Assignment 5 Solutions

CS283, Computer Vision

1. (a) We firstly write the Taylor expansion up to the h^3 term,

$$f(x+h) = f(x) + hf'(x) + \frac{h^2}{2}f''(x) + \frac{h^3}{3!}f^{(3)}(x) + O(h^4). \tag{1}$$

Then, by replacing h as appropriate, we also have

$$f(x-h) = f(x) - hf'(x) + \frac{h^2}{2}f''(x) - \frac{h^3}{3!}f^{(3)}(x) + O(h^4), \tag{2}$$

$$f(x+2h) = f(x) + 2hf'(2x) + \frac{4h^2}{2}f''(x) + \frac{8h^3}{3!}f^{(3)}(x) + O(h^4), \tag{3}$$

$$f(x-2h) = f(x) - 2hf'(2x) + \frac{4h^2}{2}f''(x) - \frac{8h^3}{3!}f^{(3)}(x) + O(h^4). \tag{4}$$

In order to obtain an order h^4 approximation error, we need to find a linear combination of (1), (2), (3), (4), such that h^2 and h^3 terms cancel out. Then, it is easy to see (either by solving a simple linear system, or just by observation) that this combination is the following

$$\begin{aligned} 8f(x+h) - 8f(x-h) - f(x+2h) + f(x-2h) &= 12hf'(x) + O(h^4) \Rightarrow \\ f'(x) &= \frac{8f(x+h) - 8f(x-h) - f(x+2h) + f(x-2h)}{12h} + O(h^4). \end{aligned} \tag{5}$$

Note that if one only uses extensions up to h, and then simply add the expressions for h, $-h$, $2h$, and $-2h$, then the order of the approximation error is still $O(h^2)$. The ability to obtain a higher order approximation error comes from the clever linear combination, which cancels out higher order derivatives.

(b) Consider the *discrete convolution* formula,

$$(f * g)[n] = \sum_{n=-\infty}^{+\infty} f[n-m]\, g[m], \tag{6}$$

where we have discretized the signal f and filter g by taking samples at intervals of length h. Compare (6) with the, analogously discretized, form of (5),

$$f'[n] = -\frac{1}{12}f[n+2] + \frac{2}{3}f[n+1] - \frac{2}{3}f[n-1] + \frac{1}{12}f[n-2]. \tag{7}$$

We observe that, in order for (7) to be of the form (6), filter g must on the (discrete) interval $\{-2,-1,0,1,2\}$ have respective values $\left\{-\frac{1}{12}, \frac{2}{3}, 0, -\frac{2}{3}, \frac{1}{12}\right\}$, and be identically zero everywhere else. Therefore, the convolution kernel corresponding to (5) is

$$\left[-\frac{1}{12}, \frac{2}{3}, 0, -\frac{2}{3}, \frac{1}{12}\right]. \tag{8}$$

It is important to take into account the "flipping" of indices in the convolution formula (6). If we do not and use the kernel $\left[\frac{1}{12}, -\frac{2}{3}, 0, \frac{2}{3}, -\frac{1}{12}\right]$, in order to obtain the equivalent of (5) we need to perform *discrete correlation*,

$$(f \star g)[n] = \sum_{n=-\infty}^{+\infty} f[n+m]\, g[m], \tag{9}$$

instead of discrete convolution (compare (9) with (6)).

2. We list below an implementation for parts (a)-(c). Some comments on the different parts:

   (a) Equation (4.21) of [3] book has a typo, as the multiplicative factor should be $1/\sigma^4$ instead of $1/\sigma^3$. However, whichever you use should not change your results: zero-crossings of a function are not affected by multiplication by a constant factor. This is why we also ignore the various $1/(2\pi)$ constants in the formulae involving Gaussians.

   (b) It is possible to look for windows with exactly two zero-crossings by clever use of subscript indexing. However, an easier (and potentially faster) way is to use convolutions: the horizontal zero-crossings of `Ilog` can be found by convolving its sign, `sign(Ilog)` with the 2D kernel

$$\begin{bmatrix} 1 & -1 \\ 1 & -1 \end{bmatrix}, \tag{10}$$

   and similarly for the vertical.

