Assignment 4

CS283, Computer Vision
Harvard University

Due Friday, Oct. 6 at 5:00pm

This problem set explores structure-from-motion and linear filtering. As usual, the assignment will be submitted via canvas and formatted according to the guidelines. The Hints and Information section is especially helpful this week.

1. *(15 points)* The assignment's data folder contains a .zip file with seven parallel-projection images of a checkered box. The .zip file also contains a .MAT file with the pixel coordinates of $n = 84$ corners detected in each image as an $n \times 2 \times 7$ array named `corners`. (For example, `corners(j,:,i)` gives the (x, y)-coordinates of $\tilde{\mathbf{x}}_j^i$, the j^{th} point in the i^{th} image.)

   (a) Use the factorization algorithm to recover the seven affine cameras and world points from these pixel coordinates. Write a function `[M,T,X]=AffineFactorization(x)` that takes an $n \times 2 \times m$ array of pixel coordinates, corresponding to n world points observed in m views, and returns $2m \times 3$ and $2m \times 1$ matrices `M` and `T` that describe the m affine cameras, and an $n \times 3$ matrix `X` with the coordinates of the n world points. Plot the recovered points using the commands `plot3(X(:,1),X(:,2),X(:,3),'.'); axis equal;`. Submit your plot. Are angles and/or parallelism presented correctly in this reconstruction? Why or why not?

   (b) One measure of the accuracy of the reconstruction is the root-mean-square (RMS) reprojection error

   $$E_{rms} = \left(\frac{1}{nm} \sum_{ij} ||\mathbf{x}_j^i - (\mathbf{M}^i \mathbf{X}_j + \mathbf{t}^i)||^2 \right)^{1/2}$$

   Compute and report the RMS reprojection error for your reconstruction.

   (c) *(Bonus Question: Earn up to 10 extra points)* Upgrade the reconstruction by enforcing constraints corresponding to 'square pixels and zero skew'. Write a function `[M,X]=Upgrade(Min,Xin)` that finds the 3×3 matrix $\mathbf{A}$ that minimizes

   $$\sum_{i=0}^{m-1} \left(\left(\mathbf{m}_{i1} \mathbf{A} \mathbf{A}^\top \mathbf{m}_{i2}^\top \right)^2 + \left(\mathbf{m}_{i1} \mathbf{A} \mathbf{A}^\top \mathbf{m}_{i1}^\top - \mathbf{m}_{i2} \mathbf{A} \mathbf{A}^\top \mathbf{m}_{i2}^\top \right)^2 \right),$$

   where $\mathbf{m}_{ij}$ is the $2i + j$th row of `M`, and uses this matrix to transform both `Xin` and `Min` appropriately. The function should also apply a scale change to (`Xin`, `Min`) such that the average distance from a world point to the world origin is $\sqrt{3}$. Plot the points using the same commands as in part (a), and submit your plot. Compute and report the RMS reprojection error for the upgraded reconstruction. How does it compare to that of part (b) and why?

2. *(10 points)* Let the x^{th} element of a vector $\mathbf{f}$ be written $f(x)$. Furthermore, if $\mathbf{f}$ is a vector with M elements, let these elements be indexed by $x = 0, \ldots, M-1$. Given an M-element vector $\mathbf{f}$, we compute its discrete Fourier transform by expressing it in terms of a linear combination of M complex-valued "basis vectors" $\{\boldsymbol{\phi}_u\}$, $u = 0, \ldots, M-1$ where the x^{th} element of the u^{th} basis vector is given by $\phi_u(x) = e^{j2\pi ux/M} = \cos(2\pi ux/M) + j \sin(2\pi ux/M)$. The linear combination is written $\mathbf{f} = \sum_{u=0}^{M-1} F(u) \boldsymbol{\phi}_u$, and the set of M coefficients $F(u)$—written in vector form as $\mathbf{F}$—is referred to as the *discrete Fourier transform* (DFT) of $\mathbf{f}$. The coefficients $F(u)$ are computed by projecting the vector $\mathbf{f}$ onto a set of orthogonal vectors, and these projections can be written in matrix form as $\mathbf{F} = \boldsymbol{\Phi} \mathbf{f}$.

