AVD624 - Computer Vision

Theory Assignment 2 and Programming Assignment 3

Submitted by

Vatsalya Gupta
SC19B098
B.Tech. ECE VI Semester

Department of Avionics
Indian Institute of Space Science and Technology
Thiruvananthapuram - 695 547
May 2022

Question - 1

Is it more efficient to filter an image with two 1D filters as opposed to a 2D filter? Why?

Given a 2D image filter of size $M \times N$, computing the filter would require $M \times N$ independent taps, along with by $M \times N$ multiply-add operations. For large filters, this can get computationally expensive, and we get quadratic scaling with the filter spatial extent.

If a filter is separable, we can decompose such filter into a sequence of two 1D filters. Each pass filters with a 1D filter, first with M, and then the second pass with N taps, in total $M + N$ operations. We get linear and not quadratic scaling. So, typically for any filter sizes larger than $\sim 4 \times 4$ using separable filters will be faster than non-separable approach.

Question - 2

Explain how sharpening an image with filters works, in English (not in math). Intuitively, what steps are we performing?

Convolution in 2D operates on two images, with one functioning as the input image and the other, called the kernel, serving as a filter. It expresses the amount overlap of one function as it is shifted over another function, as output image is produced by sliding kernel over input image.

A sharpening filter can be broken down into two steps: It takes a smoothed (blurred) image, subtracts it from the original image to obtain the "details" of the image, and adds the "details" to the original image.

Question - 3

Say I am at a pixel (r, c), where r is the row index and c is the column index. How can you find the difference between pixels to the right of me and pixels to the left of me, i.e. between pixels (r, c-1) and (r, c+1), using a filter? What about the difference between pixels below/above me, i.e. (r+1, c) and (r-1, c)?

We can use 1D derivative filters and convolve them with the image to find out the difference between the pixels. For finding the difference between pixels at (r, c-1) and (r, c+1), we can use a 1D derivative filter along x direction, such as [-1 0 1]. For finding the difference between pixels at (r+1, c) and (r-1, c), we can use a 1D derivative filter along y direction, such as $[-1\ 0\ 1]^T$.

Question - 4

Say you have an image of a tree, and an image of a sky. You compute the dI/dx and dI/dy image gradients for all pixels, using the answer to Question 3 above. Which image would have a larger number of high-magnitude gradients? Why?

An image of a tree will have a larger number of high-magnitude gradients because of the several textures that may be present in a tree, such as the sudden transition from trunk to branches and leaves. There may be other distinct features also present, which can provide difference in intensities. Whereas, the intensity of the colour of the sky and the clouds may vary very slowly comparatively. This sudden change will lead to higher magnitude gradients in case of an image of a tree.

Question - 5

How can you find patterns in an image (e.g. let's say you're looking for a plus sign in images) in an image using a filter? Would a 3x3 filter work for any image, i.e. where the plus signs appear at different sizes and orientations? Explain why it would/wouldn't.

We can take a filter of the same properties as the required pattern and perform a sliding window operation over the entire image. The points where the response is maximum denote the location of the required pattern. But this type of filtering operation depends upon the size of the kernel used and does not scale well. So, the same pattern, if its size or orientation is changed in the image, won't be detected by the kernel which was previously used. This is because we will not get maximum filter response at the location of pattern in the image.

Question - 6

How are feature distinctiveness and feature repeatability at odds with each other? Why is each of those properties desirable (in moderation)?

Distinctiveness measures how much the individual features can be matched to a large database of objects. Whereas, repeatability measures to what extent do detected regions overlap exactly the same scene region by comparing detected features in two images of the same scene. It is based only on the feature geometry. The advantages of having both the properties together include much more efficient feature extraction and the ability to identify larger numbers of features.

Question - 7

Describe the process of extracting the locations of corners from images, starting with the use of image gradients, and ending with thresholding the R scores.

The high-level pseudocode for Harris corner detection is as follows:

- Take the grayscale of the original image.
- Apply a Gaussian filter to smooth out any noise.
- Apply Sobel operator to find the x and y gradient values for every pixel in grayscale image.

