DECONVOLVULATOR

Deconvolvulator combines some common sharpening and deconvolution techniques which are useful in astronomical image processing. With high signal to noise images (typical in planetary/lunar/solar) imaging, inverse filtering works well when an appropriate point spread function is specified. The use of Laplacian sharpening (called wavelets in Registax) also works well in images with high signal to noise ratios (no point spread function needs to be guessed when using this method).

For deep-sky images with lower signal to noise ratios, slower, iterative deconvolution algorithms such as Richardson-Lucy or Landweber work better, as their algorithms control noise better, especially with regularisation. However, in some images of bright deep sky objects (i.e. high signal to noise ratios), inverse filtering can also work well.

Choice of PSF

Most planetary/lunar/solar image stacks seem to deconvolve best when a Lorentz point spread function is specified. A Moffat function fairly close to Lorentz can also be effective, i.e. where β is close to one, rather than exactly one in a Lorentz function. Tapering the point spread function wings prematurely to zero at 5-15 times the full width at half maximum (FWHM) of the point spread function also generally helps when β is close to one. For stretched deep sky images, a Moffat function with a β parameter of around 2 often works well.

QUICK-START Notes (example images processed: https://50000quaoar.github.io/Deconvolvulator)

1) **Click** the green REPAIR button (top left). This performs a deconvolution of the test image Jup_CloudyNights_Bird.tif, with default settings:
DECONVOLUTION REPAIR METHOD = Wiener, and NSR = 0.001, i.e. Noise to signal ratio (NSR) = Reciprocal of the signal to noise ratio of 1000
with POINT SPREAD FUNCTION = Lorentz/Moffat, with β = 1.0, FWHM = 4.00 pixels, and the PSF tails reduced (tapered to zero) at 10.0 times the FWHM.

2) Move the **slider** in the green area (Down for more detail, but more noise, up for smoother, but less detail). Uncheck **"Auto recalculate"** below the REPAIR button if you don't want all changes you make to trigger a new deconvolution. If you want the REPAIR to happen faster, try **right-clicking** the image -> **"Select area to process"**, click and drag.

3) **Change** some options. Change the FWHM value of 4.00 in the blue area. Change the "Reduce PSF tails at FWHM times" to 15.0 to increase contrast, or change β from 1.00 to alter the PSF shape.

4) **Zoom** the image using mouse wheel when the cursor is over the image. Left-click to display the originally loaded image. Left-click and drag to move.

5) **Store** the repaired deconvolved image by clicking the "STORE repaired" button below the green "REPAIR" button.

6) **Click** the "SHARPENING LAYERS" radio button (in Magenta, mid-left). This performs a Laplacian sharpening using default settings, i.e. at the 2 pixel layer (already checked).

7) Move the **slider** to the right of the 2 pixel layer checkbox to change the strength of this layer (changes contrast).

8) Immediately below this slider, change the value of **Smooth** to increase or decrease the noise and details.