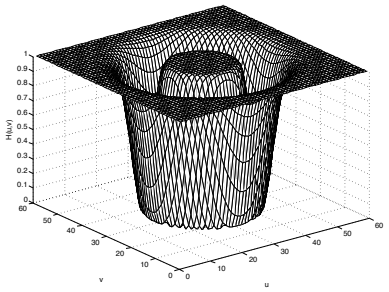

Figure 1: Transfer function $H(u, v)$ of a Gaussian band-reject filter with $M = N = 60$, $D_o = 25$ and $w = 4$.

The DFT of a two-dimensional image $f(x, y)$, $0 \leq x \leq M-1$, $0 \leq y \leq N-1$ can similarly be interpreted as the set of coefficients $F(u, v)$ defining a linear combination of orthogonal, complex-valued "basis images". The image is written

$$f(x, y) = \sum_{u=0}^{M-1} \sum_{v=0}^{N-1} F(u, v) I_{uv}(x, y), \tag{1}$$

where the $(x, y)^{\text{th}}$ element of the $(u, v)^{\text{th}}$ basis image is given by

$$I_{uv}(x, y) = e^{j2\pi(ux/M + vy/N)}. \tag{2}$$

(a) The real parts of the basis images I_{uv} are sinusoids at various orientations and spatial frequencies. Restricting your attention to horizontally-oriented sinusoids (i.e., those with $v = 0$), which value of u produces a basis image I_{u0} with the highest spatial frequency?

(b) Using the fact that $e^{j\pi n} = \cos \pi n + j \sin \pi n = (-1)^n$ when n is an integer, prove that

$$\mathcal{F}\left[f(x, y)(-1)^{x+y}\right] = F(u - M/2, v - N/2), \tag{3}$$

where $\mathcal{F}[\cdot]$ denotes the DFT of the argument. When M and N are even, this has the effect of shifting the DC component of the DFT $F(0, 0)$ to the position $(M/2, N/2)$ in frequency space, and we often refer to this process as 'centering' the DFT.

3. *(20 points)* The image `Cameraman_PeriodicNoise.tif` is available in the assignment's data folder. It is an image corrupted by (you guessed it) periodic noise. In this question you are going to design a filter in the frequency domain to remove the noise. Submit all code for this question.

(a) Display the (centered) DFT of the image (see the Hints and Information below) and manually identify the noise frequencies. These can be removed using a Gaussian *band-reject filter* that eliminates all of the frequency content within a circular ring centered at $F(0, 0)$. The transfer function of the filter is

$$H(u, v) = 1 - \exp\left\{-\frac{1}{2}\left(\frac{D^2(u, v) - D_o^2}{wD(u, v)}\right)^2\right\}, \tag{4}$$

where $D(u, v) = [(u - M/2)^2 + (v - N/2)^2]^{1/2}$ is the distance from (u, v) to the (centered DFT) origin, D_o is the radial center of the frequency band to be rejected, and w is the width of the rejection band. Determine appropriate values for D_o and w to remove your noise. Display your transfer function $H(u, v)$ as a surface using the `surf` or `meshgrid` commands. An example is shown in Fig. 1.

(b) Remove the noise using your filter by multiplying its transfer function by the DFT of the corrupted image and taking the inverse DFT of the result. (Be sure to implement proper centering and 'de-centering' as discussed below in Hints and Information.) In a single figure with four axes, show: i) the corrupted image, ii) your transfer function displayed as an image, iii) the filtered image, and iv) the difference between the two (with appropriately scaled intensity).

(c) Display an image of the impulse response of your filter. By definition, the impulse response of a filter is the output we get when we use a delta function as its input.

(d) Repeat part (b) using values of $w = 20$ and $w = 50$ in your transfer function. Does this noticeably affect your filtered image? How?

Hints and Information

- The `lsqnonlin` function is part of the Matlab Optimization Toolbox that can be used to iteratively minimize a sum-of-squares objective function. A brief example (adapted from the online Matlab documentation) is below. For further information, see the online documentation.