- For each pixel p in the grayscale image, consider a 3×3 window around it and compute the corner strength function. Call this its Harris value.
- Find all pixels that exceed a certain threshold and are the local maxima within a certain window (to prevent redundant dupes of features).
- For each pixel that meets the criteria in previous step, compute a feature descriptor.
- Compute the Harris cornerness score $R = det(H) - k\ trace(H)^2$. If $R >> 0$ then we have a corner, $R << 0$ implies an edge, and small value of $|R|$ means flat region.

Question - 8

How can I detect a keypoint by examining an auto-correlation surface?

The two-dimensional (2D) autocorrelation function of an image statistically characterises the spatial pattern within that image and presents a powerful tool for analysis. It determines shape preferred orientation, degree of alignment, and distribution anisotropy of image objects. This information can then be extracted to determine the keypoints and build descriptors.

Question - 9

What is the difference between invariance and covariance?

We want corner locations to be invariant to *photometric* transformations and covariant to *geometric* transformations. Invariance means that image is transformed and corner locations do not change. Covariance implies if we have two transformed versions of the same image, features should be detected in corresponding locations.

Question - 10

Why do we say that the Harris corner detector is not scale-invariant?

Harris corner detector uses derivatives. Scaling an image will change the intensity magnitude by multiplying a factor, and the derivative will also be changed. Some points that were earlier under threshold may be detected with such change. Therefore, Harris corner detector is not scale-invariant.

Question - 11

What does it mean to quantize the gradient orientations?

Quantisation of gradient orientations is primarily used for orientation assignment. It means the use of histograms to bin pixels within sub-patches according to the orientation of their gradient. A simple example can be as follows.

- Compute Gradient for each blurred image.
- For region around keypoint, create histogram with n bins for orientation.
- Weight each point with Gaussian window of a certain scale.
- Create keypoint for all peaks with value $>=$ x times max bin, where $x < 1$.
- Final descriptor will be normalised concatenation of all histograms.

Question - 12

What is the effect of gradient magnitude on the histogram of gradient orientations, in the computation of the SIFT descriptor?

When computing the orientation histogram, the increments are weighted by the gradient magnitude and also weighted by a Gaussian window function centred at the interest point and with its size proportional to the detection scale. The SIFT descriptor is computed from sampled values of the gradient orientation and the gradient magnitude over a locally adapted grid around each interest point, with the scale factor determined from the detection scales of the interest point and the orientation determined from the dominant peak (highest magnitude) in a gradient orientation histogram around the interest point.

Question - 13

How can we find to which cluster we should assign a new feature, which was not part of the set of features used to compute the clustering?

We can either check the norm between the new feature and cluster centres and assign, or we can use the same procedure as was used in assigning the clusters originally. Assign the new feature to any of the clusters. Reassign centroid value to be the calculated mean value for each cluster. Reassign data points to nearest centroid. Repeat until data points stay in the same cluster. This way the new feature will be assigned to a cluster, even though it was not part of the original set of features.

Question - 14

How do we use clustering to compute a bag-of-words image representation?

The following steps can be used to compute a bag-of-words image representation.

- The first step is to detect features, extract descriptors from each image in the dataset, and build a visual dictionary. This can be done by using SIFT, KAZE, etc.
- Next, we make clusters from the descriptors, using K-Means, DBSCAN, etc. The centres of each cluster will be used as the vocabularies of visual dictionary.
- For each image, we make frequency histogram from the vocabularies and the frequency of the vocabularies in the image. These histograms are the bag of visual words (BoVW).

Question - 15

When is it more efficient to create an inverted file index to match a query image to other images in the database, rather than comparing the query to all database images without an index?

The brute database search is quite computationally expensive, therefore using more efficient data structure such as inverted file system allows to obtain the query results significantly faster. Also adopting the method of visual words increases searching efficiency. This can be very efficient but only for highly textured object. When it comes to recognise low textured ones, the effectiveness drops significantly.

Question - 16

List three reasons why line fitting (fitting a line to points) is not complete just by extracting edges (finding gradients over a threshold).

If we rely on only edge points for line fitting, then we may face some difficulties.

- **Extra** edge points (clutter), multiple models: We need to know which points go with which line, if any.
- Only some parts of each line detected, and some parts are **missing**: We should find a method to get the line that bridges missing evidence.
- **Noise** in measured edge points, orientations: Need to detect true underlying parameters.