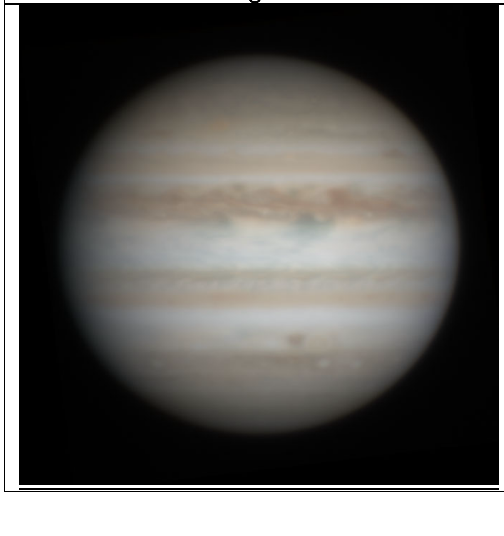
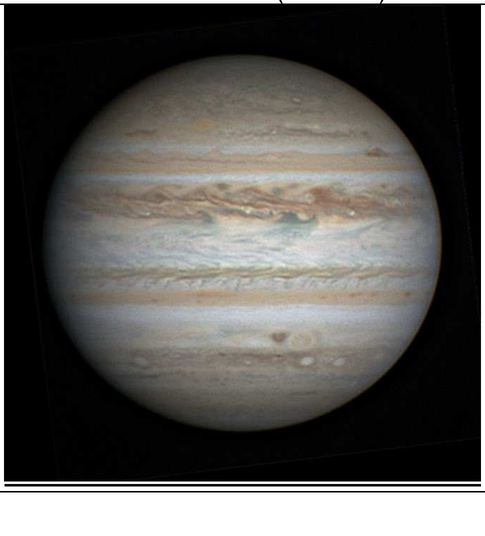
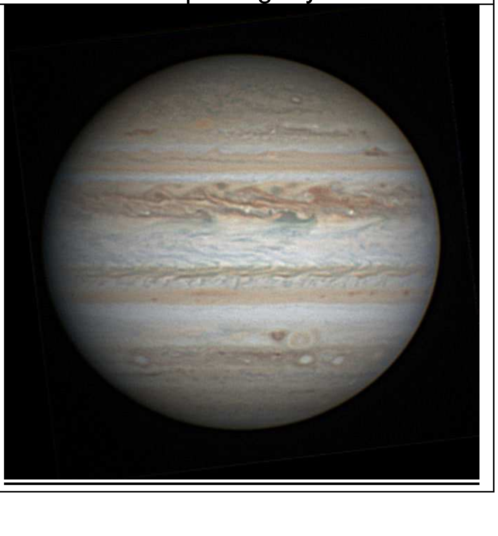
9) **Store** the sharpened image by clicking the "STORE repaired" button again.

10) Clicking on the image will now toggle between the latest stored image (stored at step 9) and the Wiener repaired image (stored at step 5). You can change the drop-down list at the bottom right to show "Original" instead of " Wiener ...".

11) **Save** the current repaired image displayed using the green "SAVE" button bottom left. Image is saved in the same format as the original image, and saved in the same directory as the input image.

12) **Choose** another image by clicking the small grey button "…" at the top left. Remember, **right-click -> "Select area to process"** is very helpful when previewing iterative deconvolution methods (i.e. "Total variation", or "Richardson-Lucy")

Image credit: Bird, https://www.cloudynights.com/topic/244642-raw-jupiter-data-for-download/

Original	Deconvolved (Wiener)	Sharpening layers
		

DECONVOLUTION THEORY

A) SINGLE STEP TECHNIQUES OF DECONVOLUTION (Inverse filtering)

If p = the point spread function, i = the blurred image, and o = unblurred image, then i = o ** p (** means convolve).
Performing a Fourier Transform on p, i and o (designated as P, I and O) means that I = O x P (simple multiplication)
To reverse the blur then **R = I ÷ P**, where R is the Fourier transform of the repaired image (an estimate of O), i.e. undo the multiplication.
r (the repaired image) is then the inverse Fourier Transform of R
Dividing by the (complex number) P is that same as multiplying by the complex conjugate of P and dividing by Abs(P) squared. Below, the complex conjugate of P is written as **Pr - Pi** (the real and imaginary parts).
Inverse filtering adds an extra term to the denominator [Abs(P) squared]. This avoids noise amplification when Abs(P) is small.
A good starting point with more information on theory is here: https://www.robots.ox.ac.uk/~az/lectures/ia/lect3.pdf

Wiener filter (one-step) deconvolution:

