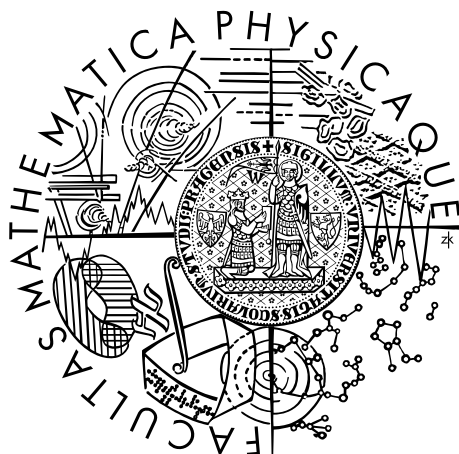


Charles University in Prague
Faculty of Mathematics and Physics

MASTER THESIS



Ondřej Šindelář

Použití gyroskopů a akcelerometrů k doostření fotografií pořízených mobilním telefonem

Ústav teorie informace a automatizace AV ČR, v.v.i.

Supervisor of the master thesis: Ing. Filip Šroubek Ph.D.

Study programme: Informatika

Specialization: Softwarové systémy

Prague 2012

I would like to thank my thesis supervisor Filip Šroubek for the initial idea, an introduction to the theoretical grounds needed for this project and his very helpful guidance throughout the research.

I am also grateful to the Institute of Information Theory and Automation for support and lending a mobile phone for testing.

Finally, I genuinely appreciate understanding and help devoted by my friends and family.

I declare that I carried out this master thesis independently, and only using the cited sources, literature and other professional sources.

I understand that my work relates to the rights and obligations under the Act No. 121/2000 Coll., the Copyright Act, as amended, in particular the fact that the Charles University in Prague has the right to conclude a license agreement on the use of this work as a school work pursuant to Section 60 paragraph 1 of the Copyright Act.

In Prague on 13th April, 2012

Ondřej Šindelář

Název práce: Použití gyroskopů a akcelerometrů k doostření fotografií pořízených mobilním telefonem

Autor: Ondřej Šindelář

Katedra: Kabinet software a výuky informatiky

Vedoucí diplomové práce: Ing. Filip Šroubek Ph.D., Ústav teorie informace a automatizace AV ČR, v.v.i.

Abstrakt: Fotografování s dlouhou expozicí z ruky s sebou nese problém rozmazání fotografie, které se těžko odstraňuje bez dodatečných informací. Cílem této práce bylo využít pohybových senzorů obsažených v moderních mobilních telefonech typu smartphone k zjištění přesného pohybu senzoru fotoaparátu během expozice a následně se pokusit pomocí těchto dat fotografii zaostřit. Byl navržen systém s dekonvolucí pomocí masky získané z naměřených dat gyroskopu a návrh byl implementován na platformě Android a ověřen na testovacím mobilním telefonu.

Klíčová slova: dekonvoluce, gyroskop, Android

Title: Using gyroscopes and accelerometers for removing blur in mobile phone photography

Author: Ondřej Šindelář

Department: Department of Software and Computer Science Education

Supervisor: Ing. Filip Šroubek Ph.D., Ústav teorie informace a automatizace AV ČR, v.v.i.

Abstract: Long exposure handheld photography is coupled with the problem of blurring, which is difficult to remove without additional information. The goal of this work was to utilize motion sensors contained in modern smartphones to detect exact motion track of the image sensor during the exposure and then to remove the blur from the resulting photograph according to this data. A system was proposed which performs deconvolution using a kernel from the recorded gyroscope data. An implementation on Android platform was proved on a test smartphone device.

Keywords: deconvolution, gyroscope, Android

Obsah

1	Introduction	3
2	Deconvolution	4
2.1	Model	4
2.2	Wiener filter	5
2.3	Constrained least squares	6
2.4	Richardson-Lucy method	7
2.5	Blind deconvolution	8
2.6	Spatially-variant blur	9
3	Analysis of camera motion blur	10
3.1	Camera projection	10
3.2	PSF derivation	11
4	Test device	17
4.1	Camera	17
4.2	Sensors	19
4.3	Processor	20
5	Implementation	21
5.1	Introduction	21
5.2	Application workflow	21
5.3	PSF estimation	22
5.4	Image resampling	23
5.5	FFT	24
5.6	Performance	25
5.7	Wiener filter	25
5.8	Image border blending	26
5.9	Camera-gyroscope synchronization	27
5.10	Preview	27
5.11	Normalization	28
5.12	Gyroscope calibration	29
5.13	Preferences	29
5.14	Conclusion	30
6	Experiments	31
6.1	Camera delay calibration	31
6.2	Noise measurements	34
6.3	Rolling shutter	35
7	Results	37
8	Discussion	40
8.1	Evaluation of the results	40
8.2	Problems	40
8.3	Future work	42

8.4 Related work	43
Conclusion	44
Bibliography	46
List of Abbreviations	47
Appendix A - Attachments	48
Appendix B - User manual	49

1. Introduction

Photography has experienced mass expansion during the last decade with consumer electronics boom coming along with dropping prices. However, the principles of image acquisition stay the same, so taking pictures at poor light condition especially using optics with small aperture must be compensated by longer exposure time. When shooting hand-held (without a tripod), this leads unavoidably to a motion blur.

Almost all modern smart mobile phones are equipped with a camera, but due to miniaturization of these devices, camera optics are very small and so is the image sensor. To compensate for the small aperture, signal gain (ISO value) of the image is usually increased to keep the exposure time within limits, where hand shake doesn't induce visible blur in the image. This in combination with relatively large number of pixels (comparable to normal digital cameras) on a small image sensor results in very noisy pictures. This problem is usually treated by some sort of denoising algorithm at the expense of losing considerable amount of details. The motion blur often cannot be eliminated completely anyway.

The blurring can be modeled as a convolution with a kernel function under certain circumstances. The inverse problem of finding the unblurred latent image, also known as deconvolution, is not easy due to the pervasive noise, but is solvable if the way how the image was blurred is known.

Recent smartphones and tablet computers are increasingly more often equipped with a set of motion sensors. We will be trying to treat the image with data obtained from these sensors taken into account. Especially gyroscopes, if present, provide good basis for relatively accurate estimate of the camera movement. That in turn may be used for deconvolution of the recorded photograph just after the exposition in the very same device, which becomes possible given the growing computing power of the handheld devices.

To the best of our knowledge, this was proved for the first time on a mobile device, only preceded by a project with a dedicated motion detection device and a separate calculation (as discussed in section 8.4).

In the next chapter, the model of blurring and possible solutions of the inverse process are described. Chapter 3 analyzes causes of motion blur and its detection using motion sensors. The device used for testing our efforts is introduced in chapter 4. All the different aspects of implementation phase are described in chapter 5. To examine some of the unknown quantities concerning the test device, we have arranged several experiments. Together with the results we list them in chapter 6. Chapter 7 brings the results we have achieved with the implemented application on the test device. In the last chapter 8 we review the results, speculate about downsides of our approach and evaluate possible future enhancements.

2. Deconvolution

2.1 Model

A standard model of image degradation is usually represented by relation

$$g = H(f) + n \quad (2.1)$$

where H is linear degradation operator and n is additive noise that appears in all signal paths to a certain extent. Image coordinate indices are omitted here for simplicity. Our goal is to find an estimate of the original image f that is as close as possible.

If H is spatially invariant, the equation 2.1 becomes

$$g = h * f + n \quad (2.2)$$

where “ $*$ ” denotes convolution and h is a so-called point-spread function, which is referred to as PSF, convolution kernel or just kernel.

The full form of the equation 2.2 for 2D image domain would be:

$$g(x, y) = \int_{u,v} h(u, v) f(x - u, y - v) du dv + n(x, y) \quad (2.3)$$

In the common case of discrete images, the integration obviously becomes summation over the pixels.

The spatial invariance assumption is not always met as we will see later.

The process of retrieving the original image f from 2.2 is called *deconvolution*. In many cases, the recorded image g is the only information available, so the degradation must be estimated first. This is called *blind deconvolution* (BD) and it is a very difficult problem given the number of unknowns. If multiple images of the same scene are provided though, the additional information makes the problem much more stable and even spatially variable PSF case can be solved as presented in [11].

According to the *Convolution theorem*, the convolution equation 2.2 has an equivalent representation in frequency domain:¹

$$G = HF + N \quad (2.4)$$

where G , H , F and N are *Fourier transforms* of the respective terms in lower-case in the original image-space domain equation — this convention will be kept throughout the publication. The deconvolution process is easier to carry out in the Fourier domain, because convolution in the spatial domain becomes a simple multiplication of the corresponding Fourier transforms. In practice, we usually work with data in discrete representation, so the discrete form of the Fourier transform is used (DFT).

Next we will explore some of the classical approaches to non-blind deconvolution.

¹This again is a shortened notation with both Fourier frequency arguments left out.

2.2 Wiener filter

A straightforward estimate of the original image would be ignoring the additive noise — a simple inversion of the degraded image by the kernel, that is

$$\hat{F} = \frac{G}{H} \quad (2.5)$$

However, at frequencies where H is very close to zero, any small amount of noise gets amplified and usually prevails in the result. It is true especially when the image is degraded by blurring, in which case the kernel is low-pass, so the small values of H as well as G are present mainly at high frequencies and therefore the already dominant noise is yet more emphasized.

This numerical instability is an attribute of an *ill-conditioned problem*, as is the deconvolution often described. The solution usually follows means of *regularization*, that is, expanding the problem by a certain prior condition of the original image which will make the problem easily and uniquely solvable.

The estimate is usually sought in the form of a so-called reconstruction filter:

$$\hat{f} = w * g \quad (2.6)$$

One option is to use the statistical least squares method, which considers f a random variable. That means minimization of the mean square error (MSE) with respect to w :

$$mse = E(|f - \hat{f}|^2)$$

which is equivalent in Fourier domain:

$$MSE = E(|F - \hat{F}|^2) \quad (2.7)$$

E denotes expected value operator here. Substituting by $\hat{F} = WG$ (2.6 in Fourier domain) we will get

$$\begin{aligned} MSE &= E(|F - WG|^2) \\ &= E(|F - W(HF + N)|^2) \end{aligned}$$

After expansion, given the condition that the noise N is independent of the signal F , setting the derivative of MSE with respect to W equal to 0 gives us finally

$$W = \frac{H^* S_f}{|H|^2 S_f + S_n} \quad (2.8)$$

where superscript $*$ denotes complex conjugation, S_f and S_n are power spectral densities of the signal f and the noise n , respectively. This comes from approximations $S_f = |F|^2$ and $S_n = |N|^2$. The solution of W in the equation 2.8 is called *Wiener filter* after Norbert Wiener, who first proposed it.

The whole deconvolution with Wiener filter can be written in the following form:

$$\hat{F} = WG = \frac{G}{H} \frac{|H|^2}{|H|^2 + \frac{S_n}{S_f}} \quad (2.9)$$

We can see it as an extension to the simple inversion filter 2.5 with the second regularization term that attenuates noise. If the fraction $\frac{S_n}{S_f}$ (reciprocal of so-called

signal to noise ratio – SNR) approaches infinity, the noise is not present at the given frequency and the filter behaves exactly as the inversion filter. Conversely, increasing amount of noise forces down the value at the particular frequency.

Although the power spectra of the original image and of the noise are not known, they can be estimated. Power spectrum of the observed image can be used in place of S_f , and S_n can usually be measured in advance from a photograph of a uniformly colored scene. However, if the observed image is blurred, it usually lacks high frequency spectrum, in which case using a selected constant value for S_f can help. The noise can be assumed to be white, making the S_n constant as well. The whole SNR may be therefore set as a constant without noticeable deterioration of the result. That establishes a simplified form of the Wiener filter:

$$W = \frac{H^*}{|H|^2 + \phi} \quad (2.10)$$

where ϕ is a constant estimation of the SNR inverse.

2.3 Constrained least squares

Another approach to the deconvolution problem is using the statistical point of view. We seek the most probable original image f given the observed image g , which can be denoted as maximizing $P(f|g)$. Using Bayes' rule this can be computed as

$$P(f|g) = \frac{P(g|f)P(f)}{P(g)} \quad (2.11)$$

The $P(g|f)$ term referred to as *likelihood* represents how the degradation of a given original image f looks like, $P(f)$ is determined by a prior knowledge of the original image, $P(g)$ is the known distribution of the observed image. Maximizing 2.11 is equivalent to minimizing

$$-\log(P(f|g)) = -\log(P(g|f)) - \log(P(f)) + \log(P(g)) \quad (2.12)$$

which is usually more suitable for solution and furthermore the last term $\log(P(g))$ can be left out as it is constant for the purpose of minimization. The method is called *maximum a posteriori* estimation (MAP). Maximizing only the likelihood term is called *maximum likelihood* estimation (ML).

As described in [13], in the case of Gaussian-distributed noise n , the negation of log-likelihood $-\log(P(g|f))$ (which equals to $-\log(P(n))$) is proportional to

$$|g - h * f|^2 \quad (2.13)$$

The term $P(f)$ may be seen as means for regularization as described in 2.2, since it allows for realization of an expected solution. We would like to impose a measure of smoothness of the original image, because less oscillation in image intensities is expected more likely. This can be done by setting a smoothing expression

$$|c * f|^2 \quad (2.14)$$

in place of the regularization term $-\log(P(f))$. The term $c * f$ is basically a high-pass filtered f , c is usually a second order differential filter, such as Laplacian

operator $\nabla^2 f$. In the discrete domain, it can be represented as a matrix

$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

The complete expression for minimization called *constrained least squares* (CLS) is then

$$|g - h * \hat{f}|^2 + \alpha |c * \hat{f}|^2 \quad (2.15)$$

which leads to this estimate in the Fourier domain:

$$\hat{F} = G \frac{H^*}{|H|^2 + \alpha |C|^2} \quad (2.16)$$

The coefficient α determines the degree of smoothing. This formula matches the Wiener filter except for the term $1/\text{SNR}$, which is here replaced by $\alpha |C|^2$.

Finding the optimal value usually requires some testing. There are objective methods for discovering a good regularization parameter value [1], [13], which need some prior information about the noise and the original image similarly as in the Wiener filter.

An alternative way to minimize 2.15 consists in solving it iteratively. It allows for constraining the domain in which the output image values are expected, such as cropping negative values. Consequently, the output image is also not limited to frequencies contained in the PSF as in 2.16. A representative of this iterative approach is the *iterative constrained Tikhonov-Miller method*, also called *iterative CLS* [1].

2.4 Richardson-Lucy method

This is another classical iterative method that was presented independently by W. H. Richardson [12] and L. B. Lucy [8]. It is based on likelihood maximization (ML) with assumption of Poisson distributed noise. The likelihood $P(g|f)$ to be maximized is equal to

$$\prod_{x,y} \left(\frac{(h * f)(x, y)^{g(x,y)} \exp[-(h * f)(x, y)]}{g(x, y)!} \right)$$

and logarithm of the likelihood is proportional to

$$\sum_{x,y} \left(g(x, y) \log \left(\frac{(h * f)(x, y)}{g(x, y)} \right) - (h * f)(x, y) + g(x, y) \right)$$

Using expectation-maximization algorithm, the iteration step of the estimated image is computed as

$$\hat{f}_{k+1}(x, y) = f_k(x, y) \left(h^T * \frac{g}{h * f_k} \right) (x, y) \quad (2.17)$$

The advantage of the *Richardson-Lucy algorithm* is that the estimate in every iteration step is non-negative and closer to the maximum likelihood.