Question - 17

Imagine we are using k-means clustering to compress images by discarding some colors (replacing each color with the mean in its cluster). We also want to compute the error, i.e. the difference between the original image's intensities and the post-compression intensities. Which error would be larger: the one corresponding to $k = 5$ or to $k = 15$? Why?

Dimensions of the compressed images will remain same as that of input images. The size of the compressed image decreases as k decreases. For $k = 15$, the output compressed images would seem reasonably good, and only lose colors which are not visible to a human eye. For $k = 5$, the output compressed images lose a lot of colors and the lossy compression is visible to a human eye. As k decreases further, the output compressed images lose almost all the colors, and the content of the image is also lost. Hence, the error corresponding to $k = 5$ will be more as compared to the case when $k = 15$.

Question - 18

List the steps required to create a mosaic using homography estimation.

The following steps can be used for creating a mosaic.

1. Take pictures such that adjacent pictures have atleast 40% overlap. Order the pictures from left to right.
2. Detect keypoints (DoG, Harris, etc.) and extract local invariant descriptors (SIFT, SURF, etc.) from the input images.
3. Match the descriptors between the images. Use the RANSAC algorithm to estimate a homography matrix using the matched feature vectors.
4. Repeat the above steps for each of the adjacent image pairs.
5. After obtaining Homographies for each of the picture pairs, get the homographies with respect to the central image. Project all images (using inverse warping) on to a blank canvas, then use bilinear interpolation.

Question - 19

Using the following two images write a python code to compute the depth map. You may use any libraries/GitHub code for this task (language: python). However, cite the library/GitHub code you have used. Clearly write the approach followed and the show your results in the report.

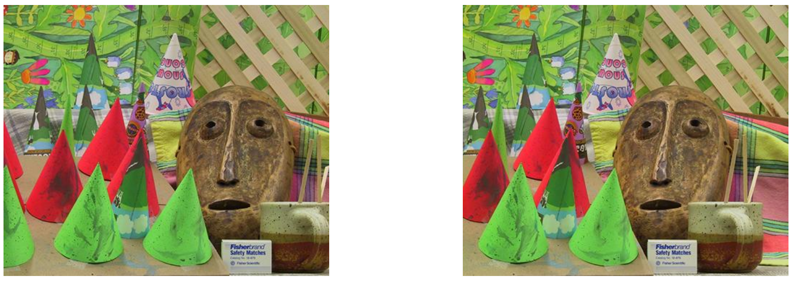


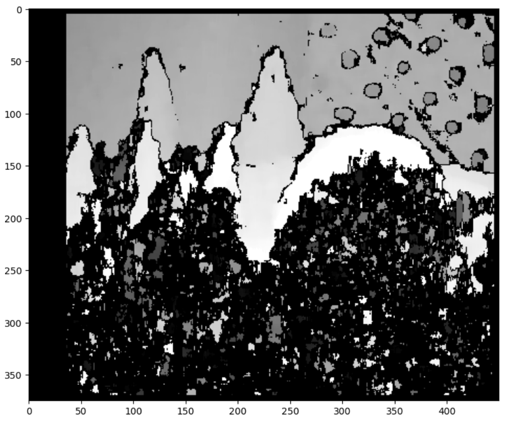
Figure 1: (a) Image taken from the left camera; (b) image taken from the right camera.

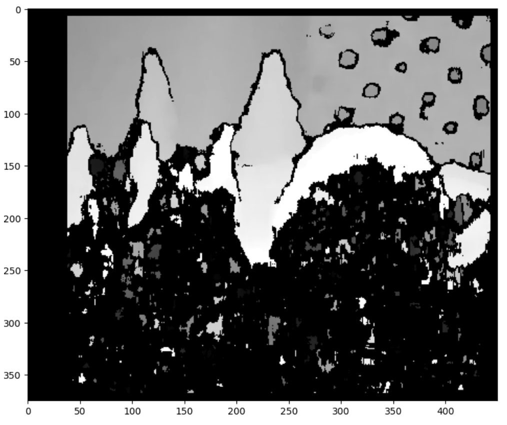
From the depth estimation equation, depth is inversely proportional to disparity, i.e., $Z \propto 1/(x_l - x_r)$. As disparity $(x_l - x_r)$ increases, Z decreases and for lower disparity $(x_l - x_r)$, we would have higher Z.

