IPOL IMAGE PROCESSING ON LINE

Published in Image Processing On Line on 2012–10–17.
Submitted on 2012–00–00, accepted on 2012–00–00.
ISSN 2105–1232 © 2012 IPOL & the authors CC–BY–NC–SA
This article is available online with supplementary materials,
software, datasets and online demo at
https://doi.org/10.5201/ipol.2012.t-fscs

The *Flutter Shutter* Camera Simulator

Yohann Tendero[1]

[1]CMLA, ENS Cachan, France (tendero@cmla.ens-cachan.fr)

Communicated by Jérôme Gilles *Demo edited by* Yohann Tendero

Abstract

The proposed method simulates an embedded *flutter shutter* camera implemented either analogically or numerically, and computes its performance. The goal of the *flutter shutter* is to make motion blur invertible, by a "fluttering" shutter that opens and closes on a well chosen sequence of time intervals. In the simulations the motion is assumed uniform, and the user can choose its velocity. Several types of *flutter shutter* codes are tested and evaluated: the original ones considered by the inventors, the classic motion blur, and finally several analog or numerical optimal codes proposed recently. In all cases the exact SNR of the deconvolved result is also computed.

Source Code

The C++ implementation of the *flutter shutter* camera simulator is available on the article web page[1], with source code and documentation.

Keywords: Fourier transform, image motion analysis, image restoration, image sensors, Poisson noise, image acquisition, computational photography, flutter shutter, motion-invariant photography, SNR

1 Introduction

Classic digital cameras are devices counting at each pixel sensor the number of photons emitted by the observed scene during an interval of time Δt called exposure time. Due to the nature of photon emission the counted number of photons is a Poisson random variable. Its mean would be the ideal pixel value. The difference between this ideal mean value and the actual value counted by the sensor is called (shot) noise. The ratio of the mean of the photon count over its standard-deviation is called signal to noise ratio (SNR). At (very) low SNR the noise is so strong compared to the underlying signal that it is almost impossible to distinguish the scene being observed from the noise. Therefore, photography has been striving to achieve the highest possible SNR. In passive imaging systems,

[1]https://doi.org/10.5201/ipol.2012.t-fscs

where there is no control over the scene lighting, the only way to increase the SNR is to accumulate more photons by increasing the exposure time Δt.

If the photographed scene moves during the exposition process, or if the scene is still and the camera moves, the resulting images are degraded by motion blur. The difficulty of motion blur is illustrated by its simplest example, the one-dimensional uniform motion blur. Indeed, if the relative velocity between the camera and the scene remains constant, then motion blur is nothing but a convolution of the image with a one-dimensional window function. This motion blur is not invertible if the blur exceeds two pixels. Consequently, when photographing with a moving camera, it is not possible to accumulate an arbitrary number of photons and to reach any SNR, contrarily to standard steady photography.

Recently, a revolutionary imaging method to circumvent the motion blur problem has been invented by Agrawal, Raskar et al. [1, 9, 10, 11]. These authors propose to use a binary shutter sequence interrupting the flux of incoming photons on time sub-intervals of the exposure time interval. Indeed for well-chosen such binary shutter sequences, arbitrarily severe motion blur can be made invertible as illustrated by figure 1. Therefore this method permits to increase at will the exposure time, and to accumulate as many photons as desired. This technological novelty has generated much interest [2, 5, 7, 8, 10]. According to the analysis proposed by Tendero et al. [12], there are two different ways to interrupt the flux of photons and therefore to achieve the stroboscopic effect of the *flutter shutter*. It can be done with an *analog flutter shutter*, by stopping part of the photons before they hit the sensor (temporal sunglasses). The other solution is to use a *numerical flutter shutter*. In this second setup, the camera takes a burst of L images using a small exposure time. The k-th image is then multiplied by a carefully chosen number α_k and added to the previous one. For both *flutter shutter* setups only one image has to be stored or transmitted. Thus the *flutter shutter* seems to be perfectly fit to Earth observation satellites or to any application where the transmission bandwidth and the computational capabilities are severely limited.

For the *analog flutter shutter*, any positive function bounded by 1 is technically implementable. In the *numerical flutter shutter*, the *flutter shutter gain function* can also have negative values. It is proven by Tendero et al. [13] that the Levin et al. *motion-invariant photography* (MIP) [7] is an example of *numerical flutter shutter camera* device.

This paper presents the detailed algorithm (sections 2 and 3) and an on line demo (section 4) simulating both kinds of *flutter shutter*. It simulates the random (Poisson) image, the blurry acquired image, the deconvolved image, and gives the final SNR. Some examples will be commented in section 5, and others can be simulated using the on line demo. The algorithm simulates for example the following setup: a camera on a satellite or a plane flies over a landscape at a high altitude and at fixed velocity v. Alternatively, the camera is assumed to be steady and the observed object moves in uniform translation. In both setups, the resulting motion blur is a convolution by the characteristic function of an interval.

Using a general formalization and simulation of the *flutter shutter*, we shall compare numerically many apparatus: standard cameras, *analog*, *numerical flutter shutter* using various flutter strategies, varying velocities, SNRs, etc. A special attention has been given to the noise simulation. Indeed, it is here crucial to give a good estimation of the SNR after the deconvolution process estimating the underlying landscape.

The theory underlying the simulations and the computation of the SNR is developed by Tendero et al. [13].

Figure 1: Left: simulated observed (blurry and noisy) image, notice the stroboscopic effect of the *flutter shutter* apparatus. The blur interval length is 52 pixels. Right: reconstructed image (RMSE = 2.55). Such reconstruction is not possible without a *flutter shutter* camera.