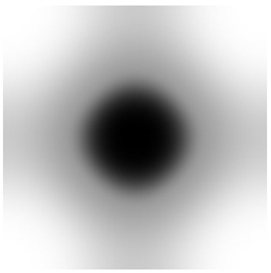
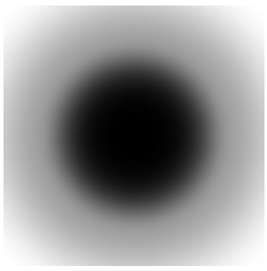
NSR = Noise to signal ratio [Average for image]
W = Wiener filter = (Pr - Pi) / (PAbs x PAbs + NSR)
The Fourier transform of the repaired image, R, then equals I x W
W can equivalently be written as W = (1 / P) x (PAbs x PAbs) / (PAbs x PAbs + SNR)

Regularised inverse filter [RIF] (one-step) deconvolution:

Same as Wiener filter, but NSR is multiplied by an extra factor to impose smoothness on the repaired image:
RIF = Regularised inverse filter = (Pr - Pi) / (PAbs x PAbs + NSR x LAbs x LAbs)
where L is a matrix that corresponds to the discretisation of a differential operator.
Again, the Fourier transform of the repaired image, R, then equals I x W
And r, the repaired image, is then the real component of the inverse Fourier transform of R.

Tikhonov filter (one-step) deconvolution (Also called Constrained least squares filtering):

Using same letter symbols as above plus:
Y = Regularisation parameter (similar numeric value to NSR in Wiener)
L = Fourier transform of 3x3 Laplacian matrix (0, -1, 0), (-1, 4, -1), (0, -1, 0), LAbs is absolute value of L
T = Tikhonov filter = (Pr - Pi) / (PAbs x PAbs + Y x LAbs x LAbs)
Again, the Fourier transform of the repaired image, R, then equals I x T
And r, the repaired image, is then the real component of the inverse Fourier transform of R.

LAbs x LAbs is shown graphically here: first for Tikhonov, then for the regularised inverse filter. The central area of the image is low frequencies and the edges are high. For low frequencies, the constant Y or NSR is multiplied by less than one, and zero in the centre (black areas). For mid to high frequencies, the constant Y or NSR is multiplied from one to about 400 at the corners (grey to white areas). This means that high frequency components (i.e. noise) are attenuated.

B) ITERATIVE TECHNIQUES OF DECONVOLUTION

Richardson-Lucy (iterative) deconvolution:

p = The point spread function, p1 = The point spread function in reverse order (the same for symmetrical point spread functions)
r(n) = Repaired image iteration n
i = Blurred input
r(n+1) = r(n) x (p1 ** (i / (r(n) ** p))), where ** means perform a convolution.

Landweber (iterative) deconvolution:

p = The point spread function, p1 = The point spread function in reverse order (the same for symmetrical point spread functions)
r(n) = Repaired image iteration n
i = Blurred input (Range 0.0 to 1.0)
r(n+1) = r(n) + tau x (p1 ** [i - r(n) ** p]) (a more robust version of the Van Cittert algorithm).

Optionally, a version with total variation regularisation has:
Div = Divergence of repaired image iteration n, i.e. gradient of normalised gradient
tau (constant, equal to 1.863), Regularisation parameter: lambda (constant, equal to 0.00001)
r(n+1) = r(n) + tau x (p1 ** [i - r(n) ** p]) - tau x lambda x Div, where ** means perform a convolution.
The **tau x lambda x Div** term is a denoising correction which reduces the total variation.

<u>(LAPLACIAN) SHARPENING LAYERS</u> (Also called Wavelets sharpening)

Inputs in user interface:
a) Contrast, c, 0.01 to 100 [Strength of sharpening effect]
b) Smooth, s, 0 to 5 [Reduce noise]
c) Layer scale x, l, 0.1 to 10
d) Kernel values (Selection of 3x3 and 5x5 kernels)
e) Layers in use (6 layers in scales of 0.5 pixels, 1 pixel, 2 pixels, 4 pixels, 8 pixels and 16 pixels)