The described algorithm is maximizing just the likelihood term in the equation 2.11 ($P(f)$ is considered constant, which implies no prior information about the original image), so various additional regularization methods may be applied — see [13].

2.5 Blind deconvolution

Since the problem we are solving assumes prior knowledge of the blur kernel, blind deconvolution methods have not been used in our solution; but for completeness and reference purposes, basic principles and some notable methods are mentioned here.

BD methods can be classified into two groups [2]: a priori blur identification methods and joint identification methods. The former class solves the two inherent problems separately. PSF is estimated first and then the image is deconvolved with this PSF using one of the classical non-blind deconvolution methods. The PSF estimate may utilize some of the well-known generic blur models (e.g. linear motion, atmospheric turbulence or out-of-focus) are other parametric prior models. However, most of the methods search for the kernel and image simultaneously.

For single-channel blind deconvolution, a stochastic model can be used:

$$P(f, h|g) \propto P(g|f, h)P(f)P(h) \quad (2.18)$$

which originates in the Bayes' theorem 2.11 mentioned earlier. It describes distribution of the desired latent image and the kernel blur in terms of likelihood of the observation and a priori distributions of the two unknowns. The relation is here expressed as proportionality — that is, equality except for a multiplicative constant. The observation g is considered fixed and so its distribution $P(g)$ (originally appearing in the denominator) becomes a constant and thus may be eliminated here in exchange for the proportionality operator.

The relationship 2.18 is then usually used for maximization of either the whole posterior probability (MAP) or only the likelihood part (ML) in a similar manner as in the non-blind mode. The maximization (or minimization the negative logarithm of the term) can be executed alternately with respect to both of the unknowns. This *alternating minimization scheme* is used for instance in the method by Shan et al. [15].

One of relatively more successful attempts that appeared in recent years is the work of Fergus et al. [4]. It approximates the full posterior distribution first and then computes the PSF with maximum marginal probability. This is done in hierarchical resolution manner — from coarse to fine — to prevent local minima. The resulting kernel is then applied to the observed image using the Richardson-Lucy deconvolution 2.4.

Some of the latest publications of BD methods showing very impressive results include for example the work of Xu and Jiaya [20] or of Wang et al. [19].

In the field of multichannel blind deconvolution, one of the best recent achievements is definitely the work by Šroubek and Flusser [17] that uses the MAP estimation and the alternating minimization described earlier. It is resilient against noise and misaligned input images unlike previous multichannel deconvolution attempts.

Overview of BD methods in more details can be found in [14].

2.6 Spatially-variant blur

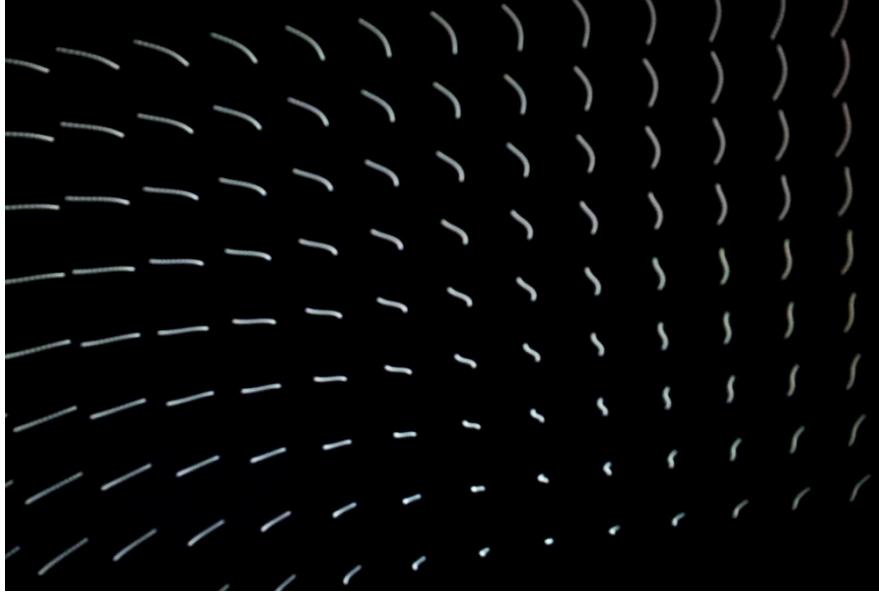


Figure 2.1: *Illustration of space-variant blur of point grid caused by a heavy rotation in the optical axis and a translation movement.*

If the blur cannot be considered the same for all the pixels in the image, we have to take space-variant PSF into account. The situation can be modeled similarly as in the space-invariant case, only with a more complex PSF term [14].

One intuitive way to deal with the spatial variance provided that the PSF is changing continuously over the image, is to split the solution or a portion of it into sections (along a rectangular grid) with particular versions of the PSF, and the total solution is then merged together possibly using linear interpolation. This approach is used for example in [9], where the probability minimization is done separately in the partitions.

However, if the blur is unknown, the estimation process is very slow. This may be aided by using input pictures with different exposure times (at least one sharp, but underexposed, one blurred) as shown in [16].

Variance of the blur can be caused by various influences, such as translation against a scene with a finite depth or rotation about the optical axis as illustrated in figure 2.1. They will be always present to some degree, and should be considered, but they pose a heavy obstruction for simple solutions. Under normal conditions, the harm made by neglecting the spatial variance of the blur is usually not enormous and since it is beyond the scope of the thesis, we consider it only for purposes of comparison and error measures in relation to our approach. More on this topic in the next chapter (section 3.2.2).

3. Analysis of camera motion blur

In this chapter we first show how an object in the scene is viewed by the camera as a projected image on its sensor. We then analyze trajectory on the image sensor left by a projection of a single point in the scene when the camera is in motion. This trace represents an image of the PSF from the deconvolution model (section 2.1), into which it can be easily transformed as shown here. We will then discuss circumstances under which the projected trajectory would be the same for any other point in the scene, which answers the question of PSF spatial variance conditions.

3.1 Camera projection

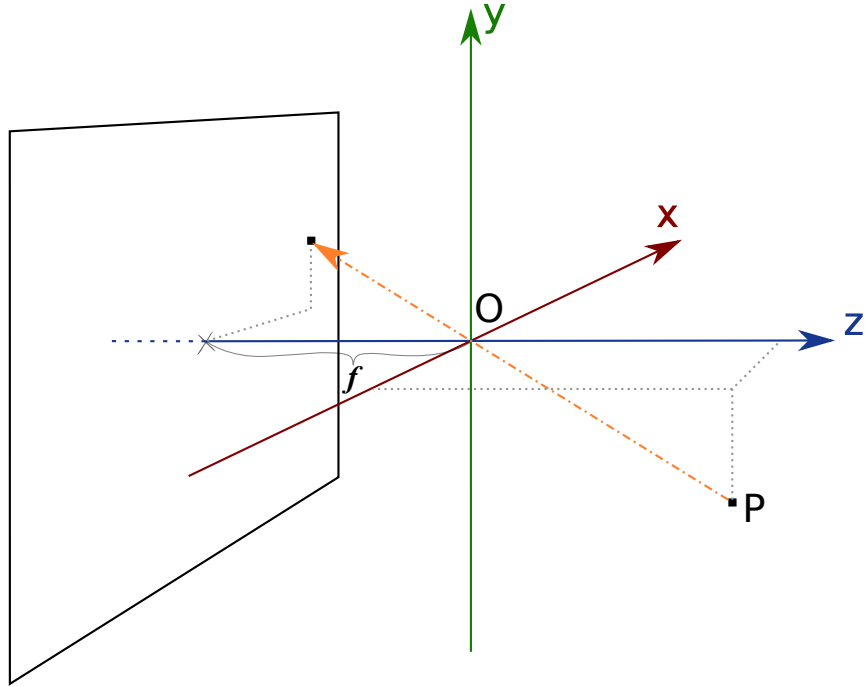


Figure 3.1: Camera projection of an object at the point P to a point on the image plane. The projection is created by sending a ray from P through the optical center O ; the intersection with image plane at distance f from O forms the point of projection. Axis z perpendicular to the image plane is the optical axis, x and y are image plane axes.

We are assuming a standard perspective (*rectilinear*) projection $\Pi : \mathbb{R}^3 \rightarrow \mathbb{R}^2$:

$$\Pi([x, y, z]^T) = \left[\frac{xf}{z}, \frac{yf}{z} \right]^T \quad (3.1)$$

This corresponds to a simplified *pinhole camera* model, which considers the optical center O to be coincident with the center of the standard orthogonal coordinate system with axes x , y and z . The axis z is the optical (principal) axis — it is perpendicular to the image sensor and intersecting its center, x and y are

the axes defining the image plane — see figure 3.1. This pinhole camera models the optical center O at a fixed distance f (*focal length*) from the image plane and projects every object point by intersecting a straight ray from that point going through O with the image plane. That leads directly to the equation 3.1 [6].

In practice, we obviously want to map the projection to screen or display coordinates, therefore the origin of the projected 2D coordinates is shifted to one of the corners of the display, so that all projected positions on the display have both coordinates positive.

A camera with refractive optics having a significant aperture size must be focused for objects close to the camera, which means that the lens equation

$$\frac{1}{d_1} + \frac{1}{d_2} = \frac{1}{f}$$

should be taken into account. An alternative form

$$d_2 = \frac{1}{\frac{1}{f} - \frac{1}{d_1}}$$

expresses distance d_2 behind O , where a focused real image of an object at distance d_1 in front of O will be formed. Therefore, to focus on a closer object, the camera lens must be rearranged so that the optical center will be positioned further from the image plane than the focal length f . Consequently, f in the projection equation 3.1 should be replaced by the distance d_2 . Note that for object distances approaching infinity, d_2 becomes equal to f as in the pinhole model equation. The difference shows up notably only at distances very close to the focal length, which is prevented by a limited focus distance and a very small f in the case of mobile phone camera. In practice, lens makers usually achieve close focusing partly by shortening the focal length or other more complex changes of lens elements' position. In conjunction with various distortions caused by optical aberrations of the lens and its corrections would make the model much more complicated and even indescribable. The required parameters of the camera lens (especially in mobile devices) are usually not documented anyway, so the projection 3.1 is probably the best generic model we can use.

3.2 PSF derivation

We need to find out characteristics of blurring caused by a camera motion — it will be used for computing a point spread function of the blur from motion sensors. The PSF will be then employed for deblurring of the recorded picture.

Let us assume that the camera aperture is open from time $\tau = 0$ to $\tau = \tau_{exp}$. During the exposure, the position and orientation of the camera is given by a translation vector $t(\tau) = [t_x(\tau), t_y(\tau), t_z(\tau)]^T$ and a 3×3 rotation matrix $R(\tau)$. Both matrices are expressed in relation to the origin and center of perspective O and relative to the initial position and orientation of the camera, that is, $t(0) = [0, 0, 0]^T$ and $R(0) = I_3$.¹

We will now describe motion blur PSF by means of calculating a trajectory of a static point light projected to the image plane. Instead of formulating the function

¹ I_n denotes identity matrix of size n .

$h(x, y)$ directly in the image space domain, we will use time parametrization for describing position $(x, y) = C(\tau)$ and value $V(\tau)$ of the light trace. It means that $h(C(\tau)) = V(\tau)$ in every instant in time $\tau \in \langle 0, \tau_{exp} \rangle$.

3.2.1 PSF value

The evaluation of the blur trace is based on the exposure formula:

$$H_v = \int_{\tau} E_v(\tau) d\tau \quad (3.2)$$

where H_v , E_v and τ are photometric (luminous) exposure, illuminance and time, respectively. The exposure, determined as the power of incident visible light energy per area over time, is the quantity measured by an image sensor as a linear value.²

Our PSF time evaluation function V is proportional to illuminance E_v as the brightness of the trace corresponds to the intensity of incident light in each moment during the exposure.

As the PSF of blurring is essentially a function expressing distribution of radiation energy [10], we will restrict it as non-negative and expect its total value \bar{h} make up the full unit, that is

$$\bar{h} = \int_{x,y} h(x, y) dx dy = 1$$

This gives us the same constraint for the value function V , too:

$$\int_{\tau} V(\tau) d\tau = \bar{h} = 1$$

That in combination with the relation to E_v sets the evaluation to

$$V(\tau) = \frac{E_v(\tau)}{H_v}$$

where H_v is the full exposure from time 0 to τ_{exp} . This follows from the exposure formula 3.2.

If the illuminance is constant over time (which was our precondition), we can write

$$V(\tau) = V = \frac{E_v}{H_v} = \frac{E_v}{E_v \cdot \tau_{exp}} = \frac{1}{\tau_{exp}} \quad (3.3)$$

This corresponds to shooting a statically illuminated scene with no moving objects.

For discretization of the PSF, evaluation of its small segments will be needed. The following quantification is formed from the above:

$$\int_{\tau}^{\tau+\delta\tau} V(\tau) d\tau = \frac{\delta\tau}{\tau_{exp}} \quad (3.4)$$

²For digital image sensors, the response to light is linear unlike the response of photographic film. However, this is true only for raw data before any adjustments like gamma correction.

3.2.2 PSF position

Let $\bar{p} = [\bar{x}, \bar{y}, \bar{z}]^T$ be the position of a point P . Then the coordinates of a curve defining the trace location is defined by

$$C(\tau) = \Pi \left(R(\tau) \begin{bmatrix} \bar{x} \\ \bar{y} \\ \bar{z} \end{bmatrix} + \begin{bmatrix} t_x(\tau) \\ t_y(\tau) \\ t_z(\tau) \end{bmatrix} \right) = \Pi (R(\tau)\bar{p} + t(\tau)) \quad (3.5)$$

It will be useful to express C in terms of the initial projection of the observed point:

$$c_0 = C(0) = \Pi(\bar{p}) = \left[\frac{\bar{x}f}{\bar{z}}, \frac{\bar{y}f}{\bar{z}} \right]^T$$

The coordinates of the point P can be expressed reversely as

$$\bar{p} = \begin{bmatrix} c_0/f \\ 1 \end{bmatrix} \bar{z}$$

The term $\begin{bmatrix} c_0/f \\ 1 \end{bmatrix}$ will be shortened as p_0 . The trace equation with substitutions is then formed as follows:

$$C = \Pi ((Rp_0) \bar{z} + t)$$

The time parameter τ is omitted for clear look. R can be split into $\begin{bmatrix} R' \\ r \end{bmatrix}$, where R' makes up first two rows and r is the last row of R . The transcript of C can then be further refactored:

$$\begin{aligned} C &= \Pi \left(\begin{bmatrix} R'p_0 \\ rp_0 \end{bmatrix} \bar{z} + t \right) \\ &= \frac{f}{rp_0\bar{z} + t_z} \left((R'p_0)\bar{z} + \begin{bmatrix} t_x \\ t_y \end{bmatrix} \right) \\ &= \frac{f}{rp_0 + t_z/\bar{z}} (R'p_0) + \frac{f}{rp_0\bar{z} + t_z} \begin{bmatrix} t_x \\ t_y \end{bmatrix} \end{aligned} \quad (3.6)$$

This form allows for some useful observations. We can notice that for objects further from camera (large \bar{z}) the influence of the translation upon changes of the projected trajectory becomes diminished. The term in the denominator of the second fraction in 3.6 ($rp_0\bar{z} + t_z = r\bar{p} + t_z$) is z -coordinate of the point P after transformations by R and t , therefore the whole term with t_x and t_y will be vanishing with \bar{z} approaching infinity. This assumes that P is not rotated so much that it would be close to O in z -coordinate. But in that case, the projection would be far beyond the field of view of the camera, which is always limited given the physical restrictions of the image sensor and camera optics. That's why this case may be ignored.