   In general, whenever performing a "sliding-window" operation, a good strategy is to consider whether it can be expressed in terms of a convolution, or some function of the convolution output. Then, we can make use of existing specialized routines for convolution, which are often heavily optimized and therefore can help achieve good (or optimal) performance at minimum effort[1]. This is particularly true if the convolution kernel is *separable*, that is, it can be written as the outer product of two 1D kernels. In this case, 2D convolution can be replaced by two successive 1D convolution operations, for additional savings. This property holds for the kernel of (10), which also provides a good example. We have,

$$\begin{bmatrix} 1 & -1 \\ 1 & -1 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & -1 \end{bmatrix}, \tag{11}$$

   (and, again, similarly for the kernel for the vertical zero-crossings); therefore, 2D convolution with the kernel of (10) can be replaced with a 1D convolution with the vector $[1 - 1]$, followed by another 1D convolution with the vector $[11]^T$. To see why this is true, observe that the matrix multiplication in (11) is equivalent to the convolution of the two vectors, and remember that convolution has the associativity property.

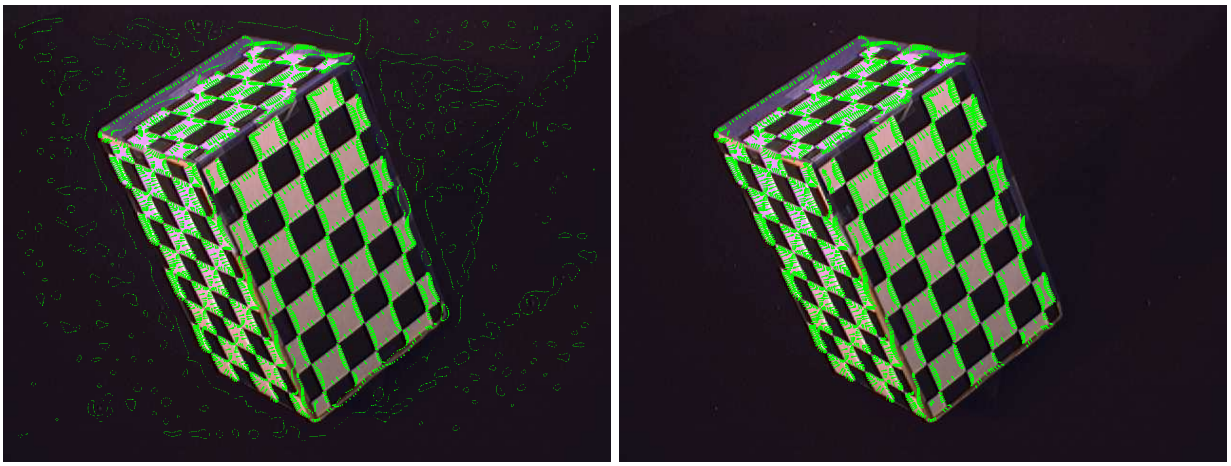
   Care needs to be taken for image patches near the boundary. In MATLAB, you can use the `conv2` function with the argument `'valid'`, to ignore patches at the boundary, or with the argument `'same'`, to evaluate convolution at the boundary through zero-padding. Note also that a potentially much faster implementation can be obtained by using `imfilter` instead of `conv2`, which has a processor-optimized implementation and which takes care to automatically take advantage of the separability property described above whenever possible (note that `imfilter` by default performs correlation and not convolution, so you need to call it with the right arguments).

   With respect to the subpixel localization, note that Equation (4.25) of [3] also has a typo: the formula should be used directly with the values of the Laplacian-of-Gaussian response, instead of their sign $S(x)$. The implementation below uses the correct formula.

   (c) For sampling the gradient, we use linear interpolation, which MATLAB provides very conveniently through the `interp2` function. As you can see in figure 1(a), the detected edges can be very noisy. One way to deal with this issue is prune out edgels where the gradient vector has a small magnitude, through an additional thresholding step (this is not required by the assignment). The results of using the mean of the gradient magnitude values as threshold are shown in figure 1(b), where we can see that the improvement is significant.

```
1 function [X, G]=edgels(im, sigma)
2 %EDGELS   Edge detector based on Laplacian of Gaussian
3 %  [X,G]=EDGELS(IM,SIGMA) computes edges in image IM at the scale given by
4 %  the scalar value SIGMA. SIGMA is the standard deviation (in pixels) of
```

[1] In this particular problem, using convolution is slower than a purely index-based solution, as the latter avoids a lot of overhead from multiplications of the form $(\pm 1)(\pm 1)$.