2 Algorithm

There exist two different types of *flutter shutter* depending on whether the gain modification takes place before (*analog flutter shutter*) or after the photons hit the sensor (*numerical flutter shutter*). For an *analog flutter shutter* the gain is defined as the proportion of incoming photons that are allowed to travel to the pixel sensor. Thus only positive (actually in $[0, 1]$) gains are feasible. The *numerical flutter shutter* camera, instead, takes a burst of L images using an exposure time of Δt. Then the k-th image is multiplied by a gain $\alpha \in \mathbb{R}$ and added to the previous one to obtain the observed image. Consequently, only one image is stored and transmitted. This implies that the observed value $o(n)$ at pixel n is always a Poisson random variable for the *analog flutter shutter*, but not for the *numerical flutter shutter*.

In the following the sequence of gains used on the camera is called "*flutter shutter* code" and is defined as the vector $(\alpha_k)_{k=0,\dots,L-1}$. Given a code the *flutter shutter gain function* is defined by $\alpha(t) = \alpha_k$ for $t \in [k\Delta t, (k+1)\Delta t[$, $\alpha(t) = 0$ otherwise.

2.1 Short Description

The algorithms will be first described in a continuous, and then in a discrete (see section 3) framework. Roughly the algorithm consists of four steps:

1. Simulate the ideal noiseless observed image.
2. Simulate Poisson (photonic) noise (see section 2.2) to obtain the observed image.
3. Estimate the landscape by deconvolution.
4. Compute the error, namely the root mean squared error (RMSE) and a contrast invariant RMSE (see section 2.4).

The implementation in C++ is detailed in the implementation section 3. For color images each component is processed independently, and the RMSE is averaged over all components.

2.1.1 The *Analog Flutter Shutter*

The simulation data are an image $u(x)$, a code (fluttering sequence or gain) $\alpha(t) \geq 0$, a velocity v, and an SNR level (and using a normalized $\Delta t = 1$). Let $\hat{u}(\xi) = \int_{-\infty}^{\infty} u(x)e^{-ix\xi}dx$ be the Fourier transform of u. We assume in the following that u is band limited: $\hat{u}(\xi) = 0$, $\forall |\xi| > \pi$. The *flutter shutter gain function*

$$\alpha(t) = \sum_{k=0}^{L-1} \alpha_k \mathbb{1}_{[k\Delta t,(k+1)\Delta t[}(t)$$

is designed so that

$$\hat{\alpha}(\xi v) = \Delta t \frac{2sin(\frac{\xi v \Delta t}{2})}{\xi v \Delta t} \sum_{k=0}^{L-1} \alpha_k e^{i\xi v \Delta t \frac{2k+1}{2}} \neq 0 \ \forall \xi \in [-\pi, \pi],$$

therefore it is invertible on the support of $\hat{u}(\xi)$. A detailed numerical implementation is given in section 3.

Step 1. Compute the ideal noiseless observed pixel value.
At pixel x, the ideal noiseless observed pixel value is $\left(\frac{1}{v}u * \alpha(\frac{\cdot}{v})\right)(x)$ computed using inverse Fourier transform $\mathcal{F}^{-1}\left(\hat{u}(\xi)\hat{\alpha}(v\xi)\right)(x)$.

Step 2. Simulate the observed pixel value.
The observed pixel value at pixel x is a Poisson random variable with intensity $\lambda(x) = \left(\frac{1}{v}u * \alpha(\frac{\cdot}{v})\right)(x)$ computed using $\lambda(x) = \mathcal{F}^{-1}\left(\hat{u}(\xi)\hat{\alpha}(v\xi)\right)(x)$ by definition of the *analog flutter shutter*. Let $o(x)$ be a realization of $Poisson(\lambda(x))$ (see section 2.2 below).

Step 3. Estimate the landscape pixel value by deconvolution.
The estimated landscape from the observed pixel value is then obtained by a deconvolution filter, $\mathfrak{u}_{est}(\xi) = o * \gamma$, where γ is the inverse filter satisfying $\left(u * \frac{1}{v}\alpha(\frac{\cdot}{v})\right) * \gamma = u$, computed by $\hat{u}_{est}(\xi) = \hat{o}(\xi)\hat{\gamma}(\xi) = \frac{\hat{o}(\xi)}{\hat{\alpha}(v\xi)}$.

Step 4. Compute error using RMSE and contrast invariant RMSE$^{\mathbf{CI}}$.
Straightforward with section 2.4.

2.1.2 The *Numerical Flutter Shutter*

For the *numerical flutter shutter*, the only difference is that there is no positivity constraint on $\alpha(t)$. But the simulation algorithm is slightly different:

$$\alpha(t) = \sum_{k=0}^{L-1} \alpha_k \mathbb{1}_{[k\Delta t,(k+1)\Delta t[}(t)$$

is designed so that

$$\hat{\alpha}(\xi v) = \Delta t \frac{2sin(\frac{\xi v \Delta t}{2})}{\xi v \Delta t} \sum_{k=0}^{L-1} \alpha_k e^{i\xi v \Delta t \frac{2k+1}{2}} \neq 0 \ \forall \xi \in [-\pi, \pi],$$

therefore invertible on the support of $\hat{u}(\xi)$. A detailed numerical implementation is available in section 3.

Step 1. Compute the ideal noiseless observed pixel value.
At pixel x the elementary noiseless pixel value is $e_k(x) = \left(\frac{1}{v}u * \mathbb{1}_{[k\Delta t,(k+1)\Delta t[}(\frac{\cdot}{v})\right)(x)$ computed using $\mathcal{F}^{-1}\left(\frac{1}{v}\hat{u}(\xi)\hat{\mathbb{1}}_{[k\Delta t,(k+1)\Delta t[}(\xi v)\right)(x)$.

Step 2. Compute the simulated observed pixel value.
By definition of the *numerical flutter shutter*, the observed pixel value is realization of the Poisson mixture $o(x) = \sum_{k=0}^{L-1} \alpha_k Poisson(e_k(x))$ (see sections 2.2, 2.3 below).

Step 3. Estimate the landscape pixel value by deconvolution.
The estimated landscape from the observed pixel value is then obtained by deconvolution $\mathfrak{u}_{est}(\xi) = o * \gamma$, where γ is the inverse filter satisfying $\left(u * \frac{1}{v}\alpha(\frac{\cdot}{v})\right) * \gamma = u$, computed by $\hat{u}_{est}(\xi) = \hat{o}(\xi)\hat{\gamma}(\xi) = \frac{\hat{o}(\xi)}{\hat{\alpha}(v\xi)}$.