Increasing \bar{z} also suppresses t_z in the first fraction of the projection 3.6, which means that if the observed scene is far enough, the whole translation part of the camera movement can be neglected.

Figure 3.2 shows an illustration of the actual dependency of pixel-wise shift size on the image sensor given an example translation size of 1 mm in the image

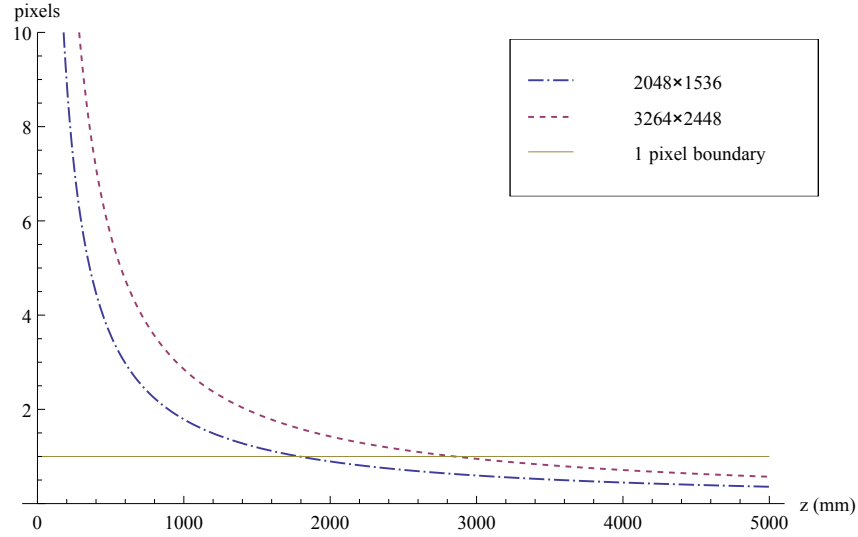


Figure 3.2: *Dependence of projection shift on object distance. Camera shift is 1 mm, angle of view is 60° , two curves represent different image sensor resolution.*

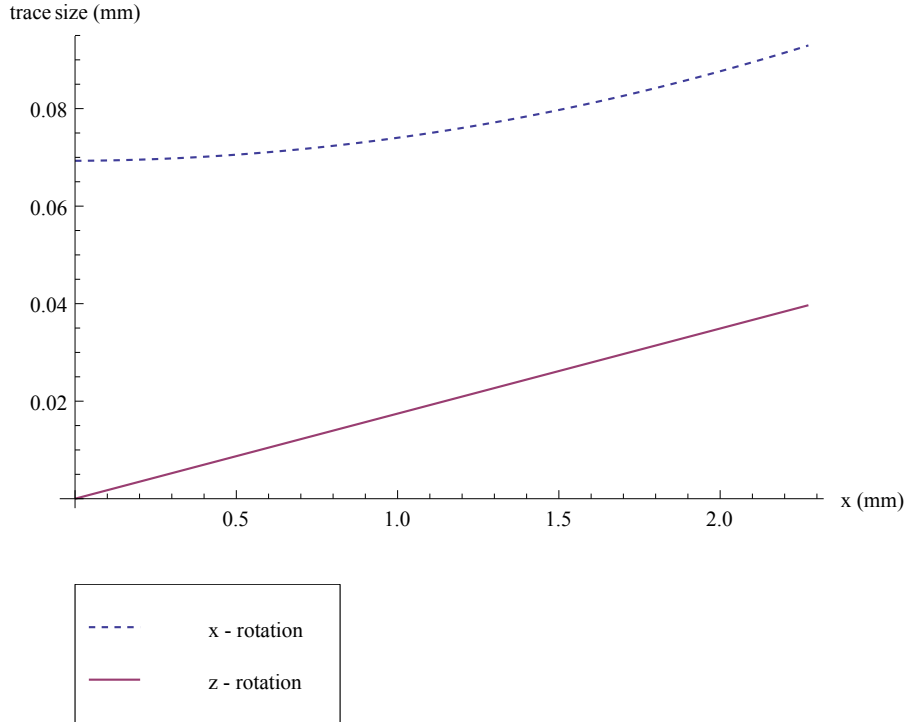


Figure 3.3: *Dependence of projection shift on rotation in different places on the sensor. Camera is rotated by one degree around the axes x and z respectively; the graph shows size of the projected trace depending on a distance from the image sensor center. Full sensor extent is shown — angle of view is 60° .*

plane. Camera parameters used here reflect the real test device as described in chapter 4. The relation was computed from the formula 3.6 ignoring the rotation.

For comparison, if the same 1 mm shift was applied on the sides of the phone in opposite directions perpendicular to the image plane, the resulting rotation of around $1/60$ rad would create a trace approximately 33, resp. 52 pixels long

(depending on which of the two image resolutions was used, but regardless of the scene depth). The same trace length would be produced by translation with an observed object in the distance of about 54 mm (outside camera focus range) and the further is the camera from the object, the less influence on the trace translation has. This estimate might be inaccurate, but nevertheless it shows that the effect of the translation part of the camera shake is prominent primarily in macro photography and ignoring that may be a viable simplification especially because of the enormous complexity of the problem with otherwise depth-dependent PSF.

One concern intuitively preventing considering the PSF as space-invariant is rotation in the optical axis. This type of rotation applied with a point light placed in the center of the picture (on the axis z) would obviously leave the projection unchanged, but points outside the center would form arc-shaped traces the larger the closer to the corners.

A comparison of the effect of the same rotation amount around the optical axis (z) and an image plane axis (x) in a real example of the test device is shown in figure 3.3. One can observe that z -rotation is much less pronounced everywhere on the sensor provided that the camera is rotated with equal amount or less around the optical axis than around the image plane axes, which is a fair assumption under normal circumstances.

The last obstacle in regarding the PSF as space-invariant is already manifested in the figure 3.3 by the trace size of x -rotation projection slightly growing along the the distance from the center. That originates in the rectilinear projection 3.1 which casts a point at an angle α from the optical axis to a point at a distance of $f \cdot \tan(\alpha)$ from the image center. The tangent function is locally very similar to a linear dependency near the center, but diverges to infinity at $\pm 90^\circ$ as shown in figure 3.4.

3.2.3 Space-invariant PSF conditions

For support of our intention to regard the PSF as space-invariant, we will now sum up the factors that justify ignoring resulting inconveniences.

As described earlier, the effects of translation are suppressed if distance to the observed object is large enough.

The z -rotation and rectilinear projection express themselves in terms of PSF space variability increasingly more towards the image borders. That favors the use of lenses with longer focal length, which implies narrower field of view, where rotation in axes x and y corresponds to linear projection more closely and z -rotation is less prevalent than the other rotations.

Anyway, at least a center part of the picture should be spared of the effects of the z -rotation and the perspective projection even if the PSF will be computed with disregard of the z -rotation and with linear approximation of the rotation projection.

Another influence not yet discussed causes a slight variance of the blur function, which is rolling shutter (described in section 4.1.2). This again can be considered as invariant in a narrowed part of the image and so it does not prohibit the use of space-invariant PSF approximation.

Because all the phenomena causing changes of PSF throughout the image

plane may be at least in most cases ignored, we will now introduce the final space-invariant PSF in a form constructed by replacing the projection $f \cdot \tan(\alpha)$ with $f \cdot \alpha$ as mentioned before. The PSF position function is then written as

$$C(\tau) = c_0 + f \begin{bmatrix} R_x(\tau) \\ R_y(\tau) \end{bmatrix} \quad (3.7)$$

where R_x and R_y are rotation angles about the respective axes.

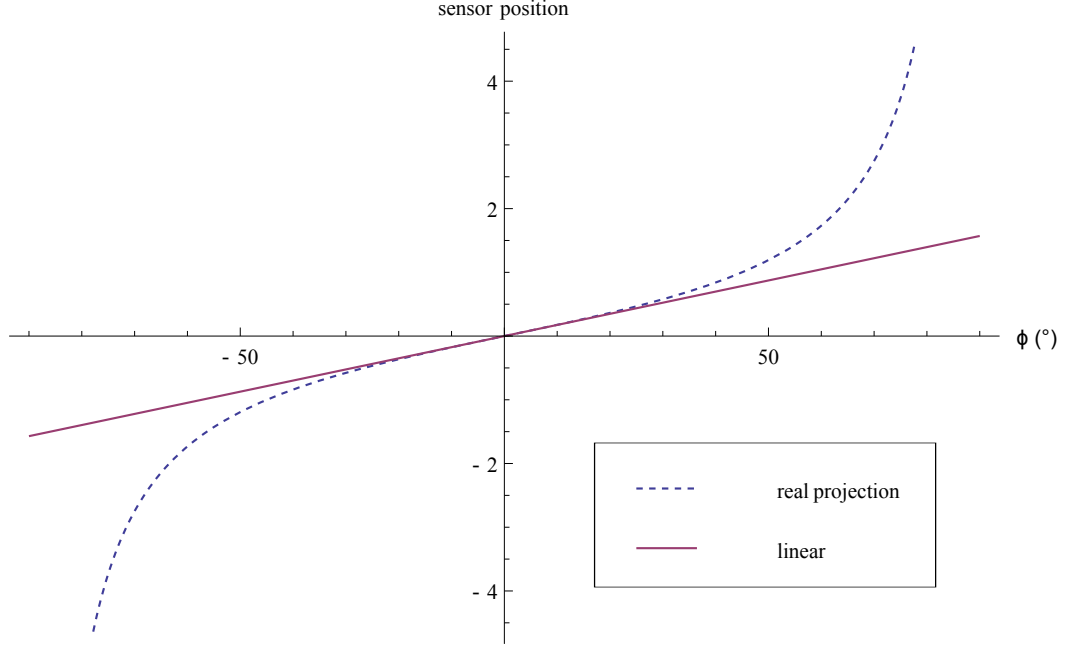


Figure 3.4: Projection distance from the image center as a function of rotation angle of an observed point against the optical axis — compared with a linear dependence. The projection position on a sensor is measured in multiples of focal length f .

4. Test device

For purposes of research testing, we have used a *Samsung Galaxy S II (GT-I9100)* smartphone. It is equipped with all the apparatus needed for our experiments. Namely it contains

- 8 Mpx camera
- motion sensors — 3-axis accelerometer and gyroscope
- a fast CPU — 1.2 GHz dual-core ARM Cortex-A9
- plenty of RAM — 1 GB



Software on the phone is managed by *Android* operating system which allows easy developing of user applications, offers quite detailed and well-structured programming documentation and has a large and rapidly growing community of users and developers.

Android contains an application framework which provides running Java code on a specialized virtual machine called *Dalvik*. That simplifies creating ordinary applications with support for manipulating hardware devices to some extent as well. Most of the applications, even those coming with Android, are written this way.

Computationally demanding applications, however, perform suboptimally given the overhead of virtual machine interpreter. Also the access to the hardware is limited to a general level as the application programming interface (API) must cover a common properties of all the numerous smartphone devices. There is an alternative programming toolkit called *Android NDK*, which allows executing native code, that is, bypass the Dalvik interpreter by writing code in C or C++ which will be compiled directly to the machine native code. Packed in a library loaded from the Java code, the native code may be executed using the standard *Java Native Interface* (JNI).

Following an example of Linux, it is based on, the Android itself was created as an open source project, which also helps developers. For instance, debugging code that interacts with OS through the API may be aided by viewing its source code. However, some pieces of the native code are not available, such as some model-specific device drivers, which prevents complete control over the hardware (more details about implementation to be found in chapter 5).

4.1 Camera

Besides a low-resolution front-facing camera, there is the main camera on the back of the phone with a CMOS chip with a native resolution of 3264×2448 . It has a mechanical auto-focus; dark scenes may be illuminated by a LED flash light, it is efficient only for close objects. Some useful camera attributes may be acquired programmatically using the camera API (through class `Camera.Parameters`). Focal length f and horizontal view angle α are reported as 3.97 mm and 59.6

degrees, respectively. The angle of view is at least roughly correct, as verified by a simple measurement experiment.

These two obtained values can be used to compute the physical size of the image sensor: the width is given by $m = 2f \tan(\alpha/2)$, which is about 4.55 mm. The relatively high pixel count on such a small chip results in a pixel density that is always associated with considerable amount of noise. The inner processing of a taken photo therefore apparently incorporates some sort of denoising algorithm (see section 6.2), which is not desirable for our purposes, as it interferes with our model preconditions. Unfortunately, it cannot be disabled, and that is also the case for any manual exposure control. The exposure is always automatic, one can merely choose from a set of scene modes, which prioritize a combination of the actual exposure settings. These are, apart from the flash use, shutter time and ISO value (electronic amplification of the signal coming from the image sensor). As said, these values cannot be set explicitly, but may be at least discovered by reading EXIF attributes from the resulting photograph. The lens aperture is fixed and has a value of f/2.6 (according to the EXIF data), which in combination with the focal length yields aperture size of about 1.5 mm.

4.1.1 Raw image

The camera outputs only a JPEG-compressed image (level of compression is adjustable at least) — the raw picture from the sensor is not available. This is probably a drawback of supporting many different devices by the common Android API since dealing with different raw formats of various camera chips would be complicated. It is also apparently hardly asked for feature, JPEG pictures are sufficient for most common users. Furthermore there are memory issues, as the uncompressed image requires considerable amount of memory of which there had not been much to spare in mobile devices until recently. Access to the unmodified original image would be very beneficial for our purposes, at least for finding how the original was modified. Without that we cannot be sure if some kind of post-processing (such as gamma-correction, sharpening, etc.) was applied, which could impair correct deconvolution.

We invested a considerable effort into getting control of lower hardware levels through the Android NDK, so that raw images output and manual exposure settings would become allowed, although it is not officially supported. It was, however, unsuccessful, partly because of the complex multi-layer inter-process communication with transitions between Java and native code, but also for the closed-source camera driver. It is possible to build the whole Android OS from the source code, but the device drivers must be added in binary form from an already working device. It is the case for Samsung devices, at least.

4.1.2 Rolling shutter

One thing not obvious on the first approach, which is important to take into account, is the shutter mechanism. In small camera systems like this, mechanical shutter is mostly substituted by an electronic one. The usual way, as used in CMOS sensors, is to read values of illuminated pixels successively line by line. This takes some time, which results in the picture not taken at a single moment,

but with a slight time delay between the first and last pixel row. This process called *rolling shutter* causes no problems with static scenes, but moving objects during the image acquisition create unnatural skewing effects. For our issue, this means that the PSF should be considered as slowly changing in the vertical direction. See an experiment in section 6.3 where the problem is explored in a greater detail.