Stage 1: Creating the 6 layers
For each of the 6 layers:
1) Resize the input image (enlarge by factor 2 for 0.5 pixel layer, same size for 1 pixel layer, times 0.5 for 2 pixel layer, times 0.25 for 4 pixel, etc.)
2) Convolve this result with the sharpening kernel (choose the kernel from the drop-down list in d))
3) Convolve this result with a Gaussian kernel, with sigma equal to input b) at the 1 pixel layer, or twice input b) at the 0.5 pixel layer, half of input b) at the 2 pixel layer, etc.
4) Resize the result back to the size of the original input image (Lanczos4 resampling)
***N.B. Alternative implementation of algorithm**: instead of resizing the input image, the sharpening kernel can be doubled in size for the 2 pixel layer, quadrupled for the 4 pixel layer, etc, before convolving with the original input image. This means that the sharpening layers can be considered one single convolution on the input image (or Fourier transform).*

Input c) allows the layers to be scaled with an additional factor. Instead of layers at 0.5, 1, 2, 4, 8, 16 pixels, you can have layers instead at 0.5 x *1.2*, 1 x *1.2*, 2 x *1.2*, 4 x *1.2* etc. In this case input c) would be *1.2*.

Stage 2: Combining the layers
1) Layers which are in use are scaled by their individual contrast levels. This is input c) multiplied by the slider values for each layer.
2) All the layers are added together to the original input image. At any pixel some layers will add, and some will subtract from the original pixel value.

Example of Laplacian Sharpening
For example, in 1D, with a (-1, 2, -1) kernel, to create the 1 pixel scale layer (no smoothing):

x Co-ordinate	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
Input values	1	2	3	6	8	6	2	1	0
Convolve with kernel, result equals		0	-2	1	4	2	-3	0	
Adding convolved row to input row		2	1	7	12	8	-1	1	

In this case the input of 8 at co-ordinate 4 is boosted to 12, whereas the values at co-ordinate 2 and 6 are reduced. Hence the central peak in the input is increased and its width reduced. This is the basis for the technique of Laplacian Sharpening.

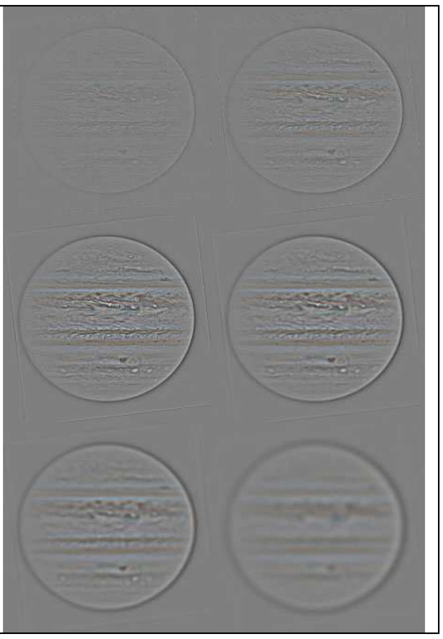
Gradient of gradient
Consider three consecutive pixels in 1D, with values a, b, and c.
The gradient between a and b is: (b-a)
The gradient between b and c is: (c-b)
The gradient of the gradient is therefore: (c-b) - (b-a) = a - 2b + c, which, if negated is equivalent to a (-1, 2, -1) kernel.
If you do the same gradient of gradient in 2D, you get a 3x3 kernel of (0, -1, 0) in 1st row; (-1, 4, -1) in 2nd row and (0, -1, 0) in the last row.
If you include diagonals as well, you get a kernel of (-1, -1, -1) (-1, 8, -1) (-1, -1, -1), which is a typical sharpening kernel

Visualising the layers

(Click on the small button "S") below the Reset button:
Six sharpening layers are displayed as shown here to the right:

Top left image is at the 0.5 pixel layer, top right is at the 1 pixel level, etc.

The mid grey background colour indicates that the sharpening layer has a value of zero, brighter means the original image pixel values will be increased, darker means the original image pixel values will be decreased.