Step 4. Compute error using RMSE and contrast invariant RMSE$^{\mathbf{CI}}$.
Straightforward with section 2.4.

2.2 Simulation of a Poisson Random Variable X with Intensity λ

The usual method, from Knuth [6], is described in algorithm 1.

Algorithm 1: Poisson random variable simulation

```
if (λ ≤ 50) then
    g = exp(−λ);
    em = −1;
    t = 1;
    rejected = true;
    while rejected do
        em = em + 1;
        t = t.rand (where rand is a uniform on [0, 1] random generator) ;
        if (t <= g) then
            X = em;
            rejected = true;
        end
    end
else
    simulate a Gaussian random variable X with mean and variance equal to λ, the method by
    Box et al. [3] is used here), and round it;
end
```

2.3 Signal to Noise Ratio Selection

The goal is to find a renormalization factor λ^2 such that the random variable X defined by

$$X \sim \frac{1}{\lambda^2}\text{Poisson}(\lambda^2 u(x))$$

has a $\text{SNR}(X) = k$ when $u(x) = 100$. It corresponds to tuning an averaged number of photons for a medium brightness value of 100. If $\lambda^2 = \frac{k^2}{100}$ then $\text{SNR}(X) = k$.

2.4 Contrast Invariant RMSE

In many cases reconstruction errors inherent to a method can be quantified using the Root-Mean-Squared-Error

$$\text{RMSE}\,(u, u_{est}) := \sqrt{\frac{\int_D |u(x) - u_{est}(x)|^2 dx}{\text{measure}(D)}}.$$

However, when a small contrast change occurs between the original and the processed image, the RMSE can become substantial, while the images remain perceptually indistinguishable. For example take an image $u(m, n)$ defined over a sub-domain $D \subset \mathbb{Z}^2$, and another one $u_{est} = u + 10$. Then $\text{RMSE}(u, u_{est}) = 10$ is large but does not reflect the quality of the reconstruction u_{est}. Comparatively, a convolution with for example a Gaussian can give a smaller RMSE while making considerable damage. This bias is avoided by normalizing the images before computing the RMSE. The principle of the normalization is that two images related to each other by a contrast change are perceptually equivalent. Their distance should reflect this fact and be zero. The Delon et al. midway equalization [4] is best suited for that purpose, because it equalizes the image histogram to a "midway" histogram depending on both images. By the midway operation both images undergo a minimal distortion and adopt exactly the same histogram. Thus we shall define the contrast invariant RMSE (RMSE^{CI}) by

$$\text{RMSE}^{\text{CI}} = \text{RMSE}(u_{est_{mid(u,u_{est})}}, u_{mid(u,u_{est})})$$

where $mid(u, u_{est})(= mid(u_{est}, u))$ is the midway histogram between u and u_{est}. $u_{mid(u,u_{est})}$ is the image u specified on the $mid(u, u_{est})$ histogram (having an histogram equal to $mid(u, u_{est})$) and $u_{est mid(u,u_{est})}$ is u_{est} specified on $mid(u_{est}, u)$.

3 Implementation

The described algorithm has been implemented in C++, the code and its documentation are available on the article web page (https://doi.org/10.5201/ipol.2012.t-fscs).

Given an image $u(m, n)$ defined for $m \in \{1, \dots, M\}$ and $n \in \{1, \dots, N\}$, a code $(\alpha_k)_{k=0,\dots,L-1}$, a velocity v, and an SNR level (and using a normalized $\Delta t = 1$), the *analog flutter shutter* camera and its restoration process are simulated as described in algorithm 2. The *numerical flutter shutter* implementation is detailed in algorithm 3. For color images each component is processed independently, the RMSE is averaged over all components. Assuming without loss of generality that the blur is in direction of the image lines, the following algorithm is repeated for each component.

4 On Line Demo

The purpose of this demo is to compare different strategies (snapshots, *numerical* and *analog flutter shutter*) on test examples and user uploaded images. The C++ source code (documented) used in the on line demo is available from the article web page (https://doi.org/10.5201/ipol.2012.t-fscs). Notice that the source code allows for *any* Δt (see section 2), but $\Delta t = 1$ is fixed here for the sake of simplicity.

Inputs are an image (PNG 8bits grayscale or 3×8 bits color), a *flutter shutter* code (binary sequence or gain function[2]), an SNR level, a velocity v, a type of *flutter shutter* (*analog* or *numerical*).

Outputs are the simulated observed image, the restored image obtained from the observed by deconvolution, the ground truth (crop of the input to ease the comparison), the residual noise image (difference between the restored and the landscape with stretched dynamic on $[0, 255]$ by an affine contrast change), the code sequence used, and the Fourier transform (modulus) of the code.

5 Examples

The purpose of this section is to compare experimentally different acquisition strategies: snapshot, *flutter shutter* using the Agrawal, Raskar et al. code [10, 11], a random code uniform over $[-1, 1]$, the *motion-invariant photography* code and the sinc code. All strategies are compared using the RMSE, the contrast invariant RMSE (RMSE^{CI}) and the visual image quality. A benchmark of acquisition strategies is given in the table 1.

6 Usual Codes

For comparison purposes the length L of all codes is 52 like in works by Agrawal, Raskar et al. [1, 2, 9, 11]. These strategies are snapshot, accumulation, Agrawal, Raskar et al. code [10, 11], random code, *motion-invariant photography* (MIP) code, and sinc code, explicitly given hereafter. The list also provides $\frac{||\alpha||_{L^1}}{sup\{|\alpha(t)|,\ t\in\mathbb{R}\}}$ the normalized L^1 norm of the associated *flutter shutter gain function* which gives the quantity of light integrated with respect to the code.

[2] A list of the most characteristic such codes is proposed to the demo users.