4.1.3 Timing

Perhaps the most critical part of a proper deconvolution process turned out to be the correct synchronization of the image acquisition and the recorded movement data. The OS provides a callback at the end of the exposure, but as stated in the documentation, it is meant for playing back a shutter sound, not for precise time measurements. After examination, it was concluded that it could not be used as a reliable means of synchronization. A treatment of this problem is shown in the implementation section 5.9.

4.2 Sensors

The Samsung smartphone is equipped with different sensors for location detection; GPS module can be used for long-distance location tracking, real-time tracking is provided by a magnetometer, accelerometer and a gyroscope. Magnetometer shows a vector of magnetic field direction, which can be used for estimating absolute orientation relative to Earth's magnetic field. It is not very accurate, though, as the weak and unstable geomagnetic field can be easily disrupted, for instance by other magnetic objects.

Relative orientation changes may be measured by a gyroscope. It is given as angular velocity in all three axes measured in rad/s. Accelerometer detects rate of translation velocity change, that is, acceleration in all three axes in m/s². Because we are interested in tracing spatial position, the acceleration has to be twice integrated over time, which introduces significant level of uncertainty. Finding the angular orientation, on the other hand, involves only one integration, which makes it generally more precise.

Maximum sampling frequency we have been able to obtain is about 100 Hz both for the gyroscope and the accelerometer using the standard Android API. Using native code though, it is possible to achieve a close to constant sampling rate of the gyroscope with frequency of 213 Hz, which improves the orientation estimation even more.

We don't know the exact physical location of the motion sensors within the mobile phone, but for accelerometer it makes no difference as the translation speed without rotation is the same in the whole body of the phone assuming it is rigid; rotation about a shifted center yields the same rotation angle, but adds an extra translation amount. This error might be compensated for if we knew the shift distance, but as we decided not to consider the translation altogether (see chapter 5), the rotation is treated as if its center would coincide with the optical center of the camera. This is probably not a poor assumption due to the very limited space for the components in the compact phone body.

4.3 Processor

The phone has a modern CPU with two cores of the ARM Cortex-A9 family, which implements ARM v7 instruction set. It features a high-performance floating-point unit and a SIMD extension called *NEON* for vector operations. This is helpful for our purposes, because the deconvolution computations are very demanding for the floating point operations.

5. Implementation

5.1 Introduction

Using the knowledge of the deconvolution process described in chapter 2, we proposed an application for image deblurring using data from motion sensors. It has been implemented for Android platform and tested on the test device shown in chapter 4.

The building environment is based on the *Android SDK*, which provides building tools and debugger, and *Eclipse IDE* enhanced by a plugin called *Android Development Tools* (ADT) — it allows for simple managing of projects, remote debugging on mobile devices, creating GUI for mobile applications and easily deploying the built application package, among other things.

Our main concern during the implementation phase was to create a user-friendly application that would utilize the available data and computation power with the best effort and produce a result within reasonable time. Since most of the advanced techniques mentioned in the chapter 2 usually take minutes or even hours to complete on high-performance desktop computers, it makes them unsuitable for our purposes. Instead, we explored the potential of using the simple *Wiener filter* (section 2.2) and afterwards we compared the results to those using more complex methods.

Wiener deconvolution works with a space-invariant PSF, so we used the simple PSF position formula 3.7. That means we only needed the orientation data, which were gathered from the gyroscope.

The normal automatic camera scene mode is constrained to a maximum shutter speed of $1/17$ s (as found out experimentally), which is compensated by maximum ISO value 400 to avoid blurring. To take advantage of the deconvolution, we turn on the *Night* scene mode, which allows exposure times up to $1/2$ s long. Maximum ISO value is decreased to 250 at the same time, which also limits the noise amount to our avail.

5.2 Application workflow

The application code-named *GyroStab* consists of a preferences screen for setting various program options and the main screen that works as an enhanced version of the standard camera application. Once the user clicks the screen to take a picture, gyroscope samples are set for recording, camera is focused and then the picture is acquired. The recorded orientation data is integrated to create a motion trace, which acts as a PSF for deconvolution. Together with the PSF, all three color channels of the image are then converted to the Fourier domain, where the deconvolution filter is applied. After converting back to the spatial domain, the resulting image is assembled and saved. The result is finally presented to the user via default image viewer application.

5.3 PSF estimation

In the native code we first create a list of gyroscope samples in all three axes together with precise timestamps of the moment they were obtained. The list contains information about rotational movement during the exposure with overlaps caused by delays before and after the actual exposure. The list is therefore cut out so that it covers only the exposure time (more on the time synchronization in section 5.9).

The samples of angular velocities are integrated to form a continuous curve describing angles of camera rotation relative to the initial orientation. The angles are first approximated in the samples points iteratively by a formula

$$\phi_n = \phi_{n-1} + \omega_n \cdot dt_n \quad (5.1)$$

where dt_n is time difference between the current and next sample, ω is angular velocity of the current sample and ϕ_n is the resulting n -th angle. The angle curve is started at the origin point $[0, 0]$ and will be shifted later (see section 5.3.2).

The list of angles is then directly converted to a projection on the image sensor according to the equation 3.7. That means just a simple multiplication by a constant transferring an angle of deviation to a shift in pixels on the sensor. Pixel resolution of the image is naturally considered here.

The continuous curve of PSF positions is formed by a linear interpolation of the list of points computed before. That is, the points are successively connected by lines. To satisfy the evaluation condition 3.4, we have to maintain energy weight given for each curve segment, which is the time spent between current two samples relative to the total exposure time, or $w = dt_n / t_{exp}$. The curve also needs to be rasterized into pixel matrix for deconvolution. The sum of pixel values along the rasterized line must be equal to the weight w .

5.3.1 Line drawing algorithm

We used a method inspired by the Bresenham's algorithm [3] that renders pixels covered by the line only partially with proportionally lower value. The algorithm walks along the pixels in the faster growing coordinate (suppose it's x) and plots a full value pixel at the appropriate rounded y position. However, if the line crosses to next y row in the current x position, the pixels at both y positions are set to split values proportional to the crossing point. The special case of line ending is handled in a similar manner: the last pixel (or two pixels) is evaluated by the part of the pixel which the line covers in the x direction. See figure 5.1 for a sample of line rendering.

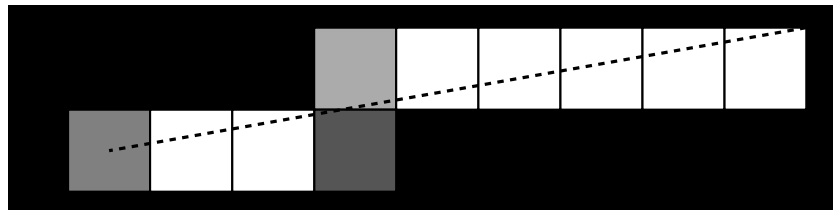


Figure 5.1: An example of a single rendered line from $[0,0]$ to $[8.5, 1.5]$. Grey level determines pixel value (black – none, white – full)

The described algorithm ensures that the weight is evenly distributed along both axes in the range given by the line extent. It also has an advantage that only those pixels crossed by the line are influenced by the line, unlike other common anti-aliased line drawing algorithms.

To evaluate the whole line segment with weight w , fully covered pixels will be set to w/L , where L is the longer of the line projections to axes x and y .

The PSF traced as a polyline is not the best estimation of the path, a smoother curve with derivatives at the sample points equal to the respective measured angular velocities would be superior because the velocity is always continuous. However, the sampling frequency is high enough, so the difference from the simple estimation wouldn't probably be noticeable anyway.

5.3.2 PSF shift

The final adjustment of the PSF will be done by shifting its origin. It should be placed in an imaginary point that is supposed to be the initial static position. In other words, we have to choose the camera orientation in which it would have been staying if it had not been shaking. The correct point is evidently indeterministic so we have to set it with the best effort. Strictly speaking, no guess is wrong, it only causes a shift of the deconvolved image. Our choice was to shift the PSF to the center of gravity, since it is intuitively the orientation where the camera spent most of the time. In any case, it is placed in between boundaries of all orientation localities, therefore it should not cause apparent deviation of the resulting image from the original.

The PSF is plotted on an image canvas of the same size as the image that will be deconvolved, because we need a DFT of the blur kernel with the same dimensions to be able to perform a frequency-wise filtering. The PSF shifting is carried out circularly in respect to these dimensions.

5.4 Image resampling

The most demanding part of otherwise simple Wiener filter is the conversion of the source image to the frequency domain — discrete Fourier transform (DFT), which is done by the FFT algorithm (see section 5.5). It has a complexity $O(N \log N)$ for the input sequence of length N , which is $O(N^2 \log N)$ for an image N pixels wide in our case. The size of the input image is therefore critical for the computation times.

To make the application as responsive as possible, we use a scaled-down input image. The output image size is selectable from a list of predefined sizes that are the fastest for the used FFT implementation (see section 5.5 for details and speed comparison). Shrinking of the image doesn't reduce much of its details, at least subjectively. The small optics image sensor seems to prevent obtaining the full resolution worth of data, so even a reduction to a default resolution 2048×1536 appears to preserve most of the original image information.

Android has an implementation of bitmap resampling, which turned out to be lacking of quality — quantization artifacts appear on results, mainly in uniform and dark areas. Therefore we created our own implementation of a resampling algorithm.

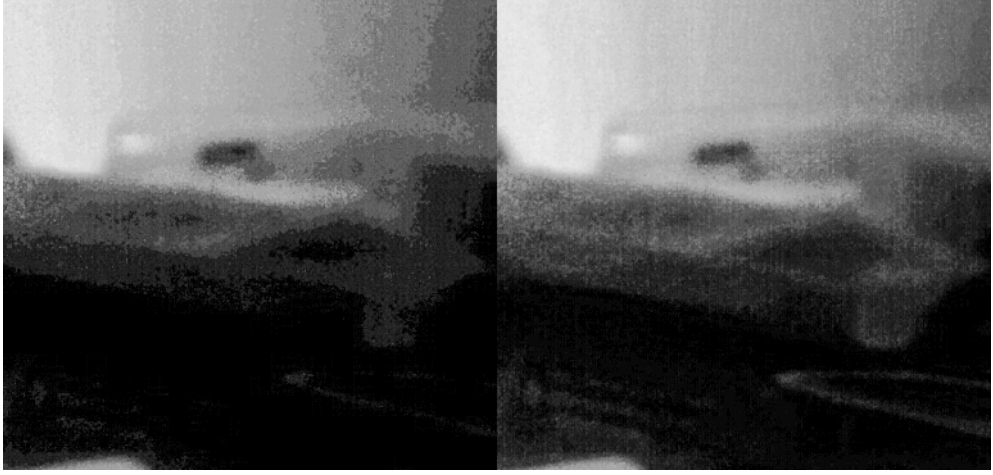


Figure 5.2: A blow-up of resample comparison — left picture is produced by Android bitmap services, our implementation is on the right. Lightness distribution of the illustration is adjusted to enhance the differences.

Each output pixel value is counted up as a weighted sum of all pixels it is covering. The weights correspond to the parts of the original pixels that are covered by the new (bigger) pixel. As the same weights are used in each row and column, they can be precomputed just once for the longer dimension.

The described algorithm is slower than the Android implementation, even in native code, but it is compensated by much better outputs which by all means improves the deconvolution results.

5.5 FFT

For transformation of the image data into frequency domain we used a library called *FFTW*, which is generally regarded as the fastest FFT implementation available [5]. Since the version 3.3.1-beta, ARM NEON extensions are supported, which makes it very fast even on mobile devices that contain a CPU implementing NEON vector instruction set.

FFTW is built as a static library, which is afterwards linked with the rest of the native C code forming a native shared library, which is then dynamically loaded from the Java code through JNI. This build chain has been set up after a rather troublesome phase of careful tuning of building parameters, but it is working fine in the end.

You can see a benchmark of transforming a test image of arbitrary sizes in table 5.1. The test was performed by FFTW compiled with different processor options. At first no floating point coprocessor unit and extensions were implied, which corresponds to older ARM CPUs (floating point instructions are executed using sequences of standard integer instructions). Then the FPU and NEON support was turned on — the enormous speed-up is clearly demonstrated here. The best times in the last column are achieved taking advantage of both CPU cores and the multi-threaded architecture offered by FFTW.

The third row shows one property of the FFTW implementation: it performs best on input data whose size is a factor of small primes. Just two extra pixels in each dimension are enough to slow down the calculation more than twice.

resolution	no NEON, no hardware FPU	NEON, 1 core	NEON, 2 cores
1536×1152	2900	185	110
2048×1536	5300	330	195
2050×1538	—	1000	540
3264×2448	21200	1450	800

Table 5.1: Time measurement of one FFT transform (real to complex) of input image of different sizes performed with different CPU settings. Average times of ten iterations are shown in milliseconds.

DFT outputs a sequence of complex numbers of the same size as input, but if the input numbers are only real, a half of the output values is redundant as two values in mirrored positions are complex conjugates. FFTW takes advantages of this by offering a function that outputs only about a half of complex numbers, so the total byte size of the output stays roughly the same as of the input (a little more due to padding). The filtering is then performed just on the half of the frequency data, which are afterwards converted back to the spatial domain by the inverse DFT. This saves a considerable amount of memory needed for storing large bitmaps and is also somewhat faster than the full complex-to-complex transform.

5.6 Performance

The Fourier transform must be applied several times: once for the PSF and twice for each color channel — one of which is an inverse. The inverse transform takes practically the same amount of time as the normal forward FFT. That yields a total of 7 FFT operations. With some overhead of bitmap transfers, the whole deconvolution phase for input image resolution 2048×1536 takes about 2.6 seconds, while the whole processing of the image since the camera shutter is done in a little over 6 seconds. This includes image resizing, counting the PSF, compressing and saving the original and result image files.

5.7 Wiener filter

The equation 2.9 is used for the filtering of spectrum values. The question is what SNR value should be used in the formula. The original intention was to use noise power spectrum amplified according to a used ISO value, but the experiment 6.2 showed no particular dependency of the power spectrum on the ISO speed. Therefore we decided to leave the spectrum of noise constant at all frequencies.

The power spectrum of signal can be estimated from the spectrum of the recorded image or simply as a single value from its variance. Testing proved that using the spectrum of the observed image rarely helps, it usually rather suppresses the high-frequency details, because the blurred image itself is often low-passed. Commonly a constant preset SNR value according to the equation 2.10 produces results subjectively as good or even better as shown in figure 5.3.

The SNR value was chosen as a constant at first, and has eventually become adjustable in the preferences screen.