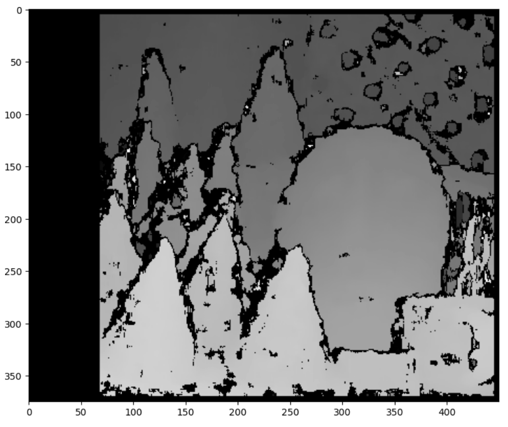
We can use **block matching** technique to find the correspondences between pixels in the two images. Since pixel values may be noisy and influenced by lighting, misalignment, etc., we have to rely on a group of surrounding pixels for comparison.

- Compute similarity score by comparing each block from the left image and in the right image. Record all the similarity scores.
- For the block with highest similarity, return the pixel location at the centre of the block as the best matching pixel.
- If x_l is the column index of the left pixel, and the highest similarity score was obtained for a block on the right image whose centre pixel has column index x_r, we will note the disparity value $|x_l - x_r|$.
- Repeat the matching process for each pixel in the left image and note all the disparity values for the left image pixel index.

The obtained results are shown below.


(a) numDisparities = 32, blockSize = 11


(b) numDisparities = 32, blockSize = 15


(c) numDisparities = 64, blockSize = 11

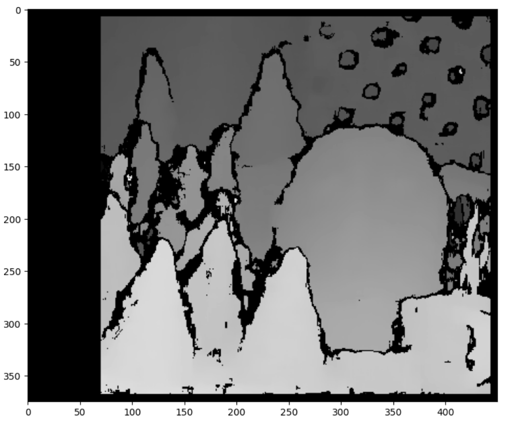

(d) numDisparities = 64, blockSize = 15

Figure 2: Disparity map computed for different values of number of disparities and block sizes.

Number of disparities: Determines the resolution of the stereo map. The higher this value, the higher resolution for depth map, but the algorithm also slows down and will return less coherent results for a very demanding parameter.

Block size: The size of the block of pixels the algorithm will compare for each stereo pair.

Reducing the number of disparities, leads to the nearby objects being categorised as being in same depth. This is resolved by increasing the *numDisparities* parameter.

A smaller value of *blockSize* will give more precise map for every object in image, but a larger block size can tackle lighting and object deformation effects more easily.

References

[1] The Depth I: Stereo Calibration & Rectification | by Ali Yasin Eser | Python in Plain English. https://python.plainenglish.io/the-depth-i-stereo-calibration-and-rectification-24da7b0fb1e0

[2] The Depth II: Block Matching | by Ali Yasin Eser | Python in Plain English. https://python.plainenglish.io/the-depth-ii-block-matching-d599e9372712

[3] Disparity Map Computation in Python and C++ | by Pramod Anantharam | Medium. https://pramod-atre.medium.com/disparity-map-computation-in-python-and-c-c8113c63d701

[4] OpenCV: Depth Map from Stereo Images. https://docs.opencv.org/5.x/dd/d53/tutorial_py_depthmap.html

[5] andijakl/python-depthmaps: Calculate and visualize disparity maps using OpenCV for Python. https://github.com/andijakl/python-depthmaps

Appendix - Python code for computing disparity map

```python
import cv2
import matplotlib.pyplot as plt

leftImg = cv2.imread('left.jpg', 0)
rightImg = cv2.imread('right.jpg', 0)

stereoMatrix = cv2.StereoBM_create(numDisparities = 64, blockSize = 15)
disparityMap = stereoMatrix.compute(leftImg, rightImg)
plt.imshow(disparityMap, 'gray')
plt.show()
```

■■■