Algorithm 2: Analog flutter shutter

Step 1
begin
compute $\tilde{u}(m,n)$, the $2D-DFT$ of u:
for $m=-\frac{M}{2},\dots,\frac{M}{2}-1$ **do**
for $n=-\frac{N}{2},\dots,\frac{N}{2}-1$ **do**
$\tilde{u}(m,n)=\frac{1}{MN}\sum_{k=0}^{M-1}\sum_{l=0}^{N-1}u(k,l)\omega_M^{-km}\omega_N^{-nl}$ where $\omega_N=\exp\left(\frac{2i\pi}{N}\right)$;
end
end
compute the motion kernel generated by the *flutter shutter*):
for $m=-\frac{M}{2},\dots,\frac{M}{2}-1$ **do**
for $n=-\frac{N}{2},\dots,\frac{N}{2}-1$ **do**
$a(m,n)=\Delta t\frac{sin(\frac{\pi v\Delta tn}{N})}{\frac{\pi v\Delta tn}{N}}\sum_{k=0}^{L-1}\alpha_k e^{-i(\frac{2\pi v\Delta tn}{N})(k+0.5)}$;
end
end
compute the product of $\tilde{u}(m,n)$ and $a(m,n)$;
compute the inverse DFT of the previous, store it in $e(m,n)$ (here $e(m,n)$ is a coefficient of the ideal noiseless image observed, up to the periodization effect);
crop the result to avoid the periodization effect;
end
Step 2
begin
foreach (m,n) **do**
simulate the Poisson random variable with intensity $e(m,n)$ and the desired SNR using sections 2.2, 2.3;
store it in $o(m,n)$ (here $o(m,n)$ contains a simulation of the observed image);
end
end
Step 3
begin
use classic mirror symmetry among the columns obtain $u_s(m,n)$;
compute the $2D-DFT$ of u_s;
compute the motion kernel like in Step 2;
divide the $2D-DFT$ of u_s by the motion kernel;
compute the inverse $2D-DFT$ of the previous;
crop to remove the mirror symmetry (here the last operation gives a simulation of the restored knowing $o(m,n)$ and the code);
end
Step 4
begin
compute the RMSE and RMSE^{CI} after cropping to avoid border effects;
end

Algorithm 3: Numerical flutter shutter

Step 1
begin
compute $\tilde{u}(m,n)$, the $2D-DFT$ of u:
for $m=-\frac{M}{2},\ldots,\frac{M}{2}-1$ **do**
for $n=-\frac{N}{2},\ldots,\frac{N}{2}-1$ **do**
$\tilde{u}(m,n)=\frac{1}{MN}\sum_{k=0}^{M-1}\sum_{l=0}^{N-1}u(k,l)\omega_M^{-km}\omega_N^{-nl}$ where $\omega_N=\exp\left(\frac{2i\pi}{N}\right)$;
end
end
compute the elementary noiseless observations $e_k(m,n)$:
for $m=-\frac{M}{2},\ldots,\frac{M}{2}-1$ **do**
for $n=-\frac{N}{2},\ldots,\frac{N}{2}-1$ **do**
$c_k(m,n)=\Delta t\frac{sin(\frac{\pi v\Delta t n}{N})}{\frac{\pi v\Delta t n}{N}}e^{-i(\frac{2\pi v\Delta t n}{N})(k+0.5)}$;
end
end
compute the product of $\tilde{u}(m,n)$ and $c_k(m,n)$;
compute the inverse $2D-DFT$ of the previous, store it in $e_k(m,n)$ (here $e_k(m,n)$ contains the ideal noiseless observed up to the periodization effect);
crop the result to avoid periodization effect;
end
Step 1
begin
foreach (m,n) **do**
simulate the Poisson random variable with intensity $e_k(m,n)$ and the desired SNR using section 2.2, 2.3, store it in $o_k(m,n)$;
end
foreach (m,n) **do**
compute the observed $o(m,n)=\sum_{k=0}^{L-1}\alpha_k e_k(m,n)$ (here $o(m,n)$ contains a simulation of the observed image);
end
end
Steps 3-4
identical to the *analog flutter shutter*;

Code type	Snapshot	Agrawal, Raskar et al. code	Random code	MIP code	Sinc code
RMSE	1.34	2.34	2.01	2.11	1.33
RMSE^{CI}	1.33	3.14	2.48	2.10	1.33

Table 1: Quantitative (RMSE, RMSE^{CI}) comparison of different strategies for fixed velocity $v=1$ (so the blur support is 52 pixels, except for the snapshot) on the *House* test image. The random code performs better than the Agrawal, Raskar et al. code or the MIP code. Indeed the random code is (on average) closer to the optimum sinc code. Unsurprisingly, the best RMSE is obtained using the sinc code. But it beats only slightly the snapshot as predicted by the theory developed Tendero et al. [13]. The Levin et al. *motion-invariant photography* and the Agrawal, Raskar et al. code of *flutter shutter* are perfects examples of the Tendero et al. *flutter shutter paradox* [13]. More acquired photons does not necessarily imply a better SNR for the deconvolved image ("*a photon can kill another photon!*").

Figures 2, 3, 4, 5, 6 and 7 display the codes on the left, and on the right their Fourier transforms (modulus) given for normalized $\Delta t = 1$, for each code: the snapshot (figure 2), the standard blur (figure 3), Agrawal, Raskar et al. code (figure 4), the random code, (figure 5), the MIP code (figure 6), and the optimal sinc code (figure 7).