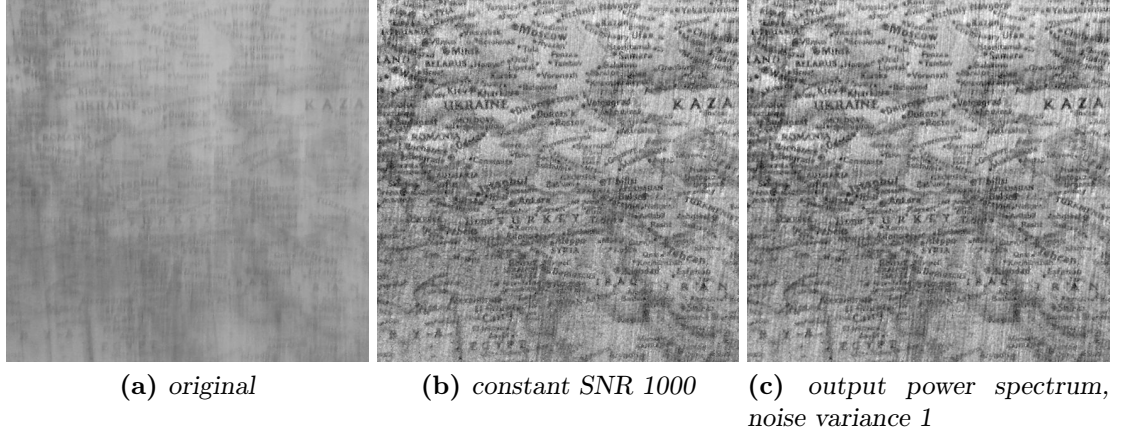


Figure 5.3: Comparison of different choices for SNR value.

5.8 Image border blending

An image converted using DFT is described using Fourier coefficients, which by definition implies that the image is periodically extending to infinity in both directions. The deconvolution considers the image as such and tries to deal with the edges arising at the image borders as if they were created by a convolution with the given blur kernel. That causes the edges appear duplicated as “ghosts” around the borders as seen in figure 5.4; it is also called a *ringing artifact*.

To eliminate this effect, it is desirable to reduce pixel value difference between opposite sides of the image. Our approach was to blend border pixels with their complements on the opposite side. Pixels within the width of the PSF in a given direction are affected with linearly decreasing strength towards the picture center. That is, the pixels in the first row are painted with the color of the last row and the following rows are mixed with the opposite rows with decreasing weight. This ensures a smooth blending of the borders exactly at the reach of the PSF and it dramatically reduces border ringing artifacts. The side-effect is loss of information in the border regions, but there is not enough data for their restoration from a single picture anyway.

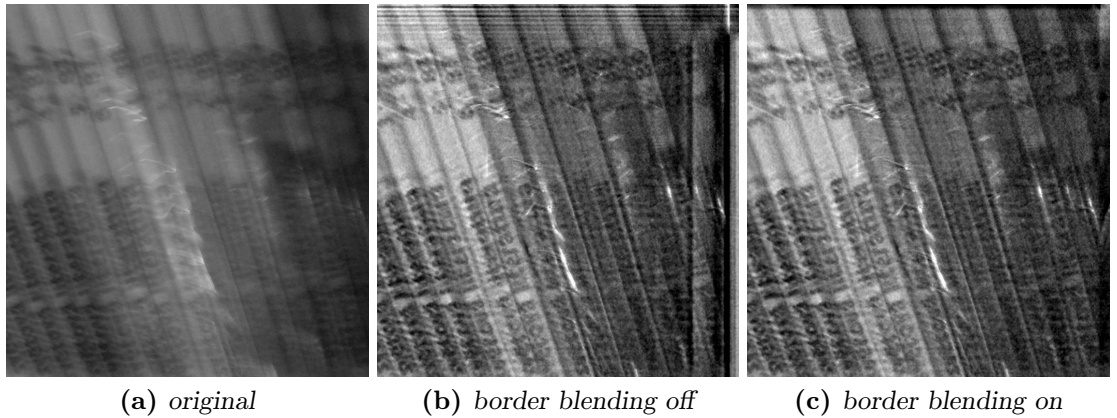


Figure 5.4: Illustration of the image border blending effect. Only the upper-right corner of the image is shown.

5.9 Camera–gyroscope synchronization

One of the key factors of successful estimation of the convolution mask proved to be the correct synchronization between the camera exposure and recorded data of its motion. The gyroscope measurements are expected to be accurately enough associated with the moment in which they were recorded — this has been verified by a simple experiment that shows a real-time gyroscope data.

Camera timing accuracy is not that good, though. As outlined in the section 4.1, the callback notification of finishing the image capture (`Camera.Shutter-Callback`) showed no reliability in the sense of a constant relation to the real exposure end. It was coming way too late and the delay seemed greatly variable.

Measuring the exposure from the moment of initiating the snapshot also exhibits big variance and is not precise enough.

We therefore searched for another way to detect the exposure period; eventually the logging was explored. Android has a central log subsystem, which is available for reading by anybody given a special application permission. This gives us opportunity to see log entries written by other applications and even device drivers. It turned out that the camera driver is logging a specific line every time it is going to take a picture. This was utilized as a synchronization event that determines relatively accurate moment before the actual exposure. There is still a slight delay, which is however fairly close to constant. This was tested and quantified by an experiment described in section 6.1 and a solution to compensate for the remaining variability is implemented according to section 5.10.

The log synchronization should be considered a highly unstable and unportable solution as it is tuned just for the given device at the specific version. Its behavior may change after software update and it likely will not work at all on other devices. At the time, no other reliable means of synchronization was known, though (see section 6.1 for additional relevant information).

The beginning of desired motion samples is thus in our application determined by reading a timestamp of the specific log entry; the end point is given by adding the shutter time, which is stored in EXIF data of the original image file.

5.10 Preview

To overcome the variability still contained in estimation of the PSF, we implemented a method of previewing the deconvolution with different parameters, which entrusts the user with making a decision about the correct setting. Currently only the delay of exposure start is varied, but it would be easy to add another parameters to choose from.

The different options are presented to the user as a grid of images showing the same part of the picture. The well-established panning and zooming using touch gestures was implemented for synchronous moving of all the images so that the user can explore various parts of the image and compare differences.

To speed up the preview and also for memory-saving reasons, deconvolution is performed using only one color channel and with a lower resolution of the input image.

When the user chooses an appropriate option by clicking the corresponding

grid image, the full deconvolution is executed as normally with the chosen parameter.

The number of images in the grid may be configured in the preferences screen.

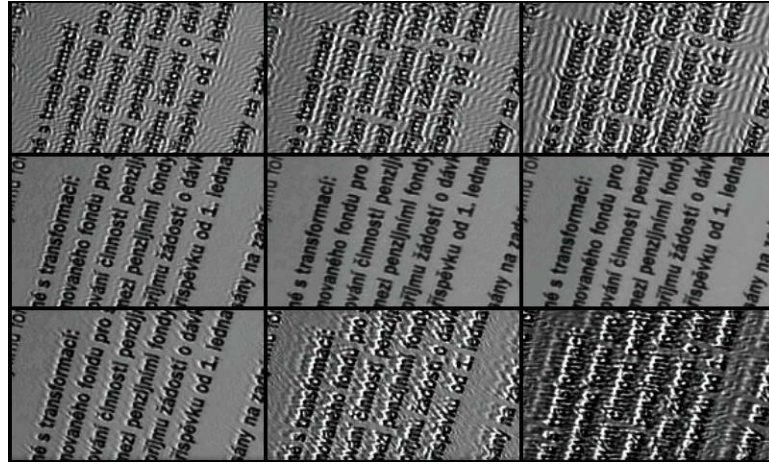


Figure 5.5: A screenshot of a sample deconvolution preview.

5.11 Normalization

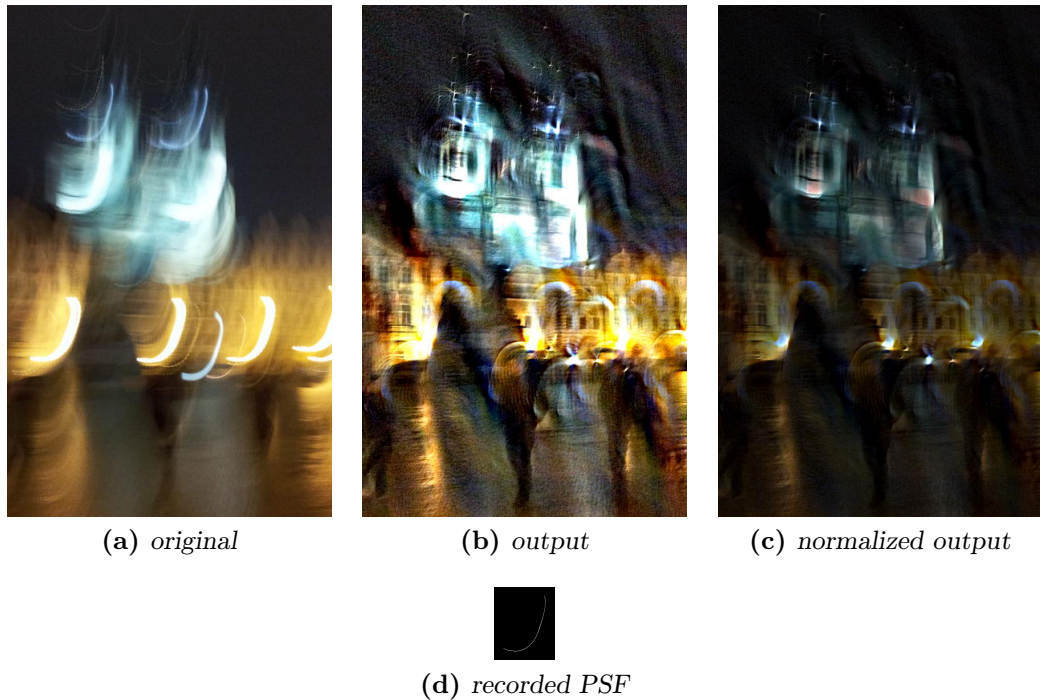


Figure 5.6: Illustration of the effect of output normalization.

It was observed that the deconvolution sometimes produces an output with pixel values outside the permitted range 0–255. It often appears when a photograph of a scene with big contrast or bright highlights is taken during heavy movement. The resulting burned pixels must be clipped, which discards some amount of useful information.

The burning may arise from a situation, when a projection of a very bright object is spread over a large area due to camera movement, whereas a static exposure would concentrate the same light energy on a smaller spot, which would become burned out and the underlying pixels' values would be clipped. The same thing is done by deconvolution, as it de facto accumulates the pixels spread by the PSF back into the static position. The difference is that we can handle the leaked values at will, unlike the hardware image sensor.

The overflowed values may be also inflicted by altering the original recorded image, for example by gamma correction or other non-linear brightness adjustment. That causes values of the deconvolved pixels end up different as they would in a static exposure, as the deconvolution treats the recorded image as created by a linear convolution with the original.

To save the clipped values, the output may be normalized. As the extreme values are often exceeding the limits multiple times, it would not be wise to normalize by the extremes. Therefore we are finding a brightness point above which lies a small selected portion of outlying pixels that are to be clipped. This is done by counting a histogram of the image within the full brightness range of the image, which was found first. The clip point is then found by summing counts of pixels in successive brightness levels of the histogram.

As demonstrated by an example in the figure 5.6, the normalization may preserve some details of the scene that would be otherwise hidden by burned out highlights, even though the result may become somewhat dull.

5.12 Gyroscope calibration

As discovered, the gyroscope sensor suffers from a significant systematic error. A detected deviation of about 0.01 rad/s would be projected as a pixel shift of 10 pixels at the center of the chip, given a test case of a 500 ms exposure and image resolution set to 2048×1536 . The largest deviation was found on the z axis which is not taken into account, but even the other axes show bias that produces a trace a couple of pixels large.

To suppress the systematic error, we implemented a calibration process of the gyroscope. Assuming a steady position during a calibration period, recorded samples are averaged and saved for use in the recording of the actual movement. The recorded biased values are then fixed by subtracting the systematic deviation.

The bias proved to be long-term stable at least in an order of magnitude, but it still changes slowly over time, so usually a slight error of roughly a single-pixel deviation for the longest exposure time cannot be prevented. This is not such a problem in real life, because natural hand shake induces much greater values.

Nevertheless, the inevitable gyroscope noise causes loss of precision of the estimated blur kernel and thus is one of the factors deteriorating the deconvolution result.

5.13 Preferences

The preferences screen allows for managing all the settings needed for the application. The notable ones are:

- output image size
- SNR value for the Wiener filter
- default camera–gyroscope synchronization delay
- preview settings: on/off switch, grid size, delay variance to choose from
- normalization switch, clipping amount
- gyroscope calibration
- options to save the recorded sensor data and estimated PSF

5.14 Conclusion

The final product of the implemented Android application is a distributable package file **GyroStab.apk** (included on the attached CD). Target platform is Android with version 2.3.3 or later.

As mentioned earlier, performance-critical portions of the application are written in native C code (calling FFT, Wiener filter, sensor recording image and resampling), the rest (application skeleton and GUI, PSF rendering) is in Java.

The application was developed and thoroughly tested solely on the Samsung Galaxy S II smartphone (see chapter 4), only basic functionality was checked on another Samsung tablet device. Because of the synchronization method described in section 5.9 optimized for the particular device model, it is not expected that the application will be functional (at least with the same precision) on other devices.

Other portability problem is the processor architecture inherently required by the native library, as we want to take advantage of the special instruction set. Android offers packing of native libraries for different CPU architectures, which could be employed together with tuning importable code for other devices. More test devices would be needed, though.

6. Experiments

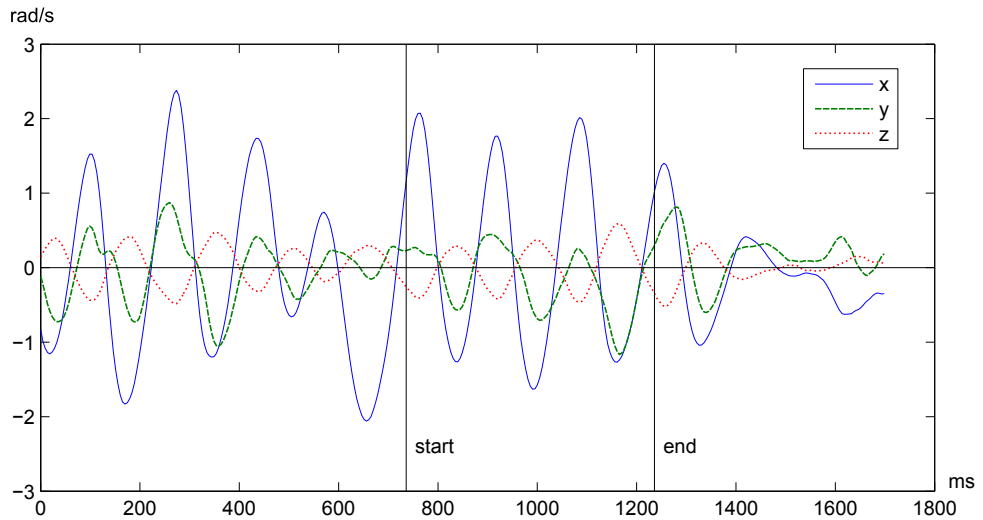
6.1 Camera delay calibration

As described in section 5.9, synchronization between camera and gyroscope using camera driver log allowed for getting closer to the moment of exposure start, but still left a time gap that needed a quantification as accurate as possible. We arranged an experiment for that using a laser pointer. The laser light beam was aimed at a diffuse surface in a dark room to create a static point light. It was then photographed using our application with a forced random movement to induce large blur. A specific trick was needed to force the camera use shorter exposition time than the longest available (1/2 s). The room was illuminated by a lamp during the preview phase, when the camera measures the light conditions, and the lamp was quickly switched off just before the exposure to let the laser light remain isolated.

