6.003: Signal Processing

Filtering and Inverse Filtering

Applications of the 2D DFT:

$$F[k_x, k_y] = \frac{1}{N_x N_y} \sum_{nx=0}^{N_x-1} \sum_{ny=0}^{N_y-1} f[n_x, n_y]\, e^{-j\left(\frac{2\pi k_x}{N_x} n_x + \frac{2\pi k_y}{N_y} n_y\right)}$$

$$f[n_x, n_y] = \sum_{kx=0}^{N_x-1} \sum_{ky=0}^{N_y-1} F[k_x, k_y]\, e^{j\left(\frac{2\pi k_x}{N_x} n_x + \frac{2\pi k_y}{N_y} n_y\right)}$$

Filtering

One of the most important applications of the 2D DFT is in computing the responses of image processing systems.

- If a system is linear and time invariant, then its response to any input $x[n_x, n_y]$ is $(x*h)[n_x, n_y]$ where $h[n_x, n_y]$ is the unit-sample response of the system.

$$x[n_x, n_y] \longrightarrow \boxed{h[n_x, n_y]} \longrightarrow (x*h)[n_x, n_y]$$

- **Convolution** follows directly from linearity and time-invariance.

Convolution can be implemented in the **frequency domain**.

$$\left.\begin{array}{l} x[n_x, n_y] \overset{\text{DTFT}}{\Longleftrightarrow} X(\Omega_x, \Omega_y) \\ h[n_x, n_y] \overset{\text{DTFT}}{\Longleftrightarrow} H(\Omega_x, \Omega_y) \end{array}\right\} (h * x)[n_x, n_y] \overset{\text{DTFT}}{\Longleftrightarrow} H(\Omega_x, \Omega_y)X(\Omega_x, \Omega_y)$$

Using the DFT **speeds** computation (but makes convolution **"circular."**)

$$\left.\begin{array}{l} x[n_x, n_y] \overset{\text{DFT}}{\Longleftrightarrow} X[k_x, k_y] \\ h[n_x, n_y] \overset{\text{DFT}}{\Longleftrightarrow} H[k_x, k_y] \end{array}\right\} (h \circledast x)[n_x, n_y] \overset{\text{DFT}}{\Longleftrightarrow} H[k_x, k_y]X[k_x, k_y]$$

2D Filtering

How can we remove the high frequencies from this image.

One method is to transform, zero out the high-frequency components, and inverse transform.

2D Filtering

Transform, zero out the high-frequency components, and inverse transform.

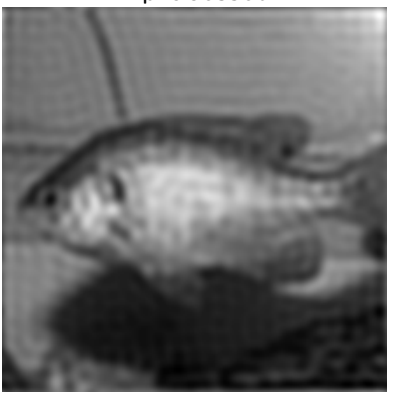
```
from lib6003.fft import fft2,ifft2
from lib6003.image import png_read,show_image

f = png_read('bluegill.png')
R,C = f.shape

F = fft2(f)
for kr in range(-R//2,R//2+1):
    for kc in range(-C//2,C//2+1):
        if (kr**2+kc**2)**0.5 > 25:
            F[kr,kc] = 0

blurred = ifft2(F)
show_image(blurred)
```

2D Filtering

Transform, zero out the high-frequency components, and inverse transform.

original

processed



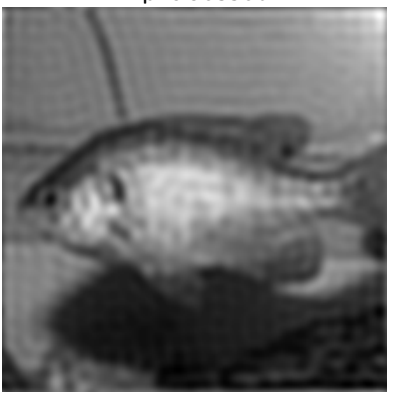
Is this what you expected?