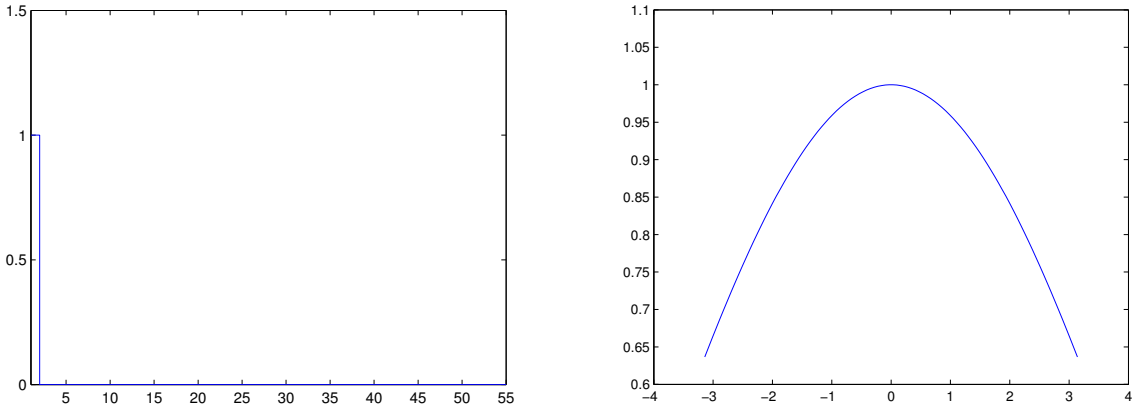

Figure 2: Snapshot. Left: the *flutter shutter gain function* for a snapshot. Right: the Fourier transform (modulus) of a snapshot.

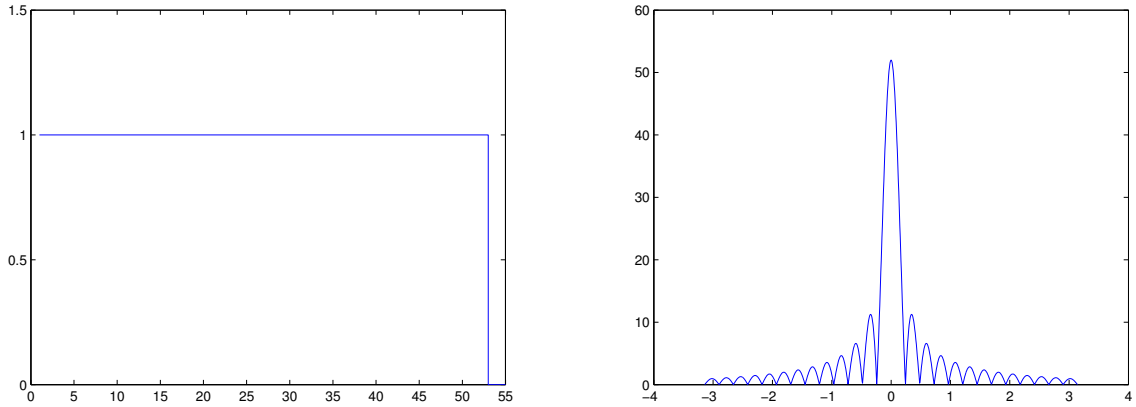

Figure 3: Accumulation code. Left: the *flutter shutter gain function* for the accumulation. Right: the Fourier transform (modulus) of the accumulation, invertible only when $Lv\Delta t < 2$.

Snapshot (figure 2)

$(1, 0, \ldots, 0)$, $\frac{||\alpha||_{L^1}}{sup\{|\alpha(t)|,\ t\in\mathbb{R}\}} = \Delta t$

The first code is a standard shutter strategy in classic cameras.

Accumulation (figure 3)

$(1, \ldots, 1)$, $\frac{||\alpha||_{L^1}}{sup\{|\alpha(t)|,\ t\in\mathbb{R}\}} = 52\Delta t$

This code is another standard shutter strategy in classic cameras.

Agrawal, Raskar et al. code (figure 4)

$(1, 0, 1, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 0, 1, 1, 1, 0, 0, 1, 0, 0, 1, 1, 0, 0, 1, 1, 1)$, $\frac{||\alpha||_{L^1}}{sup\{|\alpha(t)|,\ t\in\mathbb{R}\}} = 26\Delta t$

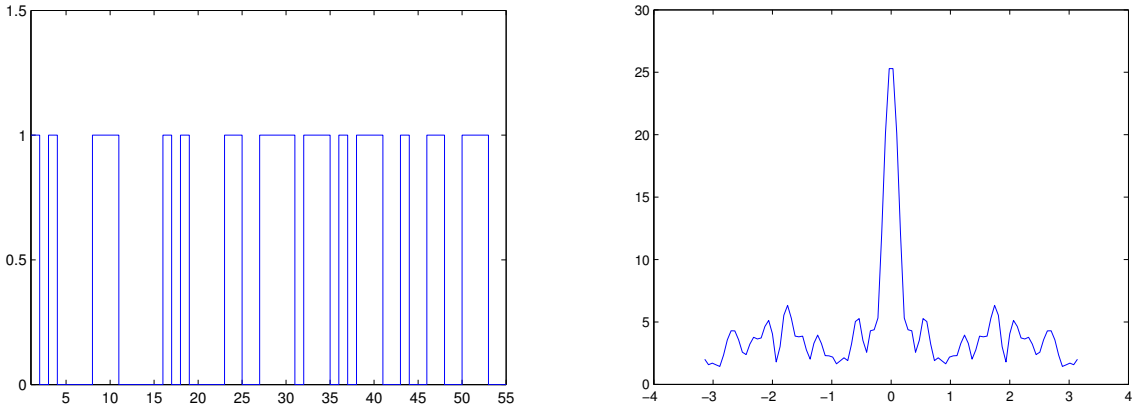

Figure 4: Agrawal, Raskar et al. code. Left: the binary *flutter shutter gain function* for the optimized Agrawal, Raskar et al. code. Right: The Fourier transform (modulus) of the Agrawal, Raskar et al. code found by an extensive research among binary sequences of length 52 [10, p. 5] and patent application [11].