The resulting movement data was used in Matlab to count a set of PSF kernels by varying the start delay and exposure time; these were then matched against the blur trace of the point light. Parameters of the best match were included in a list of results from all trials, which formed a statistical data set.



(a) Matching of an estimated PSF (white, thin) onto a photograph.



(b) Graph of a whole gyroscope measurement for one photograph, the actual exposure takes place between the markers “start” and “end”.

Figure 6.1: An example of a single trial of the experiment. A matched PSF and a corresponding gyroscope graph with marked exposure estimation is shown here. Exposure time from EXIF 500 ms, 1699 ms of gyroscope data measured, 670 ms start delay detected from the log. +66 ms start delay and +6 ms end delay detected in calibration.

The matching was done by first registering the rendered PSF onto the captured image using normalized cross-correlation. Then a similarity was computed by a least squares of difference of the two images. Both of these steps used thresholded versions of the images. The PSF with the highest similarity was finally selected as the best match.

The shift parameters in some of the results unfortunately had to be manually tuned because of a slight estimation error of the PSF that caused the least squares algorithm prefer shorter traces and the PSF was therefore chosen incorrectly.

The resulting list of corrections from 113 test cases, as shown in figure 6.2, may be summarized as follows:

- delay of exposure start: mean 66.5 ms, standard deviation 7.6 ms
- exposure end correction: mean 2.9 ms, standard deviation 9.1 ms

The exposure duration correction may be neglected as a result within the statistical error. The exposure start compensation of 67 ms has been set as a default value in the application, but it can be changed in the preferences screen. The substantial variance as seen in the graph confirms that the preview of different timing parameters as described in section 5.10 is an appropriate measure, because the deviations undetected by the device can be easily identified by the user.

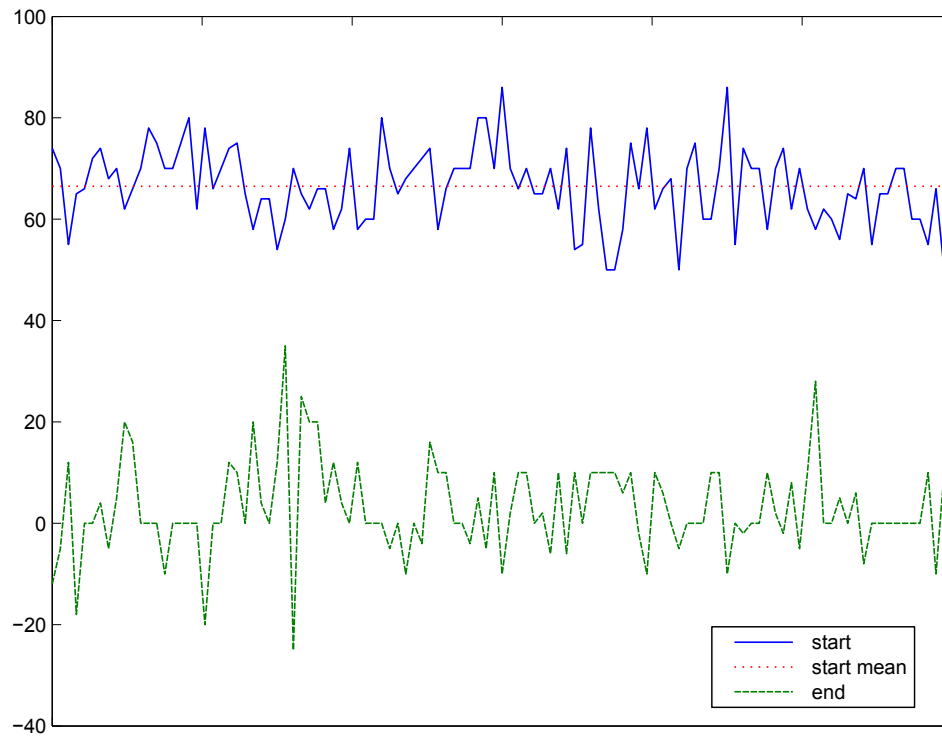


Figure 6.2: *Corrections of shutter start and end time as determined for all test cases. Vertical axis shows time in milliseconds*

A curious result emerged from an attempt to compare time of the actual end of exposure (as measured in the experiment) with the end of gyroscope measurement — that is, a time of the shutter callback that is supposed to notify about the exposure end (see section 5.9). The graph of the comparison in figure 6.3 shows a

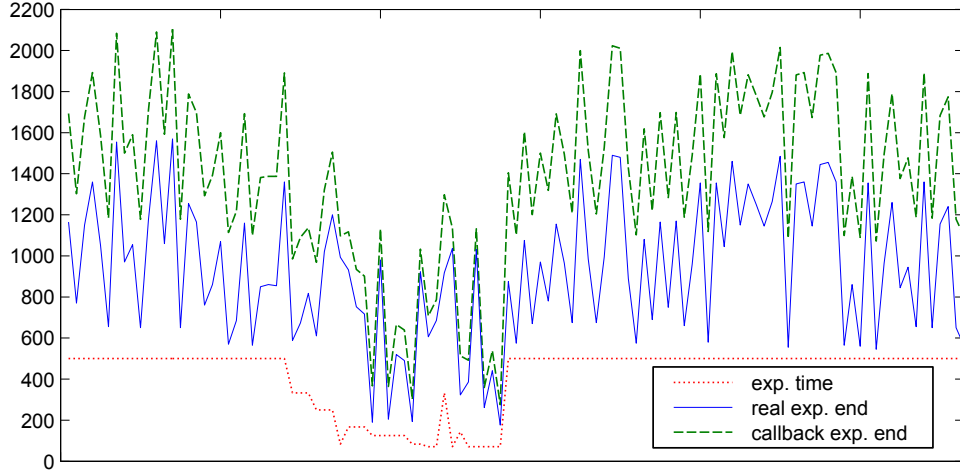


Figure 6.3: *Dependency of the shutter callback time, the time of the actual exposure end and the exposure period. Vertical axis shows times for all experiment trials in milliseconds.*

distinct dependency between the two moments mentioned above and the exposure time.

As it turned out, the notification callback is received almost exactly the exposure period after the end of the real exposure. A graph of the difference with the exposure time compensation included is shown in figure 6.4. Its mean value is 36.7 ms.

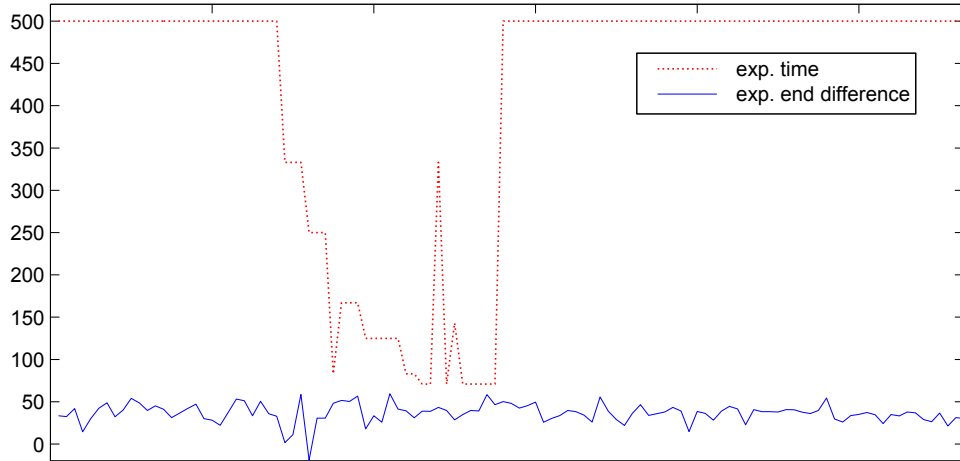


Figure 6.4: *Time difference in milliseconds of the shutter callback time and the actual exposure end time prolonged by the exposure period.*

The practically constant difference in effect means that it could be used in place of the log timestamp method described in section 5.9. It was however discovered in a late phase of the work, so it is not covered in the implementation.

Anyway, the nature of the observed dependency is not clear and it cannot be expected to appear identically in other mobile phone models, so the portability issue we talked about in section 5.14 probably couldn't be solved by it. At least this is another possible option for different devices with no means for synchronization using the log.

6.2 Noise measurements

To estimate the SNR for use in the Wiener filter (section 5.7), we tried to find a correspondence between ISO speed set in the camera and an amount of noise, which can be estimated for our purposes as a variance of a uniform image.

We carried out the measurement by photographing a white surface with no details illuminated by varying intensity of light to incur different ISO value settings. Variance was measured only in the central part of image in greyscale, because the image sensor displays significant low-frequency brightness irregularity towards edges. The resulting graph of 52 test cases is shown in figure 6.5.

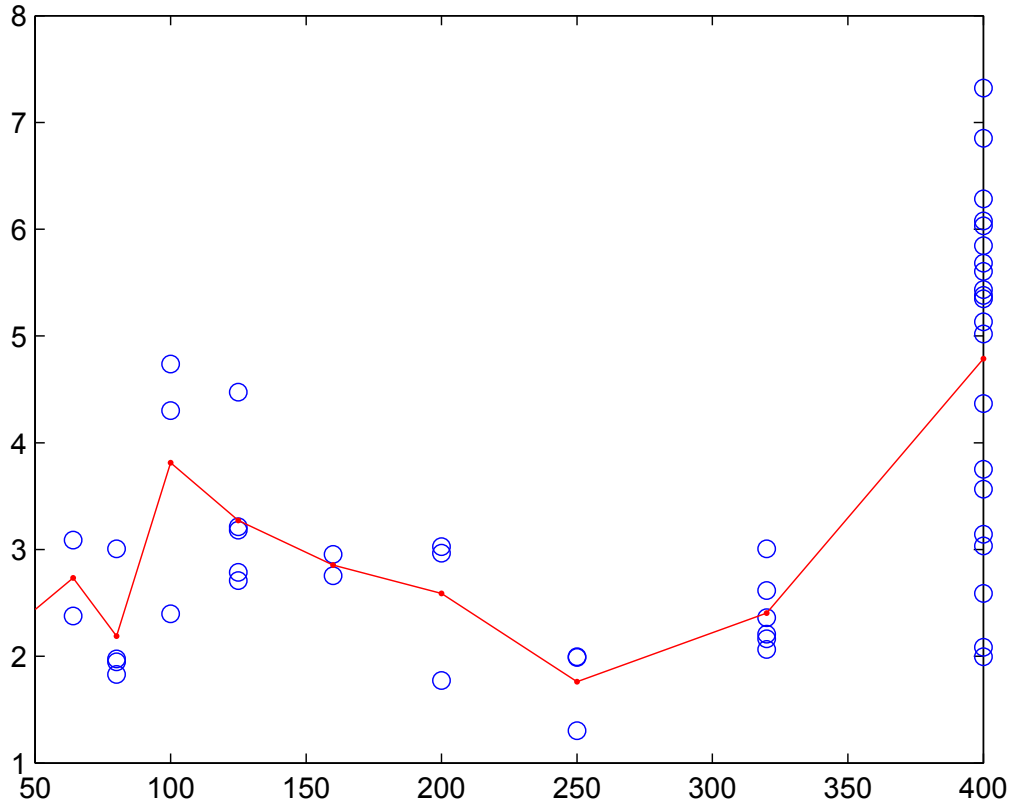


Figure 6.5: *Dependence of noise amount on ISO value. The noise value is measured as a variance of pixel value of a uniform image. The line connects mean values for each ISO speed.*

The dependence is normally quadratic, but that assumes no modifications of the original image. In our case, no such trend is visible. It rather indicates that the camera firmware is performing an algorithm to suppress the noise and it probably operates more aggressively on images with higher ISO to maintain a steady level of noise in the expense of heavy detail loss. The large dispersion for ISO 400 is caused by brightness variability of the image samples taken in different light conditions, as the darker images contain more noise than the brighter ones.

The experiment showed that no relation of the noise to the ISO setting can be anticipated, so our best estimate will be a constant noise expectation, which is what was used in the implementation (see section 5.7).

6.3 Rolling shutter

As introduced in section 4.1.2, the image acquisition is not done in a single moment with certain types of electronic shutters. To prove and measure that, we have set up the following experiment. An image of square grid of white points displayed on a monitor was photographed with our application, so that we had an exposed image and the motion samples from the gyroscope. Then we counted a set of PSF estimations using a Matlab script for different exposure start times. The traces were matched to the acquired image to estimate the actual time shift between the first and the last row of the image sensor.

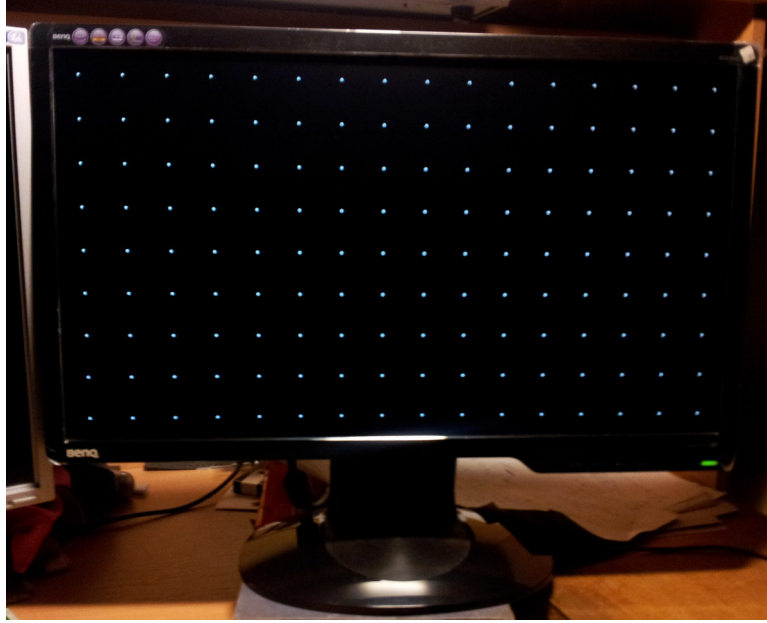
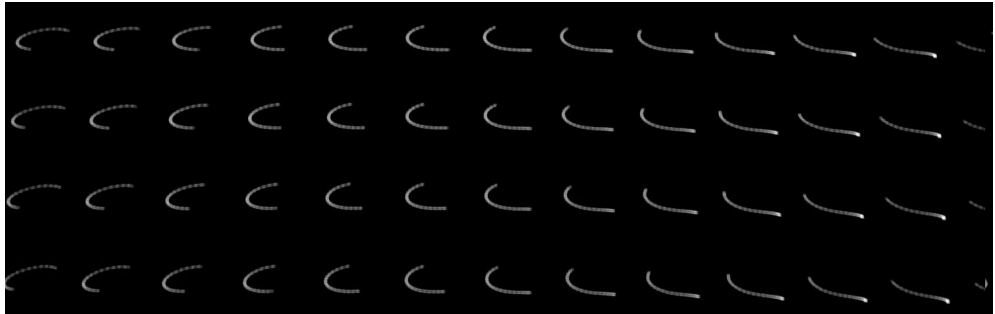


Figure 6.6: *Testing environment for the rolling shutter experiment — grid of points displayed on a LCD.*

The result displayed in figure 6.7 shows an approximate time delay of 60ms, which almost reaches the exposure period. This means that the blur kernels on the opposite vertical sides of the sensor have no common part with exposures faster than $1/17$ s. This makes the PSF variance over the image field very large.

