Fast Blur Removal for Wearable QR Code Scanners (supplemental material)

Gábor Sörös, Stephan Semmler, Luc Humair, Otmar Hilliges
Department of Computer Science
ETH Zurich
{gabor.soros|otmar.hilliges}@inf.ethz.ch, {semmlers|humairl}@student.ethz.ch

DERIVATION OF THE ENERGY FUNCTION FOR BLIND DECONVOLUTION

The uniform blur model is formulated as

$$b = k * l + n$$

where the blurred image b is a result of convolving a sharp image l with a blur kernel k and adding Gaussian noise n. In blind deconvolution, we know only the blurred image b and we try to recover the latent sharp image l. This also requires estimating the blur kernel k. The problem of finding both l and k can be formulated as minimizing the following energy function:

$$\arg\min_{l,k} ||b - k * l||_2^2 + \lambda\rho_l(l) + \gamma\rho_k(k)$$

To derive this energy function, we first express the noise on a single pixel. In the equations, the index i runs over all image pixels from 1 to N.

$$n_i = b_i - (k * l)_i$$

The probability distribution of a single pixel's additive noise is assumed to be Gaussian with zero mean and standard deviation σ:

$$p(n_i) \sim N(0, \sigma) = \frac{1}{\sqrt{2\pi}\sigma} \exp(-\frac{1}{2\sigma^2} n_i^2) \propto e^{-\frac{1}{2\sigma^2} n_i^2}$$

Assuming that the noise on each pixel is independent and identically distributed (i.i.d.), the probability of the image noise is the product of the pixel noise probabilities:

$$p(n) \propto \prod_{i=1}^{N} e^{-\frac{1}{2\sigma^2} n_i^2} = e^{-\frac{1}{2\sigma^2} \sum_{i=1}^{N} n_i^2}$$

Now let us look at the probabilities of l and k. As Bayes' rule says, the posterior probability is proportional to the product of the likelihood and the prior:

$$p(x|y) = \frac{p(y|x)p(x)}{p(y)} \propto p(y|x)p(x)$$

The term $p(y)$ is a normalization factor which does not play a role in the further minimization, as it will reduce to an additive constant after taking the logarithm.

Introducing independent variables l and k hence results in

$$p(l, k|b) \propto p(b|l, k)p(l, k) = p(b|l, k)p(l)p(k)$$

The maximum a posteriori (MAP) estimates of the unknowns k and l are

$$\begin{aligned}\arg\max_{l,k} p(l, k|b) &= \arg\min_{l,k} [-\log\ p(l, k|b)] \\ &= \arg\min_{l,k} [-\log\ p(b|l, k) - \log\ p(l) - \log\ p(k)]\end{aligned}$$

as maximizing the posterior probability is equivalent to minimizing its negative logarithm.

Likelihood term

The likelihood term follows from our blur model by expressing the noise term:

$$p(b|l, k) = p(n) \propto e^{-\frac{1}{2\sigma^2} \sum_{i=1}^{N} n_i^2}$$

$$-\log\ p(b|l, k) \propto C \sum_{i=1}^{N} [n_i]^2 = C \sum_{i=1}^{N} [b_i - (k * l)_i]^2 \propto ||b - k * l||_2^2$$

with C containing all the constant terms and i indexing the pixels in the image.

Image prior

While the pixel values can be very different across images, the (log-)distribution of the image derivatives follows a common pattern (see Figure 1) in photographs. This property has been successfully exploited in the solutions of various image processing problems. This distribution is independent of the image scale and has a heavy tail which means while most gradients are around zero (flat image areas), some large gradients are also likely (edges). In particular, the distribution is *not* Gaussian. This is very unfortunate because a Gaussian prior would make the minimization problem very simple with a closed-form solution. Images restored with a Gaussian prior are often oversmoothed and/or contain ringing artifacts. Instead, the distribution is usually modeled by a Hyper-Laplacian function (see Figure 2) with an exponent $\alpha < 1$, best values are $\alpha \in [0.5, 0.8]$.