2D Filtering

Transform, zero out the high-frequency components, and inverse transform.

original

processed



The ripples in space result from discontinuities in the transform.

2D Filtering

Zeroing out frequency components is equivalent to filtering by

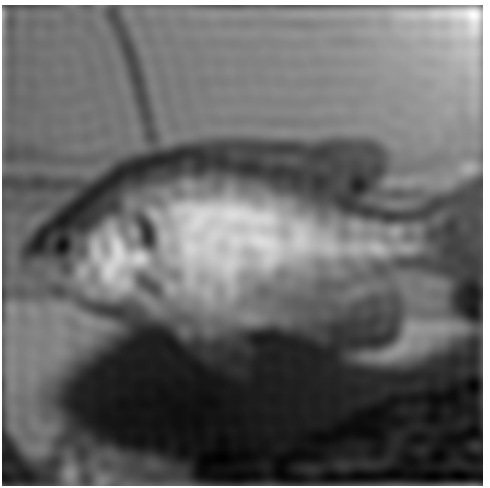
$$H_L[k_r, k_c] = \begin{cases} 1 & \text{if } \sqrt{k_r^2 + k_c^2} \leq 25 \\ 0 & \text{otherwise} \end{cases}$$

```
f = png_read('bluegill.png')
R,C = f.shape
F = fft2(f)

HL = numpy.zeros((R,C),dtype=complex)
for kr in range(-R//2,R//2+1):
    for kc in range(-C//2,C//2+1):
        if (kr**2+kc**2)**0.5 < 25:
            HL[kr,kc] = 1

lowpassed = ifft2(F*HL)
show_image(lowpassed)
```

2D Filtering

Find the 2D unit-sample response of this filter.

```
show_image(ifft2(HL))
```

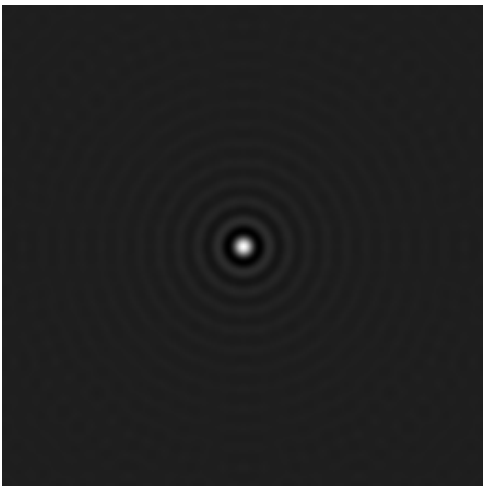


Step changes in $|H_L[k_r, k_c]| \rightarrow$ overshoot in $h_L[r, c]$: Gibb's phenomenon.

2D Filtering

Find the 2D unit-sample response of this filter.

```
show_image(ifft2(HL))
```



Step changes in $|H_L[k_r, k_c]| \rightarrow$ overshoot in $h_L[r, c]$: Gibb's phenomenon.

2D Filtering

Consider using the following filter, which is a circularly symmetric version of the Hann window.

$$H_{L2}[k_r, k_c] = \begin{cases} \frac{1}{2} + \frac{1}{2}\cos\left(\pi \times \frac{\sqrt{k_r^2+k_c^2}}{50}\right) & \text{if } \sqrt{k_r^2+k_c^2} \leq 50 \\ 0 & \text{otherwise} \end{cases}$$

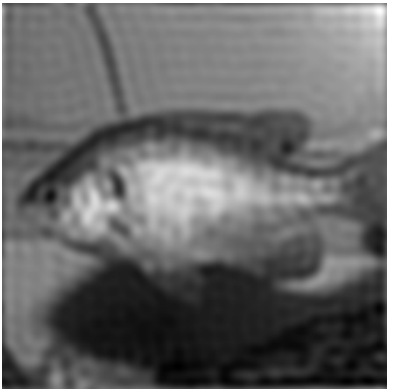
```
HL2 = numpy.zeros((R,C),dtype=complex)
for kr in range(-R//2,R//2+1):
    for kc in range(-C//2,C//2+1):
        d = (kr**2+kc**2)**0.5
        if d<=50:
            HL2[kr,kc] = 0.5+0.5*cos(pi*d/50)

hanned = ifft2(F*HL2)
show_image(hanned)
```

2D Filtering

Ripples are gone.