(a) (b)

Figure 1: Output of `edgels.m` superimposed over image `calib_right.bmp`: (a) wihout thresholding; (b) after thresholding.

```
5  %  the Gaussian filters used to computed first and second derivatives in
6  %  the image.
7  %
8  %  The output X is an Nx2 array, where each row is the (x,y) position of an
9  %  edge point (or edgel). Edgels are localized to sub-pixel accuracy, so
10 %  these coordinates are not generally integers. The output G is another
11 %  Nx2 array that stores the gradient vector at each edge point.
12 %
13 %  The norm of each row of Dx provides the gradient magnitudes, for example, and
14 %  edgels with small gradient magnitudes can be removed as a post-process,
15 %  by performing additional operations on the [X,G] returned by this function.
16 %
17 %  Reference:
18 %  ----------
19 %  Closely related to the algorithm described in Chapter Four of R.
20 %  Szeliski, "Computer Vision: Algorithms and Applications", available in
21 %  draft form at http://research.microsoft.com/en-us/um/people/szeliski/Book/
22
23 % Skeleton code for CS283. TZ, Harvard University, October 2009.
24
25 % convert image to double grayscale (so intensity values are in [0,1])
26 if ndims(im) == 3,
27     im = im2double(rgb2gray(im));
28 else
29     im = im2double(im);
30 end
31
32 % LoG response
33 Ilog = conv2(im, lapgauss(sigma), 'same');
34
35 % Horizontal
36 map = conv2(sign(Ilog), [1 -1; 1 -1], 'same');
37 [vY, vX] = find(abs(map) == 4);
38
39 % Vertical
40 map = conv2(sign(Ilog), [1 1; -1 -1], 'same');
41 [hY, hX] = find(abs(map) == 4);
42
43
44 % sub-pixel localization of these surviving zero-crossings. Use Szeliski
45 %   equation 4.25. Note that the equation has a type-o; the formula should use
```

```
%   directly the values of the LoG response Ilog, instead of their signs
%   S(Ilog).
sz = size(Ilog);

% Localize horizontal edges
vX_a = (Ilog(sub2ind(sz, vY, vX)) .* (vX + 1) - Ilog(sub2ind(sz, vY, vX + 1)) .* ...
    vX) ...
       ./ (Ilog(sub2ind(sz, vY, vX)) - Ilog(sub2ind(sz, vY, vX + 1)));
vX_b = (Ilog(sub2ind(sz, vY + 1, vX)) .* (vX + 1) - Ilog(sub2ind(sz, vY + 1, vX ...
    + 1)) .* vX) ...
       ./ (Ilog(sub2ind(sz, vY + 1, vX)) - Ilog(sub2ind(sz, vY + 1, vX + 1)));
X = [0.5*(vX_a + vX_b) vY+0.5];

% Localize vertical edges
hY_a = (Ilog(sub2ind(sz, hY, hX)) .* (hY + 1) - Ilog(sub2ind(sz, hY + 1, hX)) .* ...
    hY) ...
       ./ (Ilog(sub2ind(sz, hY, hX)) - Ilog(sub2ind(sz, hY + 1, hX)));
hY_b = (Ilog(sub2ind(sz, hY, hX + 1)) .* (hY + 1) - Ilog(sub2ind(sz, hY + 1, hX ...
    + 1)) .* hY) ...
       ./ (Ilog(sub2ind(sz, hY, hX + 1)) - Ilog(sub2ind(sz, hY + 1, hX + 1)));
X = [X; hX + 0.5 0.5 * (hY_a + hY_b)];


% sample the image gradient at these localized points. (The
%   localized points are assumed to be in an Nx2 array X
%   with rows of the form (x,y).)
[Dx, Dy]=deriv(sigma);
Ix = conv2(im, Dx, 'same');
Iy = conv2(im, Dy, 'same');
G = [interp2(Ix, X(:, 1), X(:, 2)), interp2(Iy, X(:,1), X(:,2))];

%%%
%%% SUB-ROUTINES
%%%

function [Dx,Dy] = deriv(sigma)
% Create derivative-of-Gaussian kernels for horizontal and vertical derivatives.
% Input SIGMA is the standard deviation of the Gaussian. Use equation 4.21
% in the Szelsiki book. Note that the equation has a type-o; the multiplicative
% factor 1/sigma^3 should instead be 1/sigma^4; this does not affect the results
% though.

w = 2 * floor(ceil(7 * sigma) / 2) + 1;
[xx, yy] = meshgrid(- (w - 1) / 2:(w - 1) / 2, - (w - 1) / 2:(w - 1) / 2);

Dx = - xx / (sigma ^ 4) .* exp(- 0.5 * (xx .^ 2 + yy .^ 2) / (sigma ^ 2));
Dy = - yy / (sigma ^ 4) .* exp(- 0.5 * (xx .^ 2 + yy .^ 2) / (sigma ^ 2));

return;

function DD = lapgauss(sigma)
% Create Laplacian-of-Gaussian kernel with standard deviation SIGMA.
% Use equation 4.23 in the Szelsiki book.

% use an odd-size square window with length greater than 5 times sigma
w = 2 * floor(ceil(7 * sigma) / 2) + 1;
[xx,yy]=meshgrid(-(w-1)/2:(w-1)/2,-(w-1)/2:(w-1)/2);

% Equation 4.23 from Szeliski s book. (Actually, he has a small type-o;
%   the equation here is correct)
DD = (1 / (sigma ^ 4)) .* ((xx .^ 2 + yy .^ 2) / (2 * sigma ^ 2) - 1) .* exp(- ...
     (xx .^ 2 + yy .^ 2) / (2 * sigma ^ 2));

return;
```

```
%%
```

```
% CS283 Fall 2015
% Student: Ioannis Gkioulekas
% Homework assignment 5
% Question 2
%
% Plot edgels
%
I = imread('../data/calib_right.bmp');
[x, g] = edgels(I, 5);
% Calculate gradient magnitude
gmag = sqrt(sum(g .^ 2, 2));

% Show all edgels
figure;
imshow(I);
hold on
quiver(x(:, 1), x(:, 2), g(:, 1), g(:, 2), 'Color', 'g');

% Only show edgels with gradient magnitude above the mean
idx = find(gmag > mean(gmag));
figure;
imshow(I);
hold on
quiver(x(idx, 1), x(idx, 2), g(idx, 1), g(idx, 2), 'Color', 'g');
```

