Reconstruction of a Rotationally Blurred Image

2004-08-29
Johan Henriksson (mahogny@areta.org)

Problem Description

Images are taken from an orbiting satellite which is rapidly spinning on its axis. The result is that even during the shortest exposure time, the images are to some degree rotationally blurred.

The problem is to devise and implement an algorithm to undo the rotational blur.

Requirements

There are no requirements given. Hence I will pick some suitable ones out of air. The goals are in order of priority,

1. Quality of image
2. Speed of algorithm

Quality will be measured in whichever is easiest to work with, L_1 or L_2.

Rotational Blur

Let us first look at how the blur arise; light of some wavelength comes into the camera and will etch in the pattern. As the camera rotates, the light from a certain point will be spread out over a certain area. Because the process is not instantaneous, the area will not be just a point.

In this case we will consider the following blur process; for the sake of convenience, let us for now use polar coordinates.

Define s(r,ϕ) as the light (red, green or blue. We will consider each color a separate problem) injected into the camera at a certain point. This function is 2π-periodic.

The similar function t(r,ϕ) denotes camera output.

Now the blurring process will be

$$(1)\quad t(r,f)=\frac{1}{\int_0^d w}\int_0^d w(b)\,s(r,f+b)\,db$$

where w(b) is a weight function to account for non-linear recording of light. The most simple selection of w=1 which

means to just average over the area covered by the camera during exposure. We can make the process a bit easier to work with by instead considering

$$(2)\quad t(r,f)=\int_{-inf}^{inf} k(b)\,s(r,f+b)\,db$$

where the kernel k() is a PDF.

Algebraic Approach

In our problem, s() is sampled and should be considered a function in the generalized sense. On the other hand, t() is physically determined with everything that means. Our algorithm will/can not output a function but we will sample t() as well during inversion. Therefore the following discretized version of the problem can be considered:

$$(3) \quad t_{(r,f)} = \sum_{b=-inf}^{inf} k_b s_{(r,f+b)}$$

It makes physical sense to assume k() has compact support, so the summation contains a finite number of terms. If we further only deal with a finite number of sample points, we have something we can handle numerically.

Let us now make use of the periodicity of t and s. Fix $r=r_0$ and consider f only. Assume the period of the discrete index is F. Consider the equations

$$(4) \quad t_{(r_0,0)} = \sum_{b=-inf}^{inf} k_b s_{(r_0,0+b)}$$

...

$$(5) \quad t_{(r_0,F)} = \sum_{b=-inf}^{inf} k_b s_{(r_0,F+b)}$$

This gives us F equations and F unknowns. This seems solvable; define the matrix K for the kernel by

K: $[k_{ij}=K_{b+F}]$

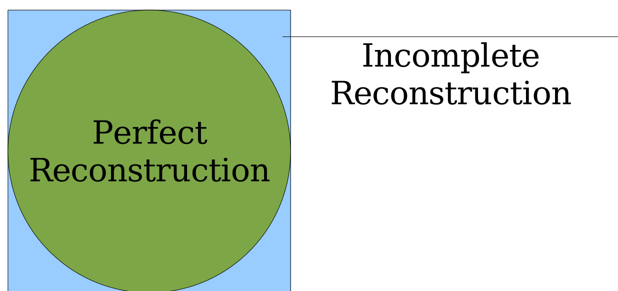
Because A is quadratic, if the det $A \neq 0$, a solution exists and it will also be unique. This is equivalent to the basis being independent. This will be the case iff the kernel functions are almost orthogonal which they will be unless the image is totally sabotaged.