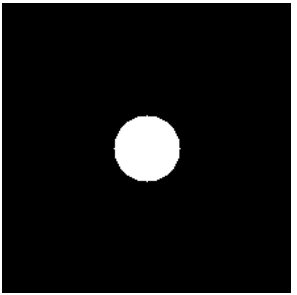
2D Filtering

Ripples are gone.

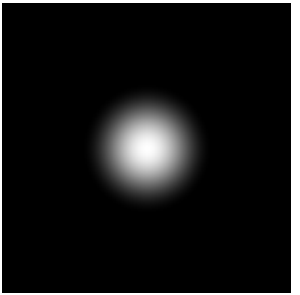


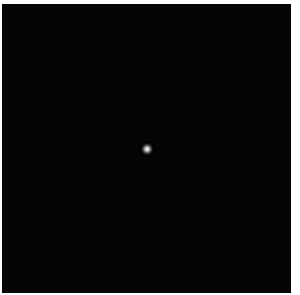
Comparing Filters

Filter 1



Filter 2





High-Pass Filtering

Use the same approaches to implement a high-pass filter.

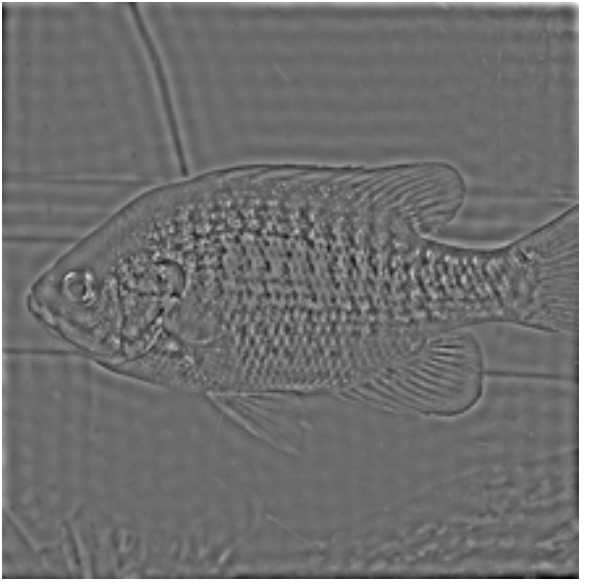
$$H_H[k_r, k_c] = 1 - H_L[k_r, k_c] = \begin{cases} 1 & \text{if } \sqrt{k_r^2 + k_c^2} > 25 \\ 0 & \text{otherwise} \end{cases}$$

In the spatial domain, then, we have:

$$h_H[r, c] = RC\delta[r, c] - h_L[r, c]$$

High-Pass Filtering

Not surprisingly, results show the same rippling effect seen in LPF.

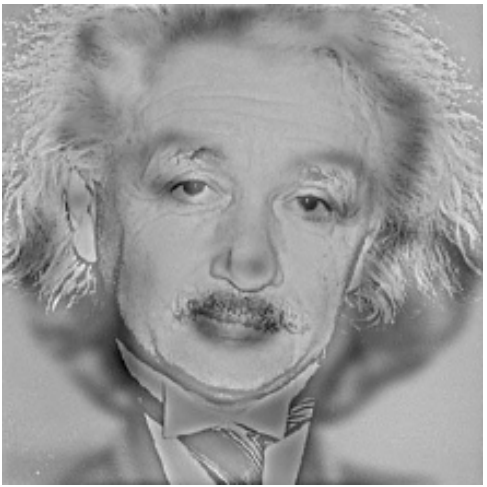


High-Pass Filtering

We can reduce the ringing artifacts by using $1 - H_{L2}[k_r, k_c]$ instead.

Who Is This?