3. (a) The very first thing to notice is that the Hessian matrix $\mathcal{H}(x, y)$ the problem describes is not the same as th second-moment matrix (or auto-correlation matrix) $\mathcal{M}_W(x, y)$ used in the derivation of the Harris corner detector (Section 4.1.1 of [3]). Therefore, we cannot directly use the "cornerness" expression of that detector (Equation 4.9 of [3]) and simply substitute the determinant and trace of $\mathcal{M}_W(x, y)$ with those of $\mathcal{H}(x, y)$. We need to derive a new expression tailored to the properties of $\mathcal{H}(x, y)$.

From figure 1 of Assignment 5, corners correspond to the "hyperbolic" case, when we have two eigenvalues of opposite sign. Furthermore, in strong corners we would expect both eigenvalues to be large in absolute value, but also of comparable scale to each other (as otherwise we approach the "parabolic" case, which is essentially an edge).

The opposite sign property implies that, in corners, we want $K(x, y)$, which is equal to the product of the eigenvalues, to be negative. Furthermove, as the two eigenvalues are large, we want the absolute value of $K(x, y)$, to be large. Combining the two, a good "cornerness" criterion should favor large values of $-K(x, y)$. At the same time, the fact that the two eigenvalues are of opposite sign but comparable scale implies that their sum should be small in absolute value. As $H(x, y)$ i equal, up to a scale factor, to this sum, a good "cornerness" criterion should favor small values of the absolute value of $H(x, y)$.