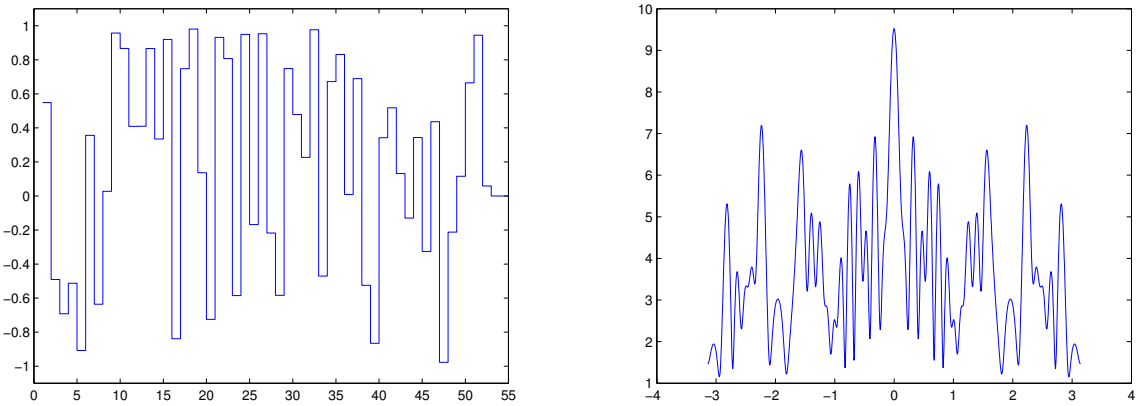

Figure 5: Random code. Left: the *flutter shutter gain function* for the random code. Right: the Fourier transform (modulus) of the random code. We shall see in table 1 that despite the lack of optimization this code performs better than the optimized Agrawal, Raskar et al. code.

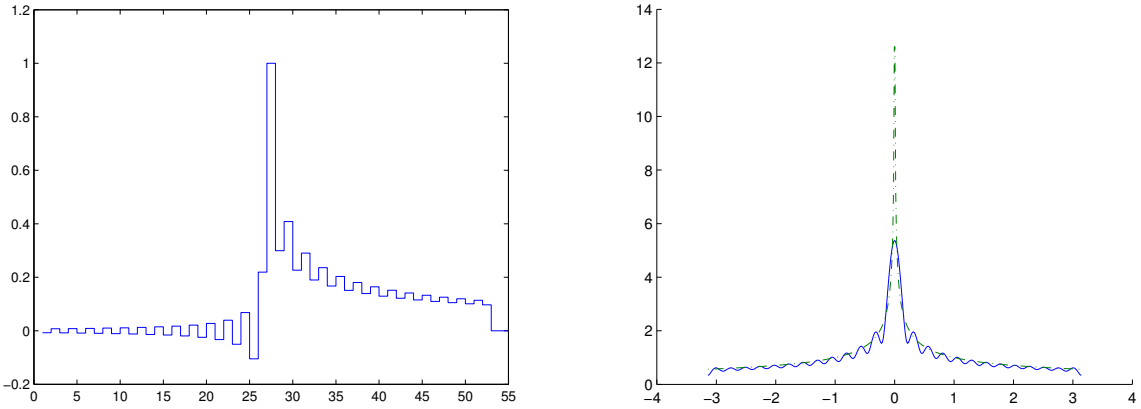

Figure 6: Motion-invariant photography code. Left: the *flutter shutter gain function* for the MIP code. Right: the Fourier transforms (modulus) of the *motion-invariant photography* code (in bold) and of the ideal *motion-invariant photography* function $\hat{\alpha}_{MIP-ideal}$ (dash dots line style). As predicted the proposed approximation is close to the ideal *motion-invariant photography* function $\hat{\alpha}_{MIP-ideal}$. As it is stated by Levin et al. [7] this apparatus performs better than the Agrawal, Raskar et al. code. However, it may be noticed that the both the ideal *motion-invariant photography* function and its piecewise constant approximation are far from the ideal *flutter shutter gain function* coming from a sinc (figure 7). Thus, the SNR of the recovered image is small compared to the best snapshot (table 1). This fact shall not surprise the reader, the constant acceleration apparatus was found by searching the best strategy among camera motions. Thus, the degrees of freedom of the *motion-invariant photography* are smaller than the *numerical flutter shutter*.

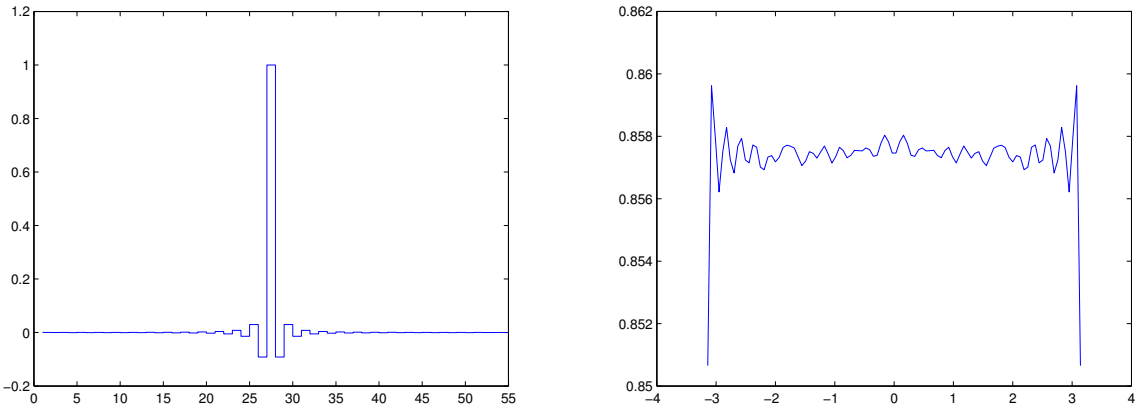

Figure 7: Sinc code. Left: the *flutter shutter gain function* for the sinc code (left). Right: the Fourier transform (modulus) of the sinc code, approximating the Fourier transform of the *ideal* gain function.

This code is published in their article [10, p. 5] and patent application [11].