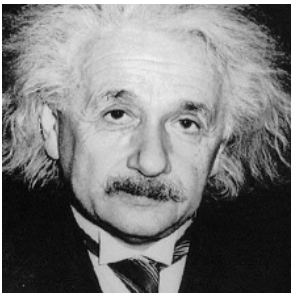
Look at this image with your eyes about a foot away from the screen. Then look again from a distance of six feet.



from Prof. Antonio Torralba

An Interesting Optical Illusion

A *hybrid image* is created by combining the low frequencies from one image (left) with the high frequencies of another (right).

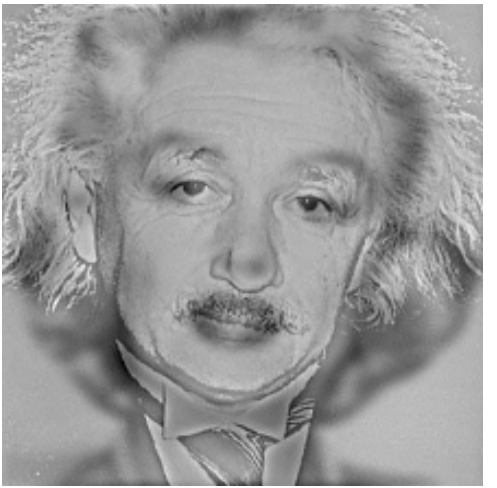


Hybrid Image

```
ein = fft2(png_read('lec11a_code/einstein.png'))
mar = fft2(png_read('lec11a_code/marilyn.png'))

ein = (1-LPF2) * ein
mar = (LPF2) * mar

show_image(ifft2(ein + mar))
```

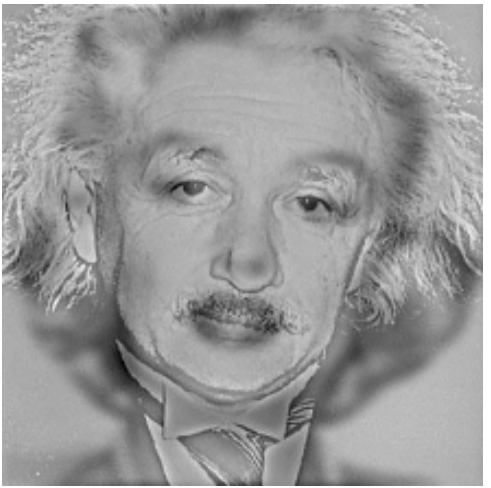
Hybrid Image

Why does the image that we see depend on our distance to the image?



Hybrid Image

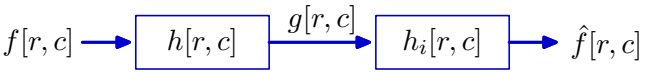
Why does the image that we see depend on our distance to the image?



Our eyes report a band-pass filtered version of the world to the brain.
Increasing distance to the image shifts the content to higher frequencies.

Filtering and Inverse Filtering

An important area of research in image processing is in **inverse filtering**, (also called deconvolution). The idea is to undo the effect of prior filtering.



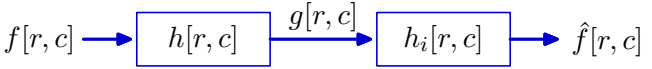
Example: enhancing images from Hubble Space Telescope.

- $f[r,c]$ represents the unknown image of a distant galaxy and
- $h[r,c]$ represents distortions in the optics of the telescope.

Goal: design an inverse filter $h_i[r,c]$ so that $\hat{f}[r,c]$ approximates $f[r,c]$.

Inverse Filtering

One simple approach is to filter by the inverse of $H[k_r, k_c]$.



In the frequency domain:

$$\hat{F}[k_r, k_c] = H_i[k_r, k_c] \times G[k_r, k_c] = H_i[k_r, k_c] \times (H[k_r, k_c] \times F[k_r, k_c])$$

If $H_i[k_r, k_c] \times H[k_r, k_c] = 1$ then $\hat{F}[k_r, k_c] = F[k_r, k_c]$!

Letting $H_i[k_r, k_c] = \dfrac{1}{H[k_r, k_c]}$ is called **inverse filtering**.