Two algebraic expressions that have the above properties are

$$C_1(x, y) = -\frac{K(x, y)}{|H(x, y)| + \alpha}, \tag{12}$$

$$C_2(x, y) = -K(x, y) - \beta |H(x, y)|, \tag{13}$$

where α and β positive constants (α in particular is important for numerical stability, considering that we expect $|H(x, y)|$ to be close to zero near corners). We can determine these constants through trial-and-error. In the following, we refer to (12) as the ratio criterion, and to (13) as the sum criterion. In both cases, the constants control the relative importance of the Gaussian and mean curvature: in the ratio criterion, larger values of α emphasize the role of $K(x, y)$; the same is true in the sum criterion for smaller values of β.

Note that, in both the ratio and sum expressions, we could replace $|H(x, y)|$ with $H^2(x, y)$. The resulting expressions would still have the two qualitative properties we described above, and therefore would also be good "cornerness" criteria. However, we will not experiment with these alternatives in the remaining of this problem.

As a last note before proceeding, it is worth comparing the ratio and sum criteria with those based on the second-moment matrix. Comparing Equations (12) and (13) above with Equations (4.9) and (4.11) of [3], we see that the main difference is the negative sign in front of the determinant factor. For both matrices, it is important that the two eigenvalues, and therefore their product, have a large magnitude. However, for the Hessian matrix, it is also important that their product is negative. Without accounting for this change, our Hessian-based criteria would be producing very spurious detections.

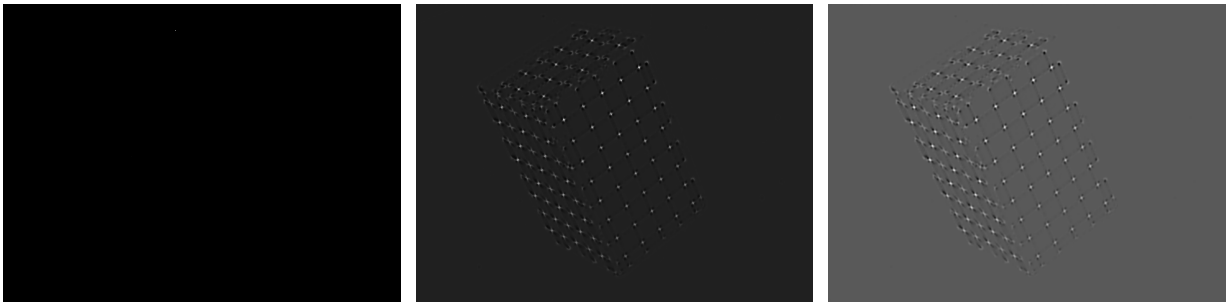
(b) We show below implementations of the ratio and sum criteria, respectively.

```
%%
% CS283 Fall 2015
% Student: Ioannis Gkioulekas
% Homework assignment 2
% Question 3, part b
%
% Detect corner strength (ratio criterion)
%
function C = cornerness_ratio(I, sigma, alpha)
% convert image to double grayscale (so intensity values are in [0,1])
I = im2double(I);

% discrete approximations for second-order derivatives
Gxx = conv2(fspecial('gaussian', 7 * sigma, sigma), [1 0 -2 0 1] / 4,...
                          'same');
Ixx = conv2(I, Gxx, 'same');

Gyy = conv2(fspecial('gaussian', 7 * sigma, sigma), [1 0 -2 0 1]' / 4,...
                          'same');
Iyy = conv2(I, Gyy, 'same');

Gxy = conv2(conv2(fspecial('gaussian', 7 * sigma, sigma), [1 0 -1] / 2,...
                          'same'), [1 0 -1]' / 2, 'same');
Ixy = conv2(I, Gxy, 'same');

% Gaussian and mean curvature
K = Ixx .* Iyy - Ixy .^ 2;
H = 1 / 2 * (Ixx + Iyy);

% cornerness criterion
C = - K ./ (abs(H) + alpha);
```

```
%%
% CS283 Fall 2015
% Student: Ioannis Gkioulekas
% Homework assignment 2
% Question 3, part b
%
% Detect corner strength (sum criterion)
%
function C = cornerness_sum(I, sigma, beta)
% convert image to double grayscale (so intensity values are in [0,1])
I = im2double(I);

% discrete approximations for second-order derivatives
Gxx = conv2(fspecial('gaussian', 7 * sigma, sigma), [1 0 -2 0 1] / 4,...
                          'same');
Ixx = conv2(I, Gxx, 'same');

Gyy = conv2(fspecial('gaussian', 7 * sigma, sigma), [1 0 -2 0 1]' / 4,...
                          'same');
Iyy = conv2(I, Gyy, 'same');

Gxy = conv2(conv2(fspecial('gaussian', 7 * sigma, sigma), [1 0 -1] / 2,...
```

```
                          'same'), [1 0 -1]' / 2, 'same');
Ixy = conv2(I, Gxy, 'same');

% Gaussian and mean curvature
K = Ixx .* Iyy - Ixy .^ 2;
H = 1 / 2 * (Ixx + Iyy);

% cornerness criterion
C = - K - beta * abs(H);
```