Random code (figure 5)

(0.5491, −0.4903, −0.6919, −0.5125, −0.9079, 0.3560, −0.6357, 0.0275, 0.9568, 0.8670, 0.4087, 0.4097, 0.8659, 0.3344, 0.9198, −0.8389, 0.7476, 0.9808, 0.1366, −0.7247, 0.9320, 0.8069, −0.5848, 0.9493, −0.1682, 0.9533, −0.2173, −0.5834, 0.7483, 0.4785, 0.2266, 0.9764, −0.4708, 0.6723, 0.8312, 0.0084, 0.6892, −0.5245, −0.8651, 0.3417, 0.5183, 0.1317, −0.1301, 0.3432, −0.3262, 0.4367, −0.9771, −0.2120, 0.1160, 0.6648, 0.9446, 0.0590), $\frac{||\alpha||_{L^1}}{\int_{-\infty}^{\infty} \alpha(t)dt} \approx 3.0702\Delta t$

This code was generated from a uniform distribution over $[-1, 1]$.

Motion-invariant photography code (figure 6)

(−0.0072629, 0.0075565, −0.0078763, 0.0082229, −0.0086028, 0.0090181, −0.0094769, 0.0099834, −0.010549, 0.01118, −0.011893, 0.012702, −0.01363, 0.014703, −0.015961, 0.017451, −0.01925, 0.021458, −0.024239, 0.027842, −0.032697, 0.03958, −0.050083, 0.067993, −0.10493, 0.21912, 1, 0.29949, 0.40839, 0.22649, 0.29058, 0.18999, 0.23588, 0.16715, 0.20305, 0.15111, 0.18066, 0.13904, 0.16419, 0.12953, 0.15143, 0.12177, 0.14119, 0.11529, 0.13274, 0.10976, 0.12561, 0.10498, 0.11949, 0.10078, 0.11418, 0.097058), $\frac{||\alpha||_{L^1}}{sup\{|\alpha(t)|,\ t\in\mathbb{R}\}} \approx 6.0031\Delta t$

This code is the best L^2 approximation of the ideal *motion-invariant photography* function. More precisely, given v, it is a discretization of the $\alpha_{MIP-ideal}(t) = \frac{\mathbb{1}_{]0,\infty[}(t)}{\sqrt{t}}$ function with cutoff frequency equal to πv (given here for $|v|$=1 and $\Delta t = 1$). As it is stated by Levin et al. [7] this apparatus performs better than the Agrawal, Raskar et al. code. However, it may be noticed that the both the ideal *motion-invariant photography* function and its piecewise constant approximation are far from the ideal *flutter shutter gain function* coming from a sinc (figure 7). Thus, the SNR of the recovered image is small compared to the best snapshot (table 1). This fact shall not surprise the reader, the constant acceleration apparatus was found by searching the best strategy among camera motions. Thus, the degrees of freedom of the *motion-invariant photography* are smaller than the *numerical flutter shutter*.

Sinc code (figure 7)

(0.0002, −0.0002, 0.0002, −0.0003, 0.0003, −0.0003, 0.0003, −0.0004, 0.0004, −0.0005, 0.0005, −0.0006, 0.0007, −0.0008, 0.0009, −0.0011, 0.0014, −0.0017, 0.0021, −0.0027, 0.0037, −0.0053, 0.0082, −0.0141, 0.0296, −0.0917, 1.0000, −0.0917, 0.0296, −0.0141, 0.0082, −0.0053, 0.0037, −0.0027, 0.0021, −0.0017, 0.0014, −0.0011, 0.0009, −0.0008, 0.0007, −0.0006, 0.0005, −0.0005, 0.0004, −0.0004, 0.0003, −0.0003, 0.0003, −0.0003, 0.0002, −0.0002), $\frac{||\alpha||_{L^1}}{sup\{|\alpha(t)|,\ t\in\mathbb{R}\}} \approx 1.3364\Delta t$

This code is the best L^2 approximation of the *ideal* gain function. More precisely, given v, it is a discretization of the $\text{sinc}(x) = \frac{sin(\pi x)}{\pi x}$ function with cutoff frequency equal to πv (given here for $|v|$=1 and $\Delta t = 1$).

7 Experiments

Different strategies are here compared on the following image using the *numerical flutter shutter*, which gives a better SNR than an *analog flutter shutter*. Without loss of generality all results are given here for a normalized velocity $v = 1$ and a SNR equal to 100 for the gray level 100.

On the house image is applied successively a snapshot (figure 9), an accumulation code (figure 10), a classic motion blur code from Agrawal, Raskar et al. (figure 11), a random code (figure 12), the *motion-invariant photography* code (figure 13), and the optimal numerical flutter sinc code (figure 14).

Figure 8: The *House* test image.

Figure 9: Snapshot. Left: observed image. The blur interval length is equal to 1 pixel here. Middle: reconstructed image (RMSE = 1.34). Right: residual noise (difference between ground truth and reconstructed, dynamic normalized on $[0, 255]$ by an affine contrast change).