Quite remarkable that you can design a system to undo the effect of a prior system. Think about how you might do "inverse convolution"!

But it's simple (?) in the frequency domain.

Example: Motion Blur

Camera images are blurred by motion of the target.

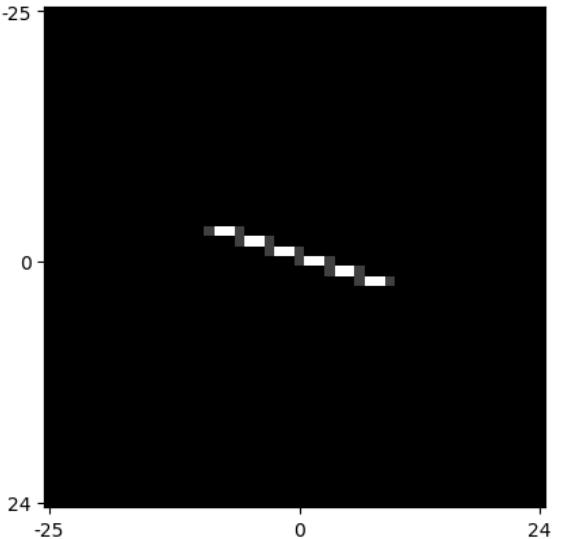
The resulting **motion blur** can be modelled as the convolution.

Modelling Motion Blur

Assume that streaks in this image resulted from the blurring. There is an isolated streak near the point r=120, c=250 (approximate 19x6 pixels).

Inverse Filtering

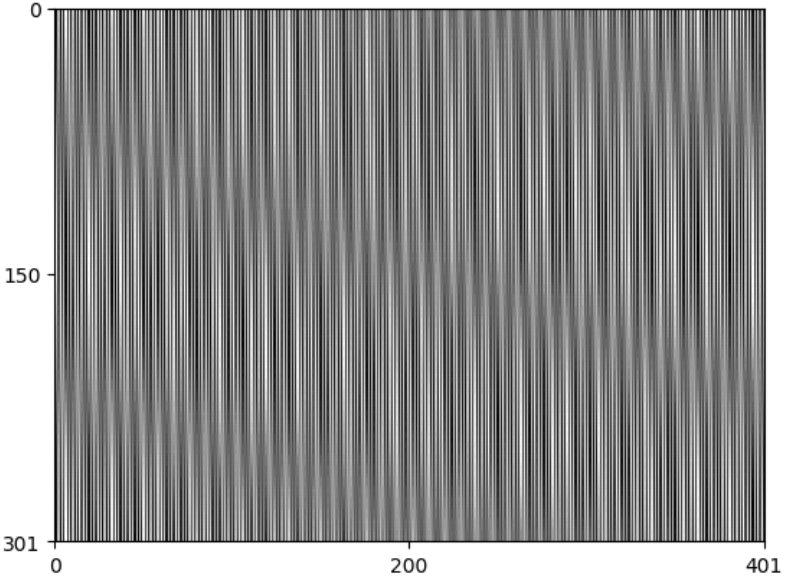
Make an image h to represent the presumed blurring function.



Let H represent the DFT of h, and filter the blurred image with $\frac{1}{H}$.

Inverse Filtering

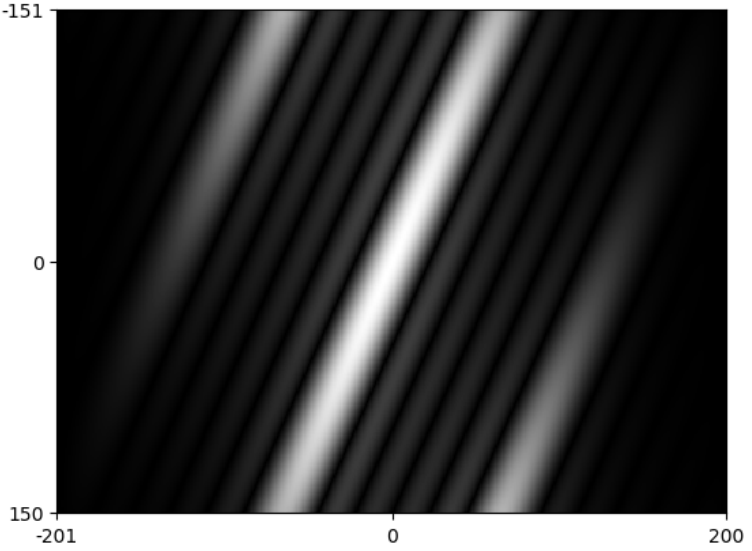
Here is the resulting inverse filtered image – not at all what we want.