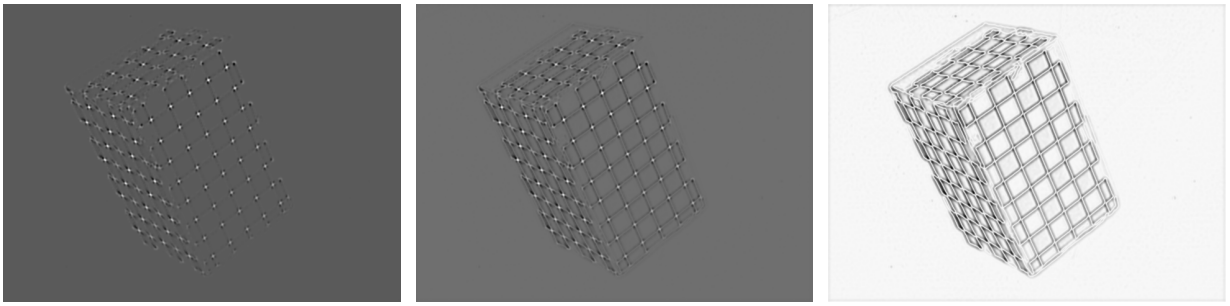
Figure 2 shows the results produced using the ratio criterion, for different values of α. Note that, when $\alpha = 0$, the result is an almost all-black image, except for a single white pixel (hard to see in figures of this scale). This shows how using a non-zero α can stabilize the results. We observe that $K(x, y)$ is more characteristic of corners (figure 2(c)), but a combination of both curvatures can make them even more prominent (figure 2(b)).



(a) $\alpha = 0$ (b) $\alpha = 0.01$ (c) $\alpha = 1$

Figure 2: Output of `cornerness_ratio.m` on the grayscale version of `calib_right.bmp` for different values of α.

Figure 3 shows the results produced using the sum criterion, for different values of β. Note that, when $\beta = 0$ (figure 3(a)), only $K(x, y)$ is used and we get results similar to figure 2(c): we see again that $K(x, y)$ alone can be used to detect corners. When, on the other hand, we use a large value for β (figure 3(c)), we observe that we also get large responses on edges. As we interpolate between these two extremes (figure 3(b)), edges become less prominent.



(a) $\beta = 0$ (b) $\beta = 0.01$ (c) $\beta = 1$

Figure 3: Output of `cornerness_sum.m` on the grayscale version of `calib_right.bmp` for different values of β.

(c) Using the same line of thought as in question 2, part b, instead of using some index-based method for performing window-based non-maxinum suppression, we would like to do it using convolution instead. This is not possible, however: convolution computes weighted sums of all of the values in a window around every pixel, whereas here we want to compute the maximum value in the same

window instead of a sum. With this in mind, we can consider using a class of filtering operators called *morphological operators*. Specifically, we can use the operation called *dilation*, which can be intuitively described as a convolution using maximum instead of sum. It is worth pointing out that morphological operators are often described as *non-linear convolution operators*. The reason being that their form is very similar to standard, linear convolution, but with the linear sum operator replaced with a non-linear operator (in the case of dilation, maximum).

As with linear convolution, there are several efficient algorithms for computing morphological operators, which have mature implementions in MATLAB and in several other numerical libraries and environments. Note that the separability discussion in question 2 about convolution also applies for morphological operators. Two-dimensional dilation is provided in MATLAB by the function `imdilate`. For more information on morphological operators, you can read section 3.3.2 of [3], Section 9 of [2], or you can check out the very detailed MATLAB tutorial[2], especially the page on dilation[3]. We show below an implementation of non-maximum suppression, using `imdilate`.