However, to create the blur as big as in the test, a strong forced high-frequency movement excitation was needed, which is almost unthinkable during normal operation. A regular movement would most likely cause a blur of a smaller extent and definitely more straight, because of limited acceleration. In that case, the time shift caused by rolling shutter would hardly form such different blurs in one picture as seen in the experiment.

Even the wildest movement should leave a horizontal band in the central area of a picture invariant with respect to an estimated PSF. The exact location of the band can be selected using the deconvolution preview (see section 5.10), which at last leaves a freedom of choice for the user.



(a) traces of points on LCD



(b) 40 ms (c) 50 ms (d) 60 ms (e) 70 ms (f) 80 ms (g) 90 ms (h) 100 ms

Figure 6.7: A snapshot of point grid displayed on a LCD screen showing the rolling shutter effect. Compared to a series of blur kernels rendered using shifted data from the gyroscope sensor. Exposure $1/14$ s, PSF images were created from data 40–100 ms after camera log timestamp (see section 6.1).

7. Results

In this chapter, we will show illustrations of results achieved by our implemented application (see chapter 5). All results were produced using SNR constant 100 and with normalization turned on.

For comparison, we show here an advanced non-blind iterative method by Shan et al. based on their work [15]¹ and a non-blind version of an algorithm for multi-channel blind deconvolution by Šroubek and Flusser [17]. The latter uses the alternating algorithm mentioned in the section 2.5 with the PSF fixed, so the minimization is executed only with respect to the unknown unblurred image.

We also tested the full blind-deconvolution from the mentioned work of Shan et al.², which had shown some impressive results. However, the result of the first test image shown at figure 7.1f illustrates our experience of total failure of the algorithm when applied to the images taken by our test device. The estimated PSF in figure 7.1g essentially only slightly sharpens the image as the algorithm apparently considers the input image as near maximum for the optimality problem it solves. The reasons are unknown, we can only guess that a slight space variance or the image post-processing block a successful estimation of the correct motion blur.

A thorough review of the results presented here can be found in the next chapter.

¹An executable is available for download at <http://www.cse.cuhk.edu.hk/~leojia/programs/deconvolution/deconvolution.htm>.

²http://www.cse.cuhk.edu.hk/~leojia/projects/robust_deblur

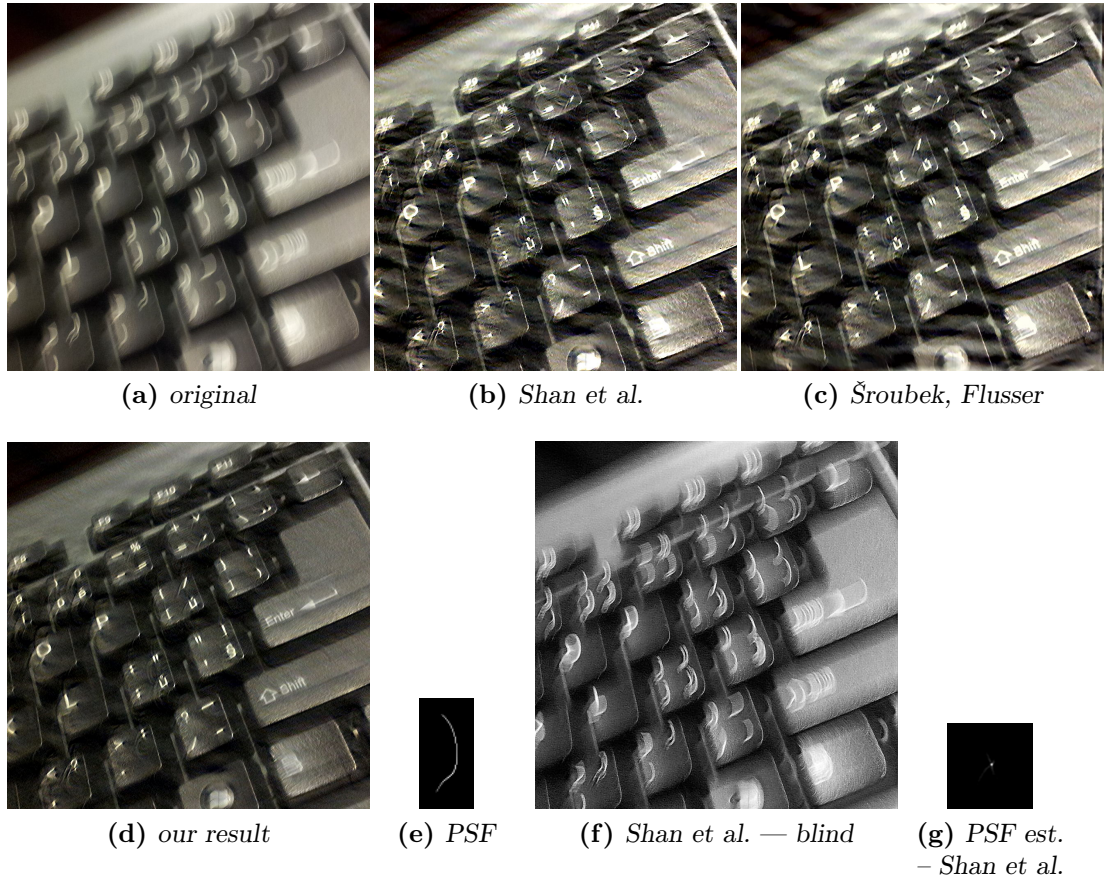


Figure 7.1: Test 1 — 1/7 s exposure, 16×59 estimated PSF.

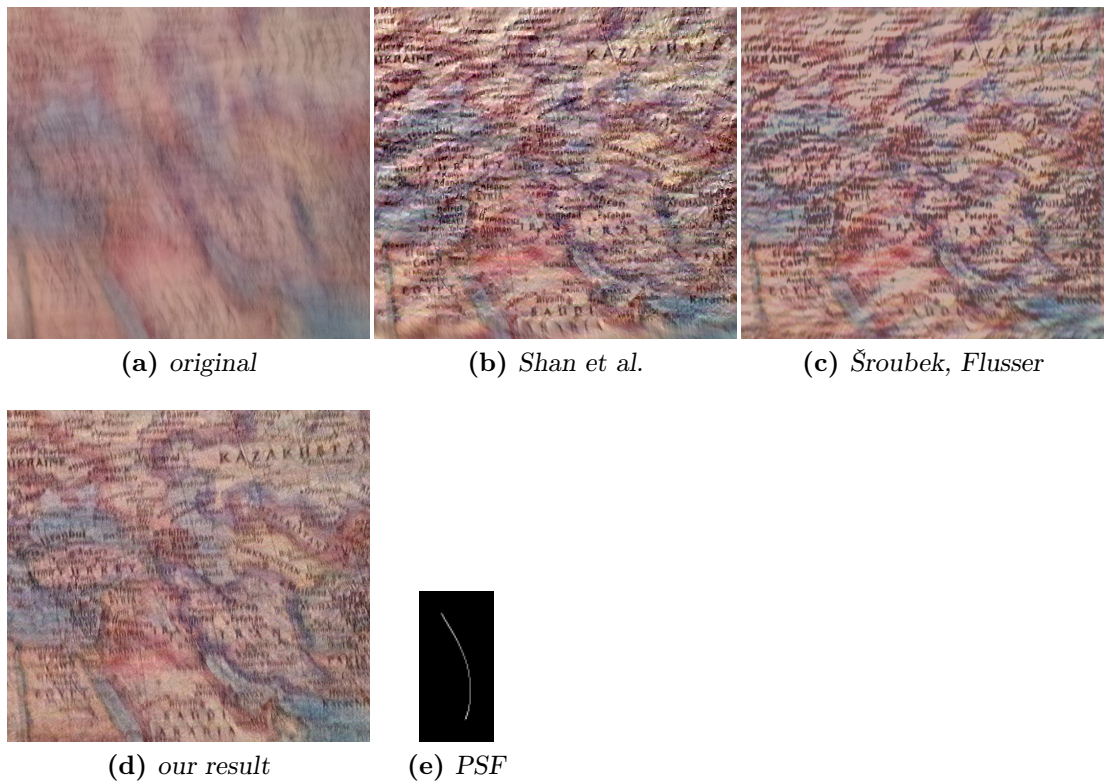


Figure 7.2: Test 2 — 1/6 s exposure, 23×80 estimated PSF.

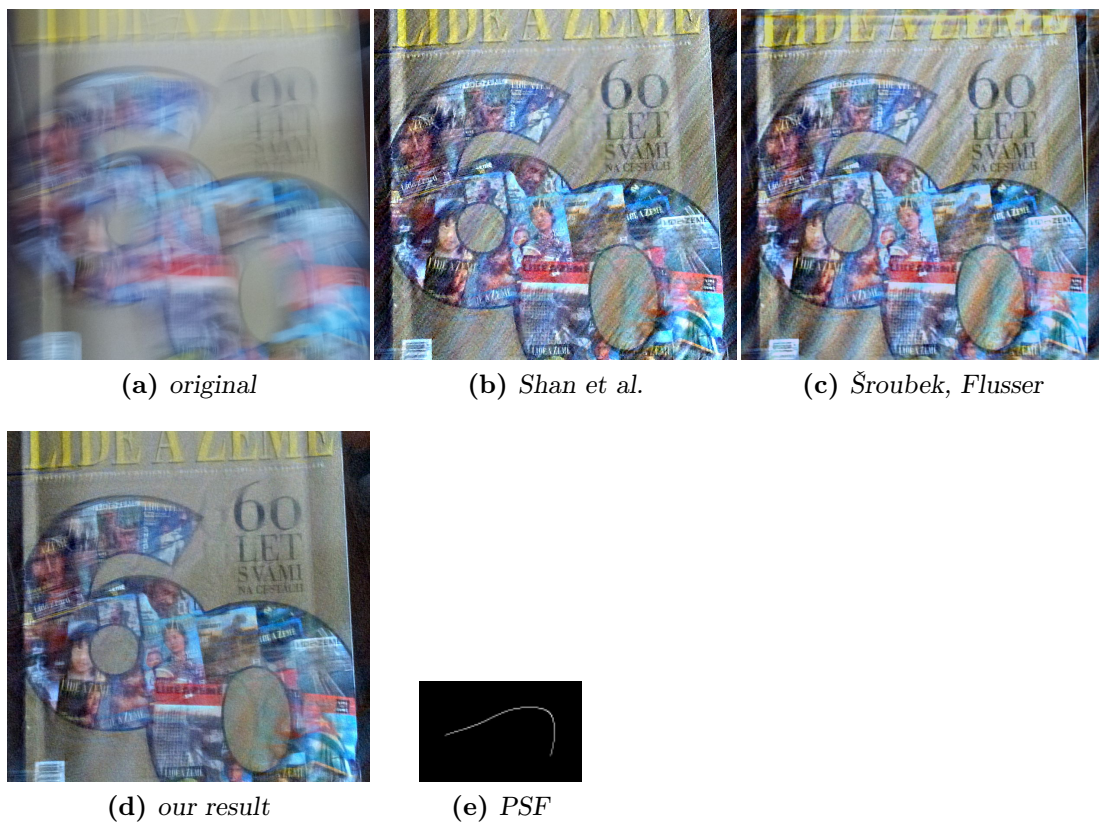


Figure 7.3: Test 3 — $1/5$ s exposure, 83×37 estimated PSF.

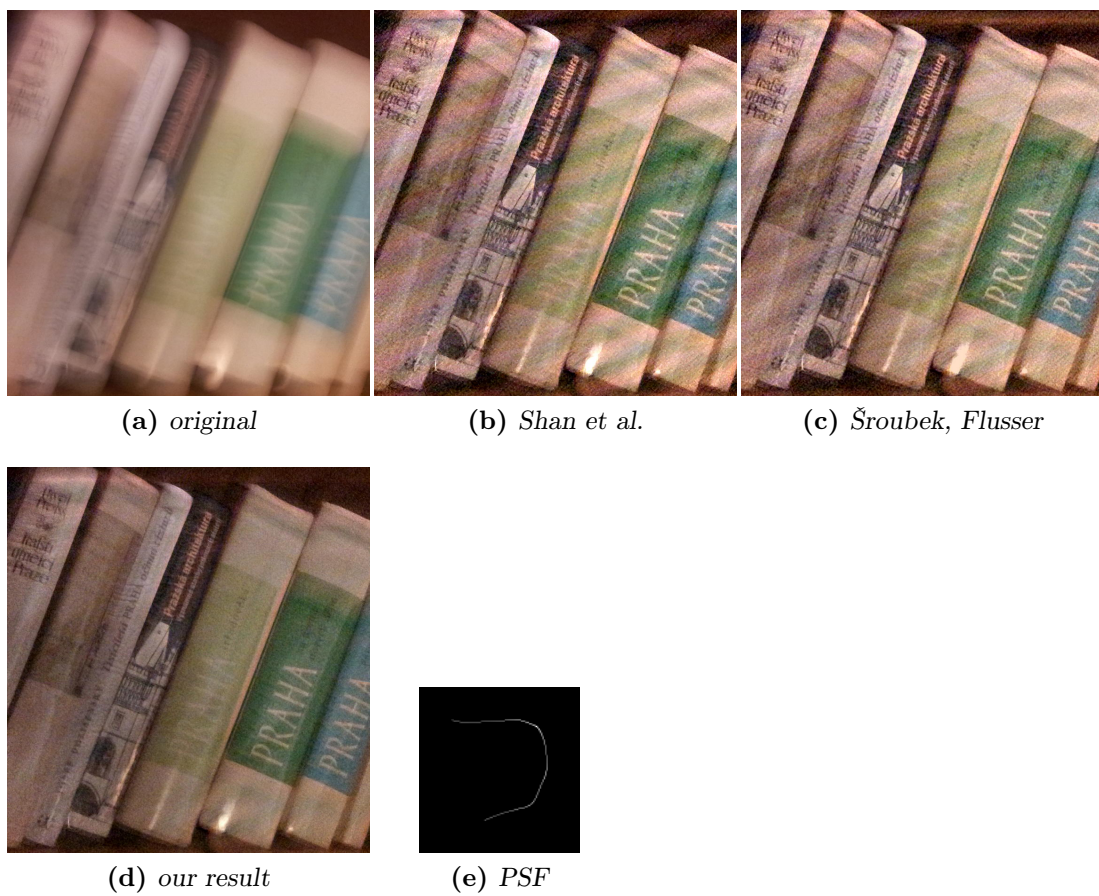


Figure 7.4: Test 4 — $1/2$ s exposure, 72×76 estimated PSF.

8. Discussion

8.1 Evaluation of the results

Illustrations of our implemented deconvolution process shown in previous chapter prove that in spite of a relatively simple approach incorporating the Wiener filter with a spatial-invariant PSF estimation, it is capable of producing convincing images exposing many details that were hidden in the original.