What went wrong?

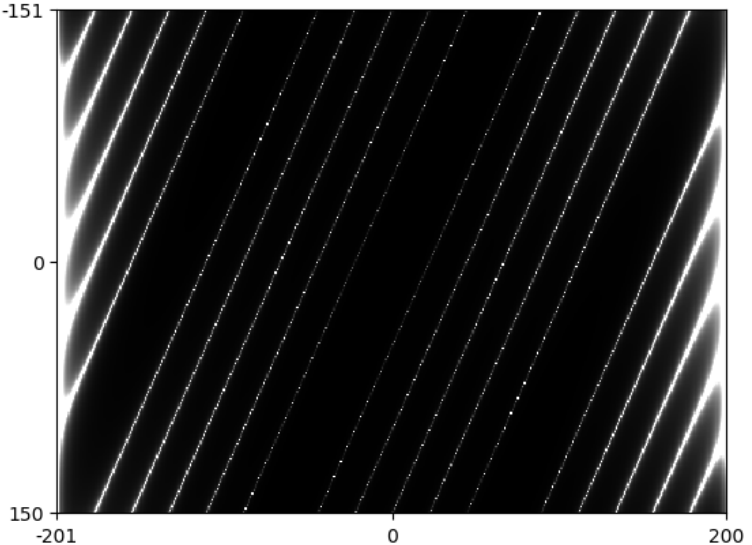
Inverse Filtering

This image shows the magnitude of H (DFT of blur function).



Inverse Filtering

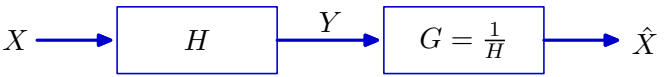
This image shows the magnitude of $\frac{1}{H}$.



What causes the bright spots? Why are they a problem?

Deblurring

The bright spots in $\frac{1}{H}$ come from points in H with values near zero.



Such bright spots dominate the result. Try limiting their magnitudes.

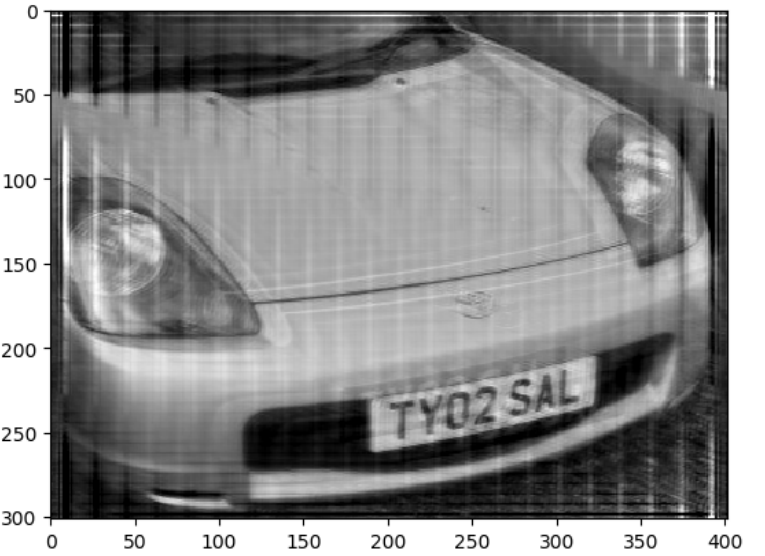
Method 1:

Start with $G = \dfrac{1}{H}$, but limit the magnitude of every point in G to 4:

```
for kr in range(R):
    for kc in range(C):
        G[kr,kc] = 1/H[kr,kc]
        if abs(G[kr,kc])>4:
            G[kr,kc] *= 4/abs(G[kr,kc])
```

Deblurring

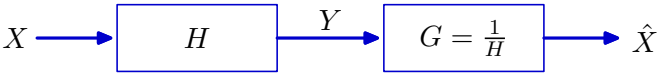
This deblurring filter works better: easy to read license number.



