on test images. Explain potential reasons why the classifier is not working well.

There are several reasons that the classifier might fail to work. The first two common reasons are underfitting and overfiitting. Underfitting happens when the classifier has not been trained enough/or the classifier model (e.g. neural network architecture) is too complex to differentiate Big Ben from other images, whereas overfitting means the classifier model is too simple to learn Big Ben or the classifier has been training for too long. Of course there are other reasons, such as the downsampling is not working (e.g. the image shrinks too much), or the feature selection is poor. (e.g. the actual input to the classifier should be image features rather than just image pixel intensities)

ii) Your second task is to develop an image retrieval system using the dataset of London buildings. Given a new image of some building, the system will retrieve similar images. Describe how you would design such a system.

First, calculate the interest point of each image in the dataset and use a suitable scheme to represent the local features at each interest point. Then cluster similar images according to the number of similar/same interest point as well as their relative locations in the image. The clustering can be done using either unsupervised machine learning methods such as K-means, or we can use traditional algorithms such as PageRank. When giving a new image, the system can first return the image with the most number of similar interest point, then it can return images within the same cluster.

The suitable scheme to represent local features is open-ended. We can use anything mentioned in the lecture: SIFT, SURF, BRIEF, or the HOG, or some better things that you might know if you are a genius :D

iii) Now the image retrieval system has been developed and you are to evaluate its performance. You provide it with a query image and the system responds with a ranked list of images in the order of decreasing probability. The relevances of the returned images are [+1, +1, -1, +1, +1, -1], where +1 means that the image is indeed relevant to query and -1 means not relevant. Calculate the precision and recall values needed for plotting the precision-recall curve.

Since the returned list of images is ordered by decreasing probability, we can set the threshold to be between each image, e.g. the first n image are Big Bens and the rest are not.

Since there are 6 images, there are 5 possible threshold. Below is the calculation process

| Threshold | Precision | Recall |
|---|---|---|
| 1 | $\frac{1}{1+3} = \frac{1}{4}$ | $\frac{1}{1+0} = 1$ |
| 2 | $\frac{2}{2+2} = \frac{1}{2}$ | $\frac{2}{2+0} = 1$ |
| 3 | $\frac{2}{2+2} = \frac{1}{2}$ | $\frac{2}{2+1}$ |
| 4 | $\frac{3}{3+1} = \frac{3}{4}$ | $\frac{3}{3+1} = \frac{3}{4}$ |
| 5 | $\frac{4}{4+0} = 1$ | $\frac{4}{4+1} = \frac{4}{5}$ |

b. The virtual background function is commonly used for video conferencing (e.g. Zoom or Microsoft Teams). To this end, we need to separate the foreground from background for an image and blend it with a new background according to the equation shown in the paper.

i) Suppose the user in video conferencing uses a green screen, how would you estimate the $\alpha$ image?

This entire question is in regard to the topic of image segmentation

If we know that user will use a green screen, then we can perform thresholding on the green channel of the image (assuming the image can be grouped into two clusters) or K-means clustering (in the more general case) on the grayscale version of the image. The $\alpha$ value can then be determined by whether then pixel green channel intensity is above the threshold or belong to certain cluster(s) that have high green channel value.

GMM is also a valid answer.

ii) For general cases without a green screen, how would you estimate the $\alpha$ image?

I would estimate $\alpha$ by using K-means/GMM with a more sophisticated feature rather than intensity, such as position + colour similarity or even representations like SIFT.

Another more reliable approach is (supervised learning) to train a convolutional neural network with labled images (e.g. the training set contains images whose foreground and background pixels are labeled)

4. a. The optic flow method is a technique to estimate motion between two images, for example, two time frames in a video (shown in the paper).

i) Explain the assumptions in the optic flow method and derive the optic flow constraint equation.

The three assumptions we normally make in the optic flow method are brightness constancy, meaning that a pixel has constant brightness across time; small motion, meaning that between frames motion is small; spatial coherence, meaning that pixels move like their surrounding neighbours.

Based on the brightness constancy assumption, we have

$$I(x+u, y+v, t+1) = I(x, y, t)$$

where (x, y, t) is a spatial-temporal coordinate and (u, v) are the displacement in frame t+1

Based on the small motion assumption, we can perform Taylor expansion to form

$$I(x+u, y+v, t+1) \approx I(x, y, t) + \frac{\partial I}{\partial x}u + \frac{\partial I}{\partial y}v + \frac{\partial I}{\partial t}$$

Combine those two equations we have the optical flow constraint equation

$$\frac{\partial I}{\partial x}u + \frac{\partial I}{\partial y}v + \frac{\partial I}{\partial t} = 0$$

ii) The optic flow constraint equation at a pixel location is an under-determined system. How does the Lucas-Kanade method convert it to an over-determined system to solve the flow field?

The Lucas-Kanade method introduces the spatial coherence assumption: the pixels movement/optic flow is constant within a small neighbourhood.

Thus, we can choose a small K x K window such that

$$\begin{bmatrix} I_x(p_1) & I_y(p_1) \\ I_x(p_2) & I_y(p_2) \\ I_x(p_3) & I_y(p_3) \\ \dots & \dots \\ I_x(p_N) & I_y(p_N) \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} = - \begin{bmatrix} I_t(p_1) \\ I_t(p_2) \\ I_t(p_3) \\ \dots \\ I_t(p_N) \end{bmatrix}$$

Now, the system of linear equations is over-determined.

iii) Estimation of large displacements is a challenge in optic flow methods. How does the Lucas-Kanade method address this challenge?

The Lucas-Kanade method introduces multi-scale framework, which downsamples the image to a smaller dimension such that the large motion will appear small in a smaller image. We then accumulate the optic flow estimation when calculating it in each scale.

iv) If the Lucas-Kanade method is applied to a colour video, how would you estimate the flow field?

Since we are dealing with a colour video, we have more information than just a grayscale image. A good idea is to perform Lucas-Kanade method to estimate optic flow in each of the three RGB channels, then calculate the final value of (u, v) by averaging on all three channels

b. Given a video dataset, you are developing an algorithm for car tracking on the street (figure below). However, a challenge is that some cars may be occluded by other objects in certain time frames and then appear again later. How would you design the algorithm so that it keeps tracking of the same car through occlusions? Explain your idea.

This is a rather open-ended question, but the idea should solve two things: 1️⃣ how to estimate the car position/motion flow during the occlusion and how long should we keep tracking, and 2️⃣ how do we detect the car after it re-appears

A good idea would be to store the (u, v) displacement of few previous frames, and then calculate the estimated movement during the occlusion by calculating the difference between (u, v) pairs of adjacent frames. The difference can be seen as the derivative of the optic flow, hence can effectively estimate the direction of movement of the car even if it's occluded. We should set a number $t$ which denotes how long we should track the car. (e.g. the car might disappear forever, so we should give a maximum number of frames to calculate such estimate)

Right after occlusion, we can use a single class convolutional neural network to perform object detection on each frame, until the car appears. A potential problem would be if two similar car (e.g. car of the same model) appears, the detection might be false positive.