If we assume the process we have defined to be correct, as we started with some fixed input values, we will have reversed the process. Because the solution is unique, we can conclude:

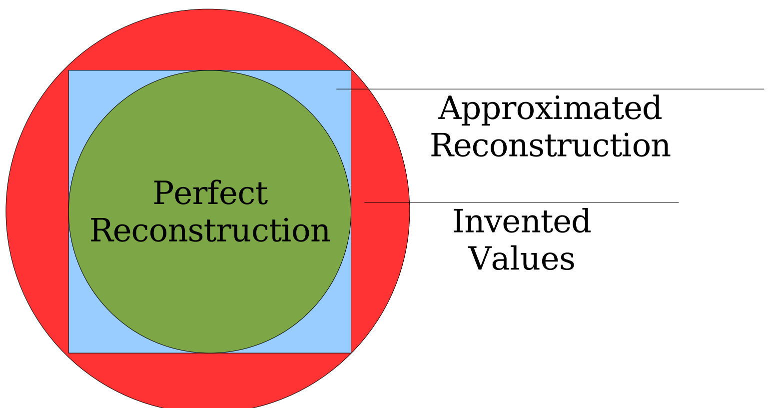
The reconstruction will always work with a physically sane process, and the reconstruction of a circular image will be perfect.

An Algorithm

Unluckily, we have assumed too much. The input isn't a circular image (next time you send up a satellite, think a bit ahead). We no longer have a complete equation system. Looking at the image, this is what we have:



We have a large for which our results apply but the equation system will not be sufficient to cover the corners. We solve it by inventing values:



The values can be chosen arbitrarily but to reduce the error we get we should try and choose something clever. The fastest method, which I believe give reasonable values, is to simply clamp (as in openGL clamp) the output t(). Other ways include extending the derivate or using FFT but all reasonable ways have to depend on the regularity of the image. If there is a lot of noise in s(), the SISO-rule will apply as always.

If we take the kernel to be H(x-a)-H(x-b), where H() is the step-function and a and b chosen depending on the rotational speed, one gets a pretty typical matrix for this kind of problem. It will be an upper band matrix. The faster the satellity rotates, the more sparse it will be. All s() depend on all t() which makes sense as we are considering a ring (which has no loose end). Thus it might be pretty badly conditioned. The more noise there is, the worse the situation will be. For an image of stars, we can expect the b-matrix to be very regular with spikes. That is the best case. However, we have to prepare for the worst and then we are best of using iterative sparse matrix methods. Matrix has these

built in and for the sake of making it easy to change parameters I will let Matlab deal with the problem. A commercial strength solution should use custom code instead, both to get speed and quality. The best iterative starting point will be t() which, while blurred, is still a pretty good approximation.

Radial algorithm, try #1

We make a first shot and just implement the algorithm head on.

Algorithm 1

ForAll radius r,

Choose a number of sampling points

Set up a kernel matrix based on the sampling points

Calculate unblurred points using inverse of matrix

Store radial points in unblurred image

This algorithm turns out to be incredibly slow. It also doesn't consider interpolation between the unblurred points so the result becomes a bit crappy.

Eulerian coordinates algorithm, try #2

Ok, back to reality. The results tell us that we can make a perfect reconstruction but the actual algorithm doesn't behave in practice. It turns out we have to be a bit more careful with the coordinate system we use. The new algorithm becomes as follows:

Algorithm 2

Set up an empty matrix C

forAll x,y in blurred image

Select number of sampling points

For each sampling point,

Add coefficients of blurring to C with respect to the sampling points

Compute unblurred image with inverted C

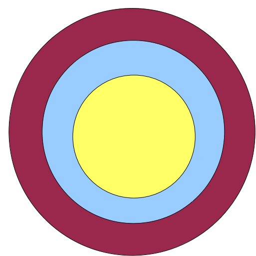
It turns out to be a quite well-working algorithm. Slow, but not as slow as before. We do not interpolate in the blurred space and keep the precision. The interpolation is instead done in the unblurred space. It is easy to set up linear interpolation between the unblurred coordinates and just throw in the weights in the C-matrix.