Finally, note that the function `ordfilt2`, which was used in several solutions to implement non-maximum suppression, implements another kind of morphological operator called ordinal filtering. Ordinal filtering takes an order parameter k, and what it does is return the k-th largest value in a neighborhood around the pixel. We see, then, that dilation is equivalent to ordinal filtering with $k = 1$.

```
%%
% CS283 Fall 2014
% Student: Ioannis Gkioulekas
% Homework assignment 2
% Question 3, part c
%
% Non-maximum suppression
%
function Co = nonmax_suppression(C)

neighborhood = strel('square', 3);
Co = C .* (C > imdilate(C, neighborhood));
```

We use the above with the cornerness results from figure 2(b), as shown in the following script. The resulting corners are shown in figure 4.

```
%%
% CS283 Fall 2015
% Student: Ioannis Gkioulekas
% Homework assignment 5
% Question 3, part c
%
% Extract and plot corners
%
I = imread('../data/calib_right.bmp');
I = im2double(rgb2gray(I));
C = cornerness_ratio(I, 2, 0.01);
Co = nonmax_suppression(C);
inds = find(Co > 0.02); [x y] = ind2sub(size(I), inds);
figure; imshow(I); hold on; plot(y, x, 'gx', 'Markersize', 10)
```

4. There are (at least) two ways to solve this problem. The first uses directly the definition of continuous convolution, and requires long algebraic manipulations. The second uses the properties of Fourier transform, and is (in this TA's opinion anyway) much simpler. Here, we adopt the second approach.

   First, we derive the Fourier transform of the Gaussian function. We use here a proof taken by [1], that makes nice use of many of the properties of the Fourier transform. Any other proof (or even looking up

[2] http://www.mathworks.com/help/images/morphological-filtering.html
[3] http://www.mathworks.com/help/images/morphology-fundamentals-dilation-and-erosio n.html

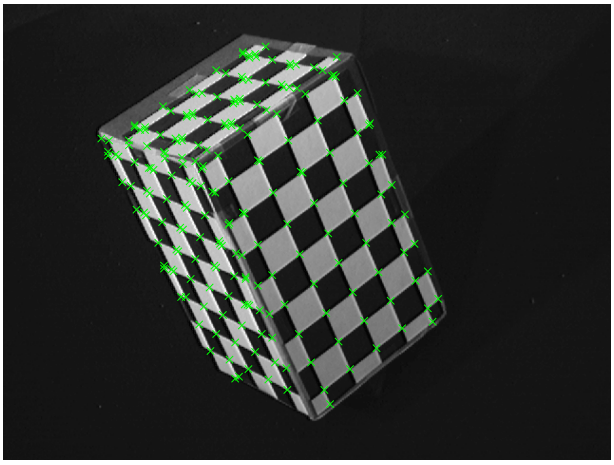


Figure 4: Detected corners on the grayscale version of `calib_right.bmp`.

the transform in Fourier transform tables) is valid. Consider the derivative of the Gaussian function,

$$\frac{\mathrm{d}G_s(x)}{\mathrm{d}x} = -\frac{xG_s(x)}{s}. \tag{14}$$

We denote by $\mathcal{F}\{\cdot\}$ the Fourier transform. Then from (14), the derivative property of the Fourier transform and its dual, we obtain

$$\begin{aligned} j\omega\mathcal{F}\{G_s\}(\omega) &= -\frac{j}{s}\frac{\mathrm{d}\mathcal{F}\{G_s\}(\omega)}{\mathrm{d}\omega} \Rightarrow \\ \omega\mathcal{F}\{G_s\}(\omega) &= -\frac{1}{s}\frac{\mathrm{d}\mathcal{F}\{G_s\}(\omega)}{\mathrm{d}\omega} \Rightarrow \\ \frac{1}{\mathcal{F}\{G_s\}(\omega)}\frac{\mathrm{d}\mathcal{F}\{G_s\}(\omega)}{\mathrm{d}\omega} &= -s\omega. \end{aligned} \tag{15}$$