  Suppose we want to find $\mathbf{x} = (x_1, x_2)$ that minimizes

  $$\sum_{k=1}^{10} f_k(\mathbf{x})^2 = \sum_{k=1}^{10} \left(2 + 2k - e^{kx_1} - e^{kx_2}\right)^2 .$$

  We write an M-file—let's call it `myfun.m`—that takes a 2-vector `x` and returns a ten-vector containing values for the ten functions f_k. That's all we have to do. By providing the `lsqnonlin` function with a pointer to `myfun.m` and an initial guess for `x`, the optimization toolbox will do the rest of the work, adjusting this initial guess and making repeated calls to `myfun.m` until a (local) minimum in the objective function is found.

  The function code is

  ```
  function F = myfun(x)
  k = 1:10;
  F = 2 + 2*k-exp(k*x(1))-exp(k*x(2));
  ```

  and this is fed to `lsqnonlin` as in `x=lsqnonlin(@myfun,[0.3,0.4])`, where the second argument is the initial 'guess' for `x`.

  For the particular example in bonus Prob. 1c, your function `myfun.m` will be evaluating the objective function for a nine-element vector representing any 'guess' for the nine elements of the matrix $\mathbf{A}$. Since the camera matrices are also required to compute the objective function (they are treated as constants and not optimized), they must also be given as input. Thus, your function will look like `F=myfun(A,M)`, and your call to `lsqnonlin` will look something like

  ```
  A=lsqnonlin(@myfun,reshape(eye(3),[9,1]),[],[],[],M);
  A=reshape(A,[3 3]);
  ```

  where the identity matrix is being used for the initial guess. The empty matrices `[]` correspond to unused options for `lsqnonlin`. Again, see the online documentation for further details.

- To compute the DFT and inverse DFT in Matlab, use the commands `fft2` and `ifft2`. Note that images must be converted to class `double` before being used as input to `fft2`. Appropriate centering is accomplished using the `meshgrid` command to generate plaid arrays of x and y coordinates, and then using a command of the form `(-1).^(x+y).*double(im)`, where `x` and `y` are your plaid arrays. Given a complex valued DFT `F` it's magnitude (or spectrum) is given by the command `abs(F)`. See the online Matlab Image Processing Toolbox User Guide for more information.

- When displaying an 8-bit (class `uint8`) grayscale image in Matlab, values of 0 are depicted as black and values of 255 are depicted as white. When displaying a real-valued function (class `double`) as in image, values of 0 and 1 are assigned black and white, respectively, and values outside of that range are clipped. In many cases, you want to display these clipped values, and this can be done by specifying the black and white grayscale limits. As examples, the command `imshow(f,[-1 5])` assigns grayscale intensity values linearly to function values between -1 and 5, and `imshow(f,[])` automatically chooses the maximum and minimum function values for the grayscale limits.

- The DC component $F(0,0)$ of the DFT of an image is the average value of the image, and since an image is positive valued, this component tends to be very large. As a result, displaying the DFT using `imshow(F,[])` is not very informative. A more useful depiction is obtained by setting the DC component to zero prior to display. Here is some sample code assuming that `F` is the (centered) DFT of an image with width and height `M` and `N`:

```
Fdisplay=F;
Fdisplay(M/2+1,N/2+1)=0;
imshow(abs(Fdisplay),[]);
```

- Note that relative to the notation used above the center (assuming even-valued `M` and `N`) of a centered DFT is shifted down and right by one pixel because Matlab's indexing begins at 1. This should be taken into account when computing the distance $D(u, v)$ in Prob. 3.
- Filtering in the frequency domain with "centering" consists of these steps:
  1. Multiply the input image by $(-1)^{x+y}$ to center the transform
  2. Compute the DFT $F(u, v)$
  3. Multiply $F(u, v)$ by the "filter transfer function" $H(u, v)$
  4. Compute the inverse DFT
  5. Obtain the real part
  6. Multiply by $(-1)^{x+y}$