The linear interpolation is simple to describe; Assume we have a box with the value at the corners being (x_{ij}, $x_{i+1,j+1}$). The length of each side is 1. The point of interpolation has coordinates (xr,yr) relative to the upper-left corner. Then we can describe the interpolation as

$$x=(1-y_r)((1-x_r)x_{ij} + x_r x_{i,j+1}) + y_r((1-x_r)x_{i+1,j} + x_r x_{i+1,j+1})$$

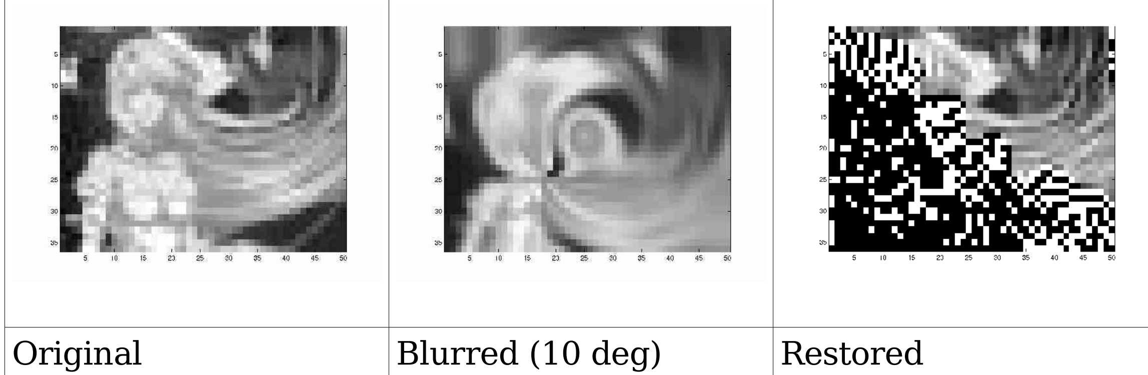
The weight are easily identified and put into the C-matrix.

Now, we notice a problem; earlier we had radial coordinates. The dependencies looked like this:



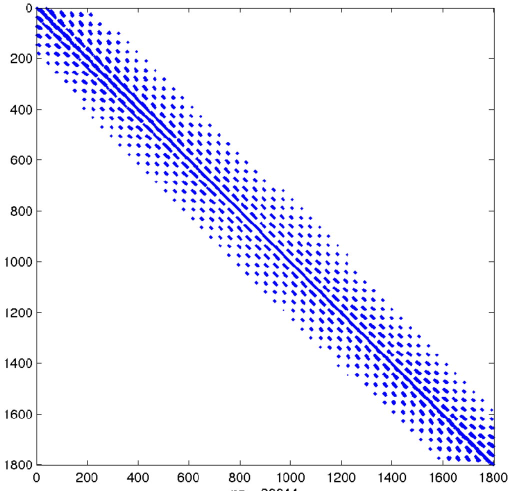
I.e. we could solve for one radius at a time. This is no longer the case; x_{ij} is no longer on a perfect circle and they will be interdependent between the circles. Hence we have two options; either we drop the dependency by only considering the nearest points in a circle. The result would look like crap. Instead we are stuck with solving a Huge matrix system. The real solution to the problem is to stick to radial coordinates from the beginning but that isn't really an option here.

Once implemented, here is the output of a very small image:



Original	Blurred (10 deg)	Restored

The restorted image is massacred. Not because the algorithm in itself is bad but because the generated matrix C is hard to inverse and Matlab dies. The matrix looks like this:

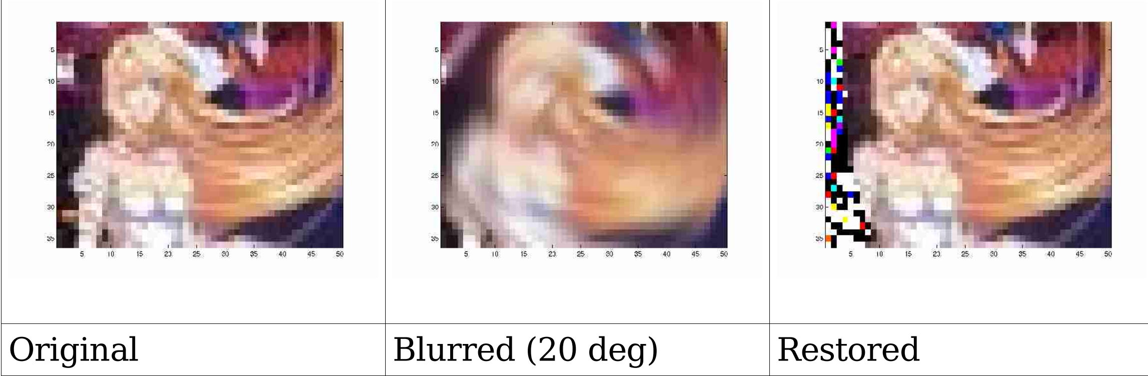


Meaning that it fullfills our earlier requirements about a “physically sane process”. Hence the problem is indeed Matlab; it's not good enough. Methods in sparse matrices can be used to solve this. Matlab also has another problem; for a decent sized image, the array required to store the matrix is too big as Matlab uses 32-bit integers for indexes.