Recognizing the left-hand-side part of (15) as the derivative of the logarithm function, we integrate both parts to obtain

$$\begin{aligned} \int_0^{\omega} \frac{1}{\mathcal{F}\{G_s\}(\omega')}\frac{\mathrm{d}\mathcal{F}\{G_s\}(\omega')}{\mathrm{d}\omega'} &= -\int_0^{\omega} s\omega' \Rightarrow \\ \ln\left[\mathcal{F}\{G_s\}(\omega)\right] - \ln\left[\mathcal{F}\{G_s\}(0)\right] &= -\frac{s\omega^2}{2}. \end{aligned} \tag{16}$$

We know that $\mathcal{F}\{G_s\}(0)$ is equal to the DC-component of G_s, for which from the problem description we have that

$$\mathcal{F}\{G_s\}(0) = \int_{-\infty}^{+\infty} G_s(x)\,\mathrm{d}x = 1. \tag{17}$$

Combining (17) with (16), we obtain

$$\ln\left[\mathcal{F}\{G_s\}(\omega)\right] = -\frac{s\omega^2}{2}, \tag{18}$$

and by exponentiating both sides, we get the Fourier transform of the Gaussian function as

$$\mathcal{F}\{G_s\}(\omega) = \exp\left(-\frac{s\omega^2}{2}\right). \tag{19}$$

Now to prove the semigroup property, we will use the convolution theorem for the Fourier transform. We have

$$\begin{aligned}
\mathcal{F}\left\{G_{s_1} * G_{s_2}\right\}(\omega) &= \mathcal{F}\left\{G_{s_1}\right\}(\omega)\,\mathcal{F}\left\{G_{s_2}\right\}(\omega) \\
&\overset{(19)}{=} \exp\left(-\frac{s_1\omega^2}{2}\right)\exp\left(-\frac{s_2\omega^2}{2}\right) \\
&= \exp\left(-\frac{(s_1+s_2)\,\omega^2}{2}\right) \\
&\overset{(19)}{=} \mathcal{F}\left\{G_{s_1+s_2}\right\}(\omega) \Rightarrow \\
\left(G_{s_1} * G_{s_2}\right)(x) &= G_{s_1+s_2}(x)\,,
\end{aligned} \tag{20}$$

as required.

The semigroup property has an interesting relation to propability theory. Consider two *independent* and zero-mean Normal random variables: $x_1 \sim \mathcal{N}\left(0, \sigma_1^2\right)$ and $x_2 \sim \mathcal{N}\left(0, \sigma_2^2\right)$, where σ_1^2 and σ_2^2 are their respective *variances* (*not* standard deviations). Then, from standard probability we know that their sum $x = x_1 + x_2$ also follows a Normal distribution, whose mean and variance is the sum of the means and variances, respectively, of the distributions of the summands. That is, $x \sim \mathcal{N}\left(0, \sigma_1^2 + \sigma_2^2\right)$. If we consider the form of the *probability density function* of the Normal distribution $\mathcal{N}\left(0, \sigma^2\right)$,

$$p(x) = \frac{1}{\sqrt{2\pi\sigma^2}}\exp\left(-\frac{x^2}{2\sigma^2}\right), \tag{21}$$

we see that it is the same as the Gaussian kernel we used above, by identifying the scale s with the variance σ^2 (once again, not with the standard deviation). It is no coincidence that the Gaussian kernel integrates to 1, like all probability density functions. So, the semigroup property can be interpreted as summing independent Normally distributed random variables. This equivalent interpretation should be obvious if we consider that, when summing two independent random variables, the probability density function of their sum is equal to the convolution of the probability density functions of the two random variables. It is also worth noting that this property about sums of random variables is usually proved using the so-called *characteristic function* of the distribution. For real random variables, the characteristic function is exactly equal to the Fourier transform of the probability density function of the random variable, making this property a restatement of the convolution theorem.

References

[1] Billingsley, P. *Probability and Measure*. Wiley, 1995.

[2] Gonzalez, R., and Woods, R. *Digital Image Processing*. Prentice Hall, 2007.

[3] Szeliski, R. *Computer Vision: Algorithms and Applications*. Springer, 2011.