But there are many artifacts.

Deblurring

The form of the previous deblurring function is a bit arbitrary.



Method 2:

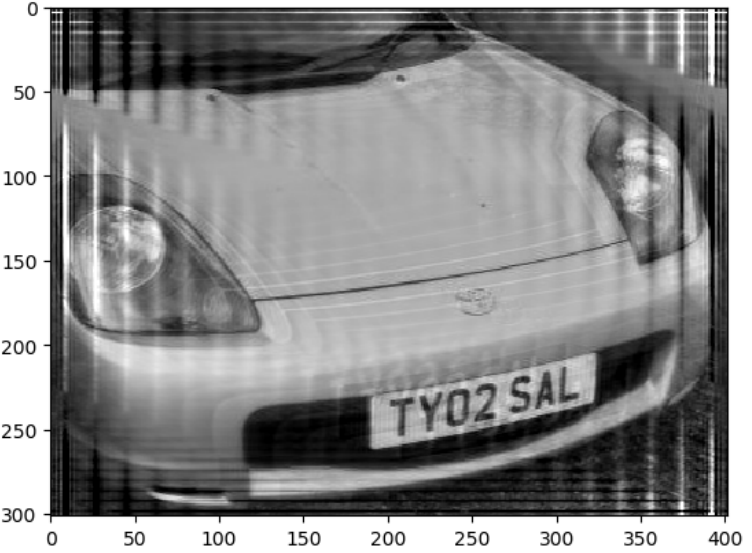
Here is a frequently used alternative (a "Weiner filter"):

$$G = \frac{1}{H}\frac{|H|^2}{|H|^2 + C}$$

where $C = 0.004$ (chosen by trial and error).

Deblurring

Alternative deblurring function.



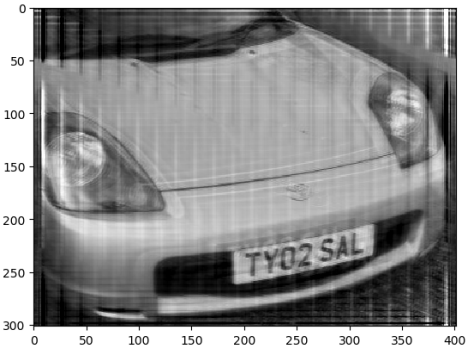
But there are still artifacts.

Edge Effects

Much of the ringing results from circular convolution. Window edges in original image to reduce step change due to periodic extension.

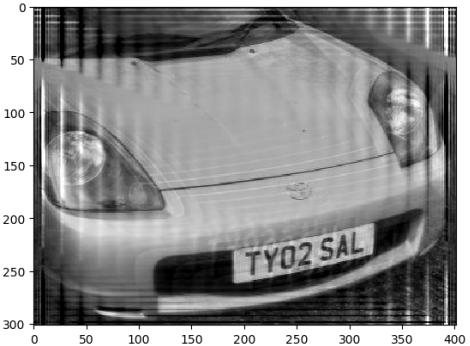
Comparison

Method 1 with and without windowing.



Comparison

Method 2 with and without windowing.



Conclusions

In general, inverse filtering worked well. It allowed a clear view of the license plate which was otherwise not legible.

Problems with inverse filtering. Inverting $H[k_r, k_c]$ doesn't work well if $H[k_r, k_c]$ is near zero. Fortunately, there were only a few such points. Arbitrarily limiting the values of such points results in useful deblurring.

Problems with circular convolution. Circular convolution introduces enormous artifacts if the left and right (or top and bottom) edges differ in brightness. These artifacts can be reduced by windowing.

Remaining problems. The resulting images still suffer from ringing – presumeably because of sharp discontinuities in the frequency representation of blurring.