(side comment: I later found a bug which made the blurring seem a bit better. But then the unblurring failed completely)

Beating Matlab, try #3

There isn't much of a choice here; we need a better matrix solver. I wrote up a quick one (really quick) which improved a ton:

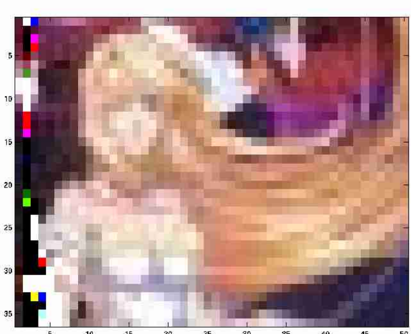


Original	Blurred (20 deg)	Restored

There is some weird singularity that my quick hack doesn't deal with but we are getting somewhere. This is as much as 20 degrees and the parts that are unblurred really are Good. In fact, it even works at 60 degrees but the singularity is starting to grow out into the image for higher numbers.

Adding another quick hack: we enforce explicitly as a boundary condition that the unblurred image has the same corners as the blurred

one. As noted before, we need some estimate of the color as we don't have enough information. This information doesn't come automatically as soon as we have hidden the unknowns. With this extra piece of machinery, we get this unblurred image:



It still doesn't get rid of all peaks but it's much better. We cannot restore this area (as the earlier proof hints) so we have to accept some kind of defeat. As we have eulerian coordinates, we can to some degree extend into the corners but that require a much more serious equation solver than what I have put together.

Next, we need to be able to handle bigger images. Adding some sick optimizations as well as building with a sparse matrix the memory limitation became less of a problem. This became incredibly slow, Matlab is too stupid to realize it should store it as a bandmatrix.

Further Work, would be try #4

For big images, our matrix grows too large. There is a fix to this; we never need to store it. We can easily calculate it on the fly. It would then also be possible to infer the boundary restrictions implicitly. The drawback is speed. There is a lot of trigonometry involved. It would not really be possible to precompute the dependence of each pixel because that is equal to storing our matrix in banded form with the current solution.

I feel this is getting out of topic for what this course really should be about; graphics - not sparse matrices. Hence I will not implement this.

Complexity Analysis

The memory used for the non-sparse version is $O(w^2h^2)$ which hints why the sparse storage is required. The computational complexity can be shown (for my iteration) to be approximately $O(w^2h^2a^2)$ in error where a is the efficient angle covered by the kernel. Luckily, the hidden constant for speed is rather small and could be much less if the code were optimized for BLAS or some other library.

Alternate Approach

Functional analysis can be used to make a more convincing proof about reconstructability than by first making a matrix. The details are much more involved hence my choice of method. It might also be possible to

come up with a transform in form of an analytic expression for unblurring. I did not succeed in this task. Such an approach would quickly run into problems however, as the real problem uses eulerian coordinates.

Summary

For the eulerian algorithm, the following holds: The middle circle of a rotationally blurred image can be perfectly restored, with up to 60 degrees tested (up to 180 degrees suspected). The algorithm can be used also for non-linear exposure but this paper only tries a constant exposure under a finite interval. The corners are lost but can partially recovered with some guessing. A strong solver for banded matrices is required. If a camera with a radial coordinate system (giving a round image) would be used, things would run much faster and with less memory.

References

Only prior knowledge used which I can't tell exactly where I picked it up. Mostly from the courses

- Linear Algebra
- Numerical Linear Algebra
- Large Sparse Matrices
- Functional Analysis
- Fourier Analysis

I can recommend taking them all.