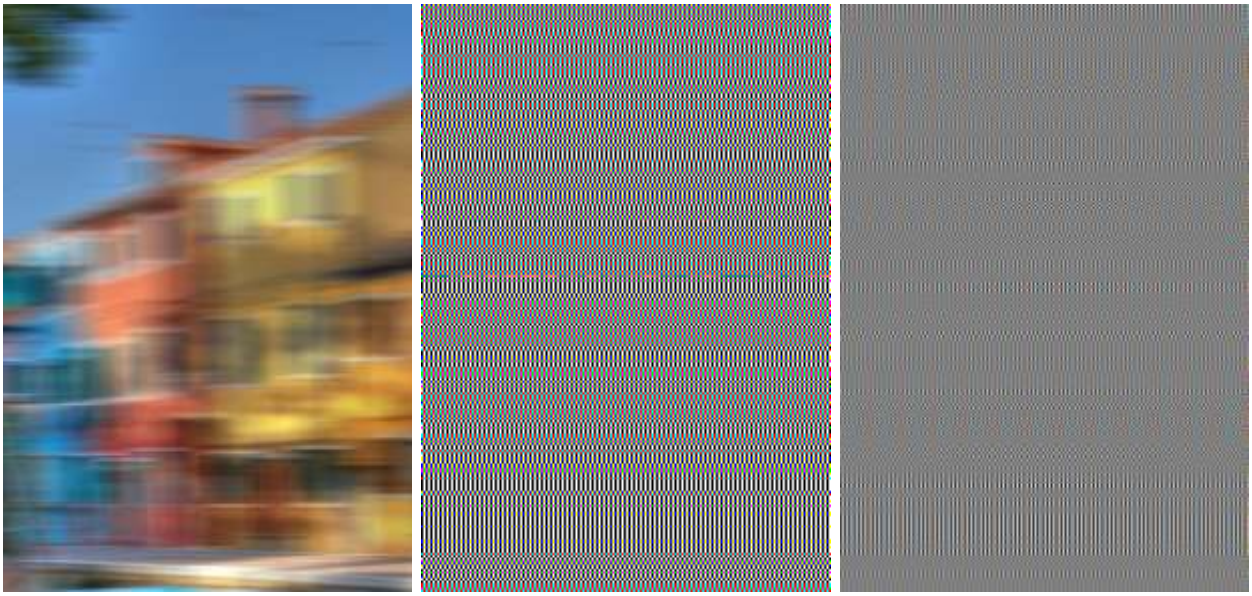

Figure 10: Accumulation code. Left: observed image. The blur interval length is equal to 52 pixels here. Middle: reconstructed image. Right: residual noise (difference between ground truth and reconstructed, dynamic normalized on $[0, 255]$ by an affine contrast change).

Figure 11: Agrawal, Raskar et al. code. Left: observed image. The blur interval length is equal to 52 pixels here. Center: reconstructed image (RMSE = 2.34). Right: residual noise (difference between ground truth and reconstructed, dynamic normalized on $[0, 255]$ by an affine contrast change).

Figure 12: Random code. Left: observed image. The blur interval length is equal to 52 pixels here. Middle: reconstructed image (RMSE = 2.01). Right: residual noise (difference between ground truth and reconstructed, dynamic normalized on $[0, 255]$ by an affine contrast change).

Figure 13: Motion-invariant photography code. Left: observed image. The blur interval length is equal to 52 pixels here. Center: reconstructed image (RMSE = 2.11). Right: residual noise (difference between ground truth and reconstructed, dynamic normalized on $[0, 255]$ by an affine contrast change).

Figure 14: Sinc code. Left: observed image. The blur interval length is equal to 52 pixels here. Middle: reconstructed image (RMSE = 1.33). Right: residual noise (difference between ground truth and reconstructed, dynamic normalized on $[0, 255]$ by an affine contrast change). The acquired image is "sharp", it is no surprise since the sinc code has a nearly constant Fourier transform thus, despite the motion it does not alter any frequency.

Acknowledgment

Work partially supported by the Direction Générale de l'Armement, the Office of Naval Research under grant N00014-97-1-0839 and by the European Research Council, advanced grant "Twelve Labours".

Image credits

Hervé Bry, Flickr CC-BY-NC-SA (http://www.flickr.com/photos/setaou/2162752903/.)

References

[1] A. Agrawal and R. Raskar. Resolving objects at higher resolution from a single motion-blurred image. In *Computer Vision and Pattern Recognition (CVPR), 2007 IEEE Conference on*, pages 1–8. IEEE, 2007. http://dx.doi.org/10.1109/CVPR.2007.383030.

[2] A. Agrawal and Y. Xu. Coded exposure deblurring: Optimized codes for PSF estimation and invertibility. In *Computer Vision and Pattern Recognition (CVPR), 2009 IEEE Conference on*, pages 2066–2073. IEEE, 2009. http://dx.doi.org/10.1109/CVPRW.2009.5206685.

[3] G.E.P. Box and M.E. Muller. A note on the generation of random normal deviates. *The Annals of Mathematical Statistics*, 29(2):610–611, 1958. http://dx.doi.org/10.1214/aoms/1177706645.

[4] J. Delon. Midway image equalization. *Journal of Mathematical Imaging and Vision*, 21(2):119–134, 2004. http://dx.doi.org/10.1023/B:JMIV.0000035178.72139.2d.

[5] J. Jelinek. Designing the optimal shutter sequences for the flutter shutter imaging method. In *Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series*, volume 7701, page 18, 2010. http://dx.doi.org/10.1117/12.850532.

[6] Donald E. Knuth. *The Art of Computer Programming, Volume II: Seminumerical Algorithms*. Addison-Wesley, 1969. http://dx.doi.org/10.1002/spe.4380120909.

[7] A. Levin, P. Sand, T.S. Cho, F. Durand, and W.T. Freeman. Motion-invariant photography. *ACM Transactions on Graphics (TOG)*, 27(3):71, 2008. http://dx.doi.org/10.1145/1360612.1360670.

[8] S. McCloskey, J. Jelinek, and K.W. Au. Method and system for determining shutter fluttering sequence, April 9 2009. US Patent 12/421,296.

[9] R. Raskar. Method and apparatus for deblurring images, July 13 2010. US Patent 7,756,407.

[10] R. Raskar, A. Agrawal, and J. Tumblin. Coded exposure photography: motion deblurring using fluttered shutter. *ACM Transactions on Graphics (TOG)*, 25(3):795–804, 2006. http://dx.doi.org/10.1145/1179352.1141957.

[11] R. Raskar, J. Tumblin, and A. Agrawal. Method for deblurring images using optimized temporal coding patterns, August 25 2009. US Patent 7,580,620.

[12] Y. Tendero, J-M Morel, and B. Rougé. A formalization of the flutter shutter. In *Proceedings of the 2nd International Workshop on New Computational Methods for Inverse Problems (NCMIP 2012)*, 2012. http://dx.doi.org/10.1088/1742-6596/386/1/012001.

[13] Y. Tendero, J-M. Morel, and B. Rougé. The *Flutter Shutter* Paradox. *SIAM Journal on Imaging Sciences*, page 35, 2012. (accepted).