The results are comparable to other more advanced non-blind methods that used the same deconvolution kernel. In some respects our results even surpass the other two compared methods — it arguably contains at least as much of fine details with not too much pronounced noise at the same time. Both the algorithm by Šroubek and Flusser and that of Shan et al. exhibit similar tendency of over-amplifying the signal which rather promotes the noise and false edges than gain signal improvements. Conversely, the high frequency details are more suppressed there probably due to being treated as noise, despite of careful attempts to tune the parameters of noise level expectation. Within our testing environment, the simplified Wiener filter is more advantageous as it filters all frequencies evenly which apparently matches the real spectrum characteristics of most of the tested images more closely.

Our results seem to lack contrast compared to the other outputs, which is largely because of the normalization turned on. On the other hand, it helps retaining the full dynamic range without burnouts as clearly seen for instance in the comparison figure 7.1.

A clear advantage resides also in the matter of computational demands. Because our algorithm uses a direct filter, it is with around 6 seconds of total time on the smartphone (see 5.6) much faster than the other iterative methods used for comparison — both needed 1–2 minutes for deconvolution of a cropped image on a desktop computer.

Unfortunately, we cannot make a worthy comparison of our results to blind deconvolution methods, as the Shan et al. algorithm completely failed with all test images tried as figure 7.1g illustrates. Anyway it shows a kind of (probably implicit) disposition towards certain preconditioned characteristics of input images. It would be interesting to see if other blind method, such as [19], demonstrated a similar outcome.

8.2 Problems

Our deconvolution process admittedly has downsides, as well. Focusing in dark environment may be unsuccessful and then the deconvolved result cannot be sharp even if the PSF estimation is correct.

The subjective quality of the deconvolution output is not entirely consistent, too. The images shown in chapter 7 have indeed been chosen from the best-looking results produced. Outputs with a similar improvement qualities are developed most of the time, but quite frequently the image is impaired by visual anomalies worsening its appearance. Most often it is manifested as ringing artifacts surrounding sharp edges in the picture.

We can only speculate about the roots of these errors, because we have no means of ground truth verification of the PSF estimate other than simple visual comparison with traces of point lights present in the photographed scene — that usually gives the impression of a perfect correspondence although this is of a subjective value. Also, trying to estimate the correct PSF using blind deconvolution was not successful because of the problems described above.

An error in the deconvolution algorithm was ruled out, as the same procedure was later repeated using a Matlab script and the input data generated in the mobile phone with identical results. Other non-blind deconvolution methods were then tried with the same badly conditioned input data which produced similar or even worse results than the simple in-device Wiener filter.

In principle, there are two possible sources of errors in deconvolution: either incorrectly estimated blur kernel or an error of the convolution process model as such.

The first error class has already been discussed earlier and also compensated for in our implementation. The PSF can be incorrect if the motion data from the sensors are not corresponding to the recorded image, i.e., the recording of both input sources is not precisely synchronized (this is handled according to section 5.9). The gyroscope might be giving wrong data because of a systematic error (see section 5.12), noisy signal or possibly an insufficient sampling rate for accurate PSF estimation. The latter is likely not a major problem due to the physical conditions of hand-holding a mobile phone, which is not susceptible to shake frequency much over 10 Hz.

The deconvolution (in the simple case) assumes a model situation with certain conditions, whichever of them is violated, the deconvolution will not be working as expected. The main problem is a space variance of the blurring function as summarized in section 3.2.3. First of all, the photographed scene is supposed to be static with no light intensity changes. Note that even artificial light powered by alternating current breaks this precondition, but that causes only a small error (this can be seen for instance in figure 5.6). Next, only a simple rotation of the camera around image plane axes (x and y) is considered by the model. Translation is permitted only if all objects in the observed scene lie at virtually infinite distance from the camera. A simplified projection model is assumed, which is approximately valid only in a central part of the image sensor.

Finally, image acquisition supposes a linear response to light evenly over the whole image, where the value is counted by integrating the amount of incident light intensity (illuminance) over the time of exposure at a specified spot on the sensor. Several imperfection of the imaging system violate these restrictions, such as various lens aberrations (probably much less important than the lens projection approximation problem) and the rolling shutter (more in section 4.1.2). The linear exposure assumes no later modifications of the image, which is broken at least by the JPEG compression, but probably with other adjustments: denoising algorithm and contrast alterations, such as gamma correction.

Image post-processing might present a serious problem for the deconvolution. Furthermore, different processing depending on circumstances (light conditions, dominating color) is possible and even likely. Denoising algorithm probably chooses different mode of operation depending on set ISO speed as revealed in the noise experiment (section 6.2). Automatic white balance changes ratios between

primary color intensities according to a dominant color, which might not be linear, too.

It has been experienced during the testing phase that certain working conditions are more suitable for successful deconvolution than others. Somewhat counterintuitively, darker environments and thus longer exposure times seem to work better for our application. This may be only an illusion because of more distinct improvement on the original image, but it might also be caused by the mentioned dynamic modifications of the image.

One infringement of space-invariant PSF can be at least partly eliminated using deconvolution preview implemented in our application (see section 5.10). Apart from its original purpose to shift an inaccurate estimation of the camera-gyroscope delay, the rolling shutter effect may be treated by that. The rolling shutter mechanism effectively narrows the domain of a given PSF validity to a horizontal band on the image sensor. The deconvolution preview theoretically allows to choose the position of the band on the image and thus select the area which will be deconvolved correctly. If a central part of the image is chosen, this should also help eliminate the z -rotation and projection error.

In summary, there are numerous possible contributions to making the deconvolution imprecise or even totally wrong. The testing trials gave an impression, that the main causes are the images post-processing and camera translation for near-object photography, but the true causes are not determinable without additional elaborate experiments.

8.3 Future work

There are surely possibilities of improvement in this project. One thing worth researching would be incorporating the translation movement to the model. As the data are available from the accelerometer sensor, it can be used for estimating a modified PSF that would consider a scene of a fixed depth at least.

Given the accessible interactivity on a mobile device and relatively fast computing capabilities, the functionality of the deconvolution preview we introduced could be extended for user-driven setting of various other parameters. One of them could be the depth of the scene mentioned above, others could be the SNR constant, normalization amount or compensations for spatial-variant PSF at different places on the sensor. It could be also dynamically switched between different combinations of functions for instance. Given small enough preview bars, the deconvolution could be possibly made (almost) real-time, which would allow for even more interactive user input.

Concerning the full space-variant PSF, one could approximate the space-variant deconvolution by dividing the image sensor to smaller rectangular parts (with possible overlaps), which would be deconvolved piecewise using space-invariant PSFs counted for each rectangle and then fused to the final image.

To deal with the quality inconsistency described earlier in this chapter, thorough experiments could be arranged to clarify the reasons for the errors observed. It would be difficult and with unreliable vision of success because of the lack of camera control available, but it would certainly be worth the try.

In future, the portability issues stated in section 5.14 should be definitely addressed. The biggest challenge in porting the application to another device will

be the camera-gyroscope synchronization problem, which could be attempted to solve using the logging approach (section 5.9) or possibly analyzing the camera timing relations as we did in section 6.1. A possible generalization given enough tested devices would enable spreading the application among the Android community; Google Play service could be used to distribute the application.

8.4 Related work

The only existing work using motion sensors for image deblurring that we are aware of, is the work of Joshi et al. [7]. However, they used a special attachment for a normal consumer camera that measures rotation and translation. The space-variant blur with constant depth is assumed. The deblurring algorithm is using the motion data in combination with the blurred image to find the best estimate of the motion. That is done to overcome errors of accelerometer measurements. The deconvolution is performed with a custom algorithm using the MAP technique described in section 2.3.

The method shows some successful results, which however comes with some inconvenient translation restrictions due to the space-variant nature of the solved problem. The initial translation velocity must be zero, the rotation in relation to gravity must be always detectable from the accelerometers and the translation amount during the exposure must be within a certain limit. The disadvantage of this work is also the computational demands: the time needed only for the search of corrected PSF is 5 minutes according to the paper.

Our work in comparison uses much simpler model which handles no translation movement, but intensive translations would be probably not handled very well by the algorithm of Joshi et al., either. Our application is unique in the interactive abilities and a potential for portability along mobile device platform, which are the biggest advantages over the above work.

Conclusion

The problem of image blurring in situations with lack of light has been lasting since camera introduction. In recent years it has come also to the world of mobile devices equipped with camera. However, with the spread of high-performance computing, it is becoming possible to make up for mediocre optical capabilities with the use of digital image processing.

In this thesis, we have proposed a novel system for photograph deblurring on a mobile device using its embedded motions sensors. It has been implemented as an application for devices running Android OS and a successful operation has been achieved on a test smartphone device. It uses the familiar Wiener deconvolution method that is working in the frequency domain of the image.

The application was written user-friendly with efforts to optimize its interactive usability.

Several experiments have been arranged and executed to discover certain unknown attributes of the phone camera, such as timing conditions of its operation and noise handling. These data were then used for tuning the quality of the outputs.

The application is capable of clear improvements of originally blurred photographs as shown in sample images. It is also operating very fast owing to the powerful CPU in the phone and optimized application performance.

Due to the hardware limitations of the device, the application is not expected to be portable to a different Android device in the current state. However, an extension should not be complicated given the hardware support.

Bibliography

- [1] M.R. Banham and A.K. Katsaggelos. Digital image restoration. *Signal Processing Magazine, IEEE*, 14(2):24–41, 1997.
- [2] T. E. Bishop, S. D. Babacan, B. Amizic, A. K. Katsaggelos, T. Chan, and R. Molina. In P. Campisi and K. Egiazarian, editors, *Blind image deconvolution: theory and applications*, chapter 1. CRC press, 2007.
- [3] J.E. Bresenham. Algorithm for computer control of a digital plotter. *IBM Systems journal*, 4(1):25–30, 1965.
- [4] R. Fergus, B. Singh, A. Hertzmann, S.T. Roweis, and W.T. Freeman. Removing camera shake from a single photograph. In *ACM Transactions on Graphics*, volume 25, pages 787–794. ACM, 2006.
- [5] M. Frigo and S.G. Johnson. The design and implementation of FFTW3. *Proceedings of the IEEE*, 93(2):216–231, 2005.
- [6] R. I. Hartley and A. Zisserman. *Multiple View Geometry in Computer Vision*. Cambridge University Press, ISBN: 0521540518, second edition, 2004.
- [7] N. Joshi, S.B. Kang, C.L. Zitnick, and R. Szeliski. Image deblurring using inertial measurement sensors. *ACM Transactions on Graphics (TOG)*, 29(4):30, 2010.
- [8] L.B. Lucy. An iterative technique for the rectification of observed distributions. *The astronomical journal*, 79:745–754, 1974.
- [9] J.G. Nagy and D.P. O’Leary. Restoring images degraded by spatially variant blur. *SIAM Journal on Scientific Computing*, 19:1063, 1998.
- [10] SK Nayar and M. Ben-Ezra. Motion-based motion deblurring. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 26(6):689–698, 2004.
- [11] M. Šorel and J. Flusser. Space-variant restoration of images degraded by camera motion blur. *IEEE Transactions on Image Processing*, 17(2):105–116, 2008.
- [12] W.H. Richardson. Bayesian-based iterative method of image restoration. *JOSA*, 62(1):55–59, 1972.
- [13] F. Rooms, B. Goossens, A. Pižurica, and W. Philips. Image restoration and applications in biomedical processing. In *Optical and Digital Image Processing*, pages 571–591. Wiley-VCH Verlag GmbH & Co. KGaA, 2011.
- [14] F. Šroubek and M. Šorel. Blind deconvolution imaging. In *Optical and Digital Image Processing*, pages 529–548. Wiley-VCH Verlag GmbH & Co. KGaA, 2011.
- [15] Q. Shan, J. Jia, and A. Agarwala. High-quality motion deblurring from a single image. *ACM Transactions on Graphics*, 27(3), 2008.

- [16] M. Šorel and F. Šroubek. Space-variant deblurring using one blurred and one underexposed image. In *Proceedings of the 16th IEEE international conference on Image processing*, pages 157–160. IEEE Press, 2009.
- [17] F. Šroubek and J. Flusser. Multichannel blind deconvolution of spatially misaligned images. *IEEE Transactions on Image Processing*, 14(7), 2005.
- [18] G.M.P. van Kempen, L.J. van Vliet, and P.J. Verveer. Application of image restoration methods for confocal fluorescence microscopy. In *Proc. SPIE*, volume 2984, pages 114–124, 1997.
- [19] C. Wang, L. Sun, P. Cui, J. Zhang, and SQ Yang. Analyzing image deblurring through three paradigms. *IEEE Transactions on Image Processing*, (99):1–1, 2012.
- [20] L. Xu and J. Jia. Two-phase kernel estimation for robust motion deblurring. *Computer Vision–ECCV 2010*, pages 157–170, 2010.

List of Abbreviations

API	application programming interface
BD	blind deconvolution
CPU	central processing unit
DFT	discrete Fourier transform
FFT	fast Fourier transform
FPU	floating point unit
GUI	graphical user interface
IDE	integrated development environment
JNI	Java Native Interface
NDK	native development kit
OS	operating system
PSF	point spread function
SIMD	single instruction, multiple data
SNR	signal-to-noise ratio

Appendix A - Attachments

CD with the following contents:

- this thesis in PDF format
- GyroStab application for Android
 - ready-to-install APK application
 - source codes for the application in the form of Eclipse project
- sample result images

Appendix B - User manual

On the CD attached to the thesis you can find the application for Android.

Installation and basic usage

The application package **GyroStab.apk** can be installed in the standard Android manner by opening the package file on the mobile device.

The installed application opens directly in the photographing mode. A preview image from the camera is shown on display. To take picture, simply touch the screen. The camera will try to autofocus and then acquire a picture.

The taken picture (possibly blurred) will be processed by the device and the resulting image will be then saved and displayed on the screen using the default image viewer. To shoot another pictures, just exit the viewer.

Deconvolution preview

If the deconvolution preview is switched on (see section Settings), a grid of smaller preview versions of the result in black-and-white will be first computed and shown on the display, so that you can choose the optimal set of parameters. Initially, only a detail of the preview pictures are shown, but you can scroll the pictures to compare other parts of the image. Zooming is also possible using the pinch gesture. To choose a version of the picture, simply click the rectangle containing the selected picture. Then the picture is processed completely, saved and displayed as without the preview turned on.

Resulting images are saved to the default camera storage directory, which is usually `/sdcard/DCIM/Camera`. The results are isolated in separate directories to keep the related files together. The directories are named according to current date and time in the form `YYYYMMDDhhmmss` (year, month, day, hour, minute, second) to ensure unique names. The original image is named `<timestamp>.jpg` and the resulting image `<timestamp>.jpg_out.jpg`.

Settings

To adjust various preferences of the application, click the menu button on the main screen and then select the settings menu item. You can change parameters of the deblurring process, such as SNR constant, default exposure start delay, resolution of the output image; you can switch on or off the deconvolution preview, number of preview pictures shown and variance of the delay for the preview. Normalization of the output can also be set, as well as saving additional data with the output (recorded gyroscope samples, computed PSF or the PSF track drawn on top of the image). Gyroscope can be also calibrated here, which is highly recommended.