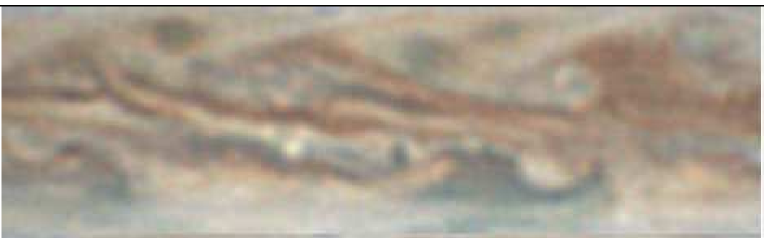
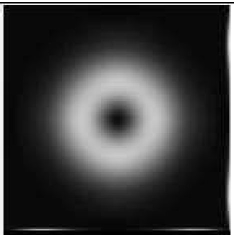
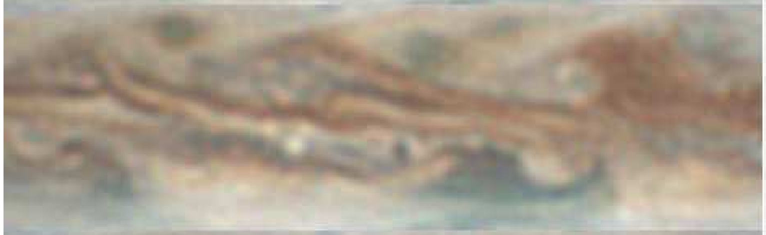
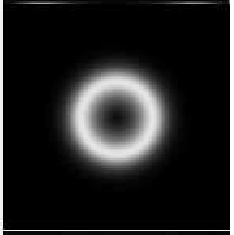
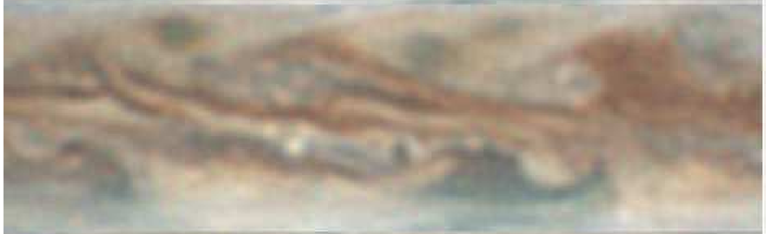
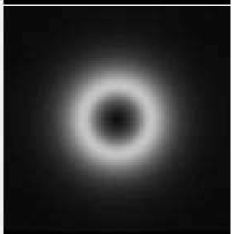
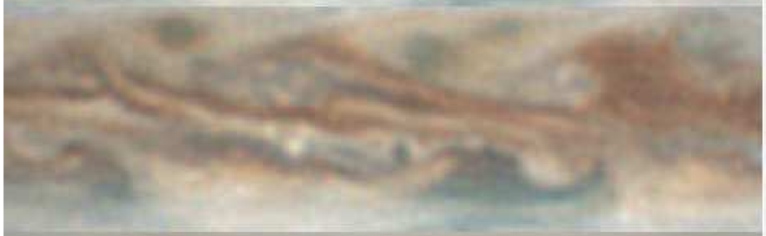
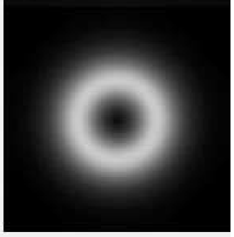
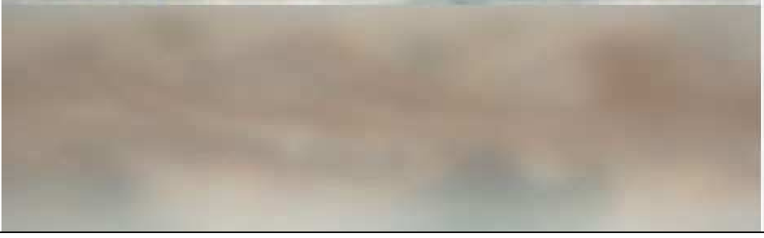
Comparison of inverse filtering, iterative deconvolution and sharpening layers

