Takh2100- week 2 task

Code

Introduction

This weeks task focused on two key areas of computer vision, high pass filtering using Fast Fourier Transform (FFT) and feature detection using Harris corner detection and ORB (Oriented fast and rotated brief). The objective was to deepen our understanding of frequency domain filtering and explore the role of feature descriptors in applications like augmented reality.

High pass filtering in the frequency domain

One of the main tasks was to update the python boilerplate code from last week to implement a high-pass filter in the frequency domain using FFT. By transforming the image or the frame captured by the camera we can manipulate its frequency components before applying an inverse FFT to return back to the spatial domain. This approach allows us to filter out low-frequency components highlighting edges and other high frequency features in an image. By applying the high-pass filter on the camera model side by side to the original image we can see how this affects the photo, i also applied a slider that changes the cutoff frequency value giving us feedback in real time on how the filtering works. By pressing “t” you can also switch between spatial domain to magnitude domain, this gives us insights on the content of the image.

Feature Detection and Descriptors

The second task was about studying a feature descriptor and their role in computer vision applications. Feature descriptors are used to represent keypoints in an image, enabling us to do things such as object recognition and AR applications. I chose to reed further about the Harris detector and ORB.

Harris corner detector

I implemented the Harris corner detector from scratch without using built-in functions from external libraries. To build this detector a step by step approach was applied. The steps was:

1. Converting the image to grayscale: This step is not a must but is good to optimize the application and reduce the computational complexity, since we are looking for corners colors in 3 different channels are not needed and to run this detector in real time the performance increase of having grey scale is more important.
2. Computing image gradients using Sobel filters: We start by using gaussian blur to remove noise ensuring we get better edges and that noise with high frequency doesn’t

interfere with the result. The we use sobel filters to find changes both vertically and horizontally helping us identify edges.

```
smoothed = cv2.GaussianBlur(gray, (5, 5), 1.4
Ix = cv2.Sobel(smoothed, cv2.CV_64F, 1, 0, ksize=3
Iy = cv2.Sobel(smoothed, cv2.CV_64F, 0, 1, ksize=3
Ix2 = Ix ** 2
Iy2 = Iy ** 2
Ixy = Ix * Iy
```

3. Then we compute the products of the gradients which will later help us construct the Harris Matrix. Ix**2 and Iy**2 tells us how how much the intensities change in x and y directions. Xy is intensity changes in both x and y directions.
4. Apply Gaussian filtering to the products: We smooth gradient values to remove noise and smooth them. Ensuring a more accurate harris response.
5. Compute the harris response function R.

```
det_M = S_Ix2 * S_Iy2 - S_Ixy ** 2  # determinant
trace_M = S_Ix2 + S_Iy2  # trace
R = det_M - k * (trace_M ** 2)  # Harris response function
```

The Harris matrix for a small image window is:

$$\begin{bmatrix} I_x^2 & I_x I_y \\ I_y I_x & I_y^2 \end{bmatrix}$$

The corner response function R is calculated using:
R = Det(M)- k* (Trace(M)^2
The higher the R value the more certain it is that it is a corner.

6. Thresholding the response
   The harris response function produces a large number of values, we threshold the values to keep only the strongest corner responses.
   We create a binary conidition where pixels with R>threshold*Max(R) are considered corners and set to 255 while other pixels are set to 0 which is not a corner.

```
corner_response = np.zeros_like(R)
corner_response[R > threshold * R.max()] = 255
```

7. Mark detected corners on the image by copying the original image so we don't modify the original image. Then we mark the corners in red visualizing the corners detected by the algorithm.

ORB Feature Detector

I also applied ORB which is widely used feature detector and descriptor to compare its performance with the Harris detector. ORB combines the FAST keypoint detector and the BRIEF descriptor while adding rotation invariance, making it good for real time applications like augmented reality. I also observed that ORB sticks to a point found and almost always detect it. This is important for AR to match keypoints for overlaying virtual objects onto the real-world scene.

Conclusion

This lab provided hands on experience in feature detection which is an important part of computer vision. Implementing algorithms from scratch deepened my understanding of their underlying mechanics and made it much clearer to how to think about computer vision. I also got to learn about different methods what their pros and cons are and how and when to use which.