In typical astronomical images, deconvolution using a good PSF estimate combined with an inverse filter (e.g. the Wiener filter), iterative deconvolution, and Laplacian sharpening layers all produce very similar image restoration in practical applications. In fact, the Fourier transform of the inverse filter, and the Fourier transform of the deconvolution kernel applied in sharpening layers are often very similar (see below in right column). However, the Laplacian sharpening kernel of (-1, -1, -1) (-1, 8, -1) (-1, -1, -1) does not perform as well in restoring images which have a Gaussian blur - it works much better when the original image has been degraded by close to a Lorentz PSF (i.e. Moffat β = 1).

With sufficient iterations (300 non-accelerated Landweber shown below), iterative deconvolution techniques produce very similar results to those from inverse filtering. The overall Fourier transform that the iterative deconvolution is performing can be calculated by dividing the Fourier transform of the repaired output by the Fouier transform of the blurred input image. This calculated filter closely resembles the Fourier transform of inverse filters (which are computed from the PSF).

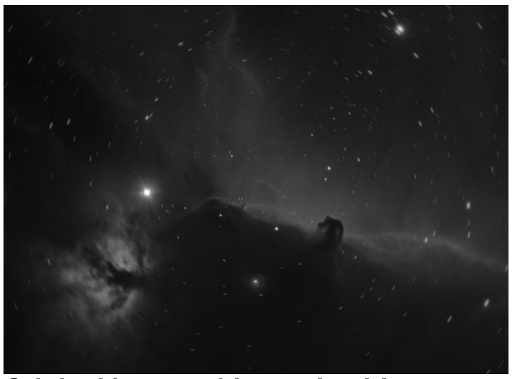
In some cases of very high signal to noise ratios (the extreme example is when sharp input images are convolved artificially by software and no noise is added), inverse filters can produce almost perfect restoration, whereas sharpening layers will be substantially less capable.

Perhaps more so with sharpening layers, a greater variety of repaired outputs can be produced by combining different layer weights. This can highlight features at some desired scale for more emphasis. In this case the output may not be an accurate mathematical deconvolution, but can nonetheless be useful. The same is true with an inverse filter. Using a FWHM for the PSF which is larger than it should be will give greater contrast of features, but still performs a fairly accurate overall deconvolution.

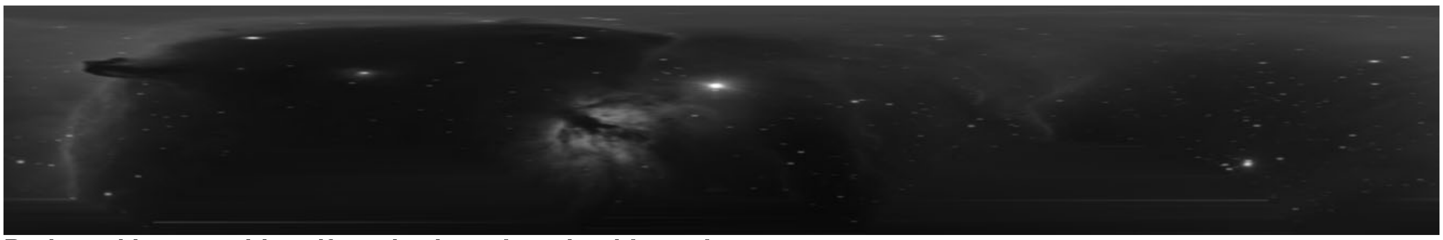
Method	Repaired	Fourier transform of filter
Sharpening layers -1,-1,-1; -1,8,-1; -1,-1,-1 1 Px layer only Contrast 4.52 Smooth 1.3		
Regularised inverse filter NSR 0.00063 PSF Moffat: Beta = 1.1 FWHM 4.0 Reduce PSF tails at 20 x FWHM		
Landweber 300 iterations PSF Moffat: Beta = 1.1 FWHM 4.0 Reduce PSF tails at 20 x FWHM		
Wiener filter NSR 0.001 PSF Moffat: Beta = 1.1 FWHM 4.0 Reduce PSF tails at 20 x FWHM		
Original		

Motion blur and Field de-rotation

Deconvolution also has applications in astronomical images affected by motion blur: either through bad or absent tracking, or field rotation (such as in Alt-Az mounts with longer exposure times). The PSF for images affected by motion blur is a simple line. For field rotation, the PSF would need to vary across the image, which is a more complex problem to solve. In this case deconvolution can be achieved by first projecting the image into an alternative co-ordinate system. The co-ordinate system is such that the original radial distance from the centre of rotation becomes the new vertical co-ordinate in the projection (where top means the original centre). Stars near the centre of rotation in the original image then get stretched out horizontally to the same extent as stars at the corners. A standard motion blur deconvolution can then be applied, and the projection subsequently reversed. Note that gaps in the projection are filled in by interpolation from the last/next edge on the horizontal row to avoid edge artefacts when applying deconvolution.


Original image with rotation blur
Centre of rotation is centre of image, with 1.45 degrees of field rotation

The pixel at the centre of the image above becomes the top row of the projected image below. Pixels around a circle 10 pixels from the centre of the image above become the tenth row in the image below, etc.


Projected image, with uniform horizonal motion blur only

Now, this image has a horizontal blur of the same length everywhere, and can be deconvolved using a standard horizontal motion blur PSF. The end result can be viewed in the example images on:
https://50000quaoar.github.io/Deconvolvulator

SOURCE CODE

Deconvolvulator is written in C# (with Microsoft Visual Studio Community 2015). It targets the .Net Framework 4.0. The main dependency is on OpenCvSharp3-AnyCPU by shimat (and OpenCV 4.x wrapper), installed via Nuget. If you do want to compile the source code, you can open the .sln file and then use Nuget to download the package, which will be reported as being missing by Visual Studio. If that fails, you could try getting the Nuget OpenCV wrapper package working first in your own test project, then add the deconvolvulator project form file manually to the project.

In Deconvolvulator, OpenCV is used both for image processing and Fourier Transforms. Project references are needed to OpenCvSharp (OpenCvSharp.dll) and OpenCvSharpExtensions (OpenCvSharp.Extensions.dll). Deconvolvulator does not use mutli-threading, or any other substantial optimisations. It relies on the optimised code in OpenCV for efficient image manipulation and Fourier transforms.

In order to perform motion blur de-ringing, SExtractor.exe is used to extract stars (a .Net wrapper for it: SExtractorWrapper.dll, is also required). Neither of these filse are required for the main functions of Deconvolvulator.

The file, settings.txt, stores last used settings for next time the program is run.

Performance and memory use seem fine on a modest computer when doing a single filter deconvolution (e.g. Wiener) on small to medium image sizes. There is of course scope for further performance enhancement.

Other (free / open source) relevant software

Registax (https://www.astronomie.be/registax/) Wavelet sharpening (i.e. Laplacian sharpening in layers of different scale)
SmartDeblur (https://github.com/Y-Vladimir/SmartDeblur) - General purpose deconvolution
DStation (https://github.com/blackhaz/DStation) - Astronomical deconvolution
Aberrator (http://aberrator.astronomy.net/) - Useful for creating artificially blurred images to test deconvolution on
DeconvolutionLab2 (http://bigwww.epfl.ch/deconvolution/deconvolutionlab2) - Large selection of deconvolution algorithms

LICENCE

The C# code that I have written in Deconvolvulator is free, you can use it for any purpose.
OpenCv licence information is here: https://opencv.org/license/.
The OpenCvSharp wrapper has licence information here: https://github.com/shimat/opencvsharp/blob/master/LICENSE