Let us denote the x- and y-derivatives of the image l at pixel i as $\partial_x l_i$ and $\partial_y l_i$, respectively. The prior $p(l)$ can then be formulated as

$$p(l) \propto e^{-\frac{1}{2\eta^2} \sum_i^N |\partial_x l_i|^\alpha + |\partial_y l_i|^\alpha}$$

In general form:

$$p(l) = \prod_{i=1}^{L} \Phi(\nabla l_i) = \prod_{i=1}^{L} e^{-\phi(\nabla l_i)}$$

and after taking the logarithm:

$$-\log\ p(l) = \sum_{i=1}^{L} \phi(\nabla l_i)$$

The two most commonly used curves are the **Gaussian prior**:

$$-\log\ p(l) = \sum_{i=1}^{L} |\nabla l_i|^2 = ||\nabla l||_2^2$$

and the **hyper-Laplacian prior**:

$$-\log\ p(l) = \sum_{i=1}^{L} |\nabla l_i|^\alpha$$

Other parametric curves are also used in the literature.

In our algorithm, we apply the Laplacian prior ($\alpha = 1$) because it matches well the black and white code images and fast solution methods exists for minimizing the energy function:

$$\rho_l(l) = \sum_{i=1}^{L} |\nabla l_i|$$

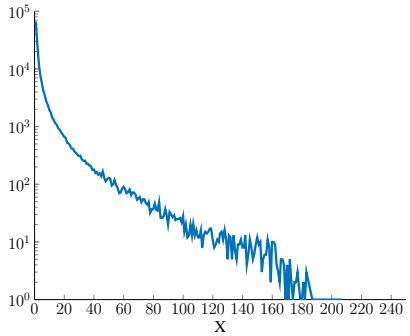


Figure 1: A natural image prior: log-gradient histogram of an image. The shape of this curve can be approximated by various parametric models.

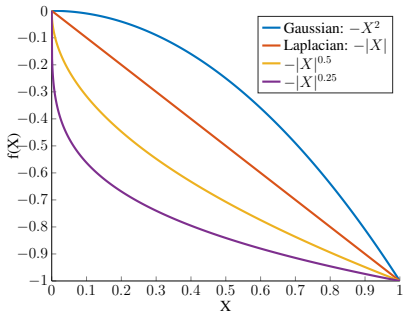


Figure 2: Parametric models as natural image priors.

Kernel prior

In the simplest case the kernel prior $p(k)$ is assumed **uniform** so it is ignored.

Earlier methods applied a **sum-of-exponentials** prior: For a single pixel k_i of the kernel k the distribution is a sum of D exponential distributions:

$$p(k_i) \sim \sum_{d=1}^{D} w_d \varepsilon_{\lambda_d}(k) =$$

$$= w_1\lambda_1 e^{-\lambda_1 k} + w_2\lambda_2 e^{-\lambda_2 k} + w_3\lambda_3 e^{-\lambda_3 k} + \dots$$

For the whole kernel then:

$$p(k) \sim \prod_{k=1}^{K} \sum_{d=1}^{D} w_d \varepsilon_{\lambda_d}(k)$$

Its negative logarithm is:

$$-\log\ p(k) = -\sum_{k=1}^{K}\ \log \sum_{d=1}^{D} w_d \varepsilon_{\lambda_d}(k) =$$

$$= -\sum_{k=1}^{K}\ \log[w_1\lambda_1 e^{-\lambda_1 k} + w_2\lambda_2 e^{-\lambda_2 k} + w_3\lambda_3 e^{-\lambda_3 k} + \dots\ ]$$

Alternatively, a **Gaussian gradient prior** can be applied, this prior enforces connectedness.

$$p(k) = \prod_{k=1}^{K} e^{-c(\Delta k)^2}$$

$$-\log\ p(k) = C \sum_{i=1}^{K} (\Delta k)^2 = ||\Delta k||_2^2$$

In most methods and also in our method, a **Gaussian intensity prior** is applied on the kernel, this prior enforces small values and avoids Dirac kernels.

$$p(k) = \prod_{i=1}^{K} e^{-k_i^2} = e^{-\sum_{i=1}^{K} k_i^2}$$

$$-\log\ p(k) = \sum_{i=1}^{K} k_i^2 = ||k||_2^2$$

$$\rho_k(k) = ||k||_2^2$$

Also, $||k||_1 = \sum_{i=1}^{K} k_i = 1$ is always necessary so that the blurring does not change the overall image intensity.

FIGURES OF THE PAPER IN HIGHER RESOLUTION

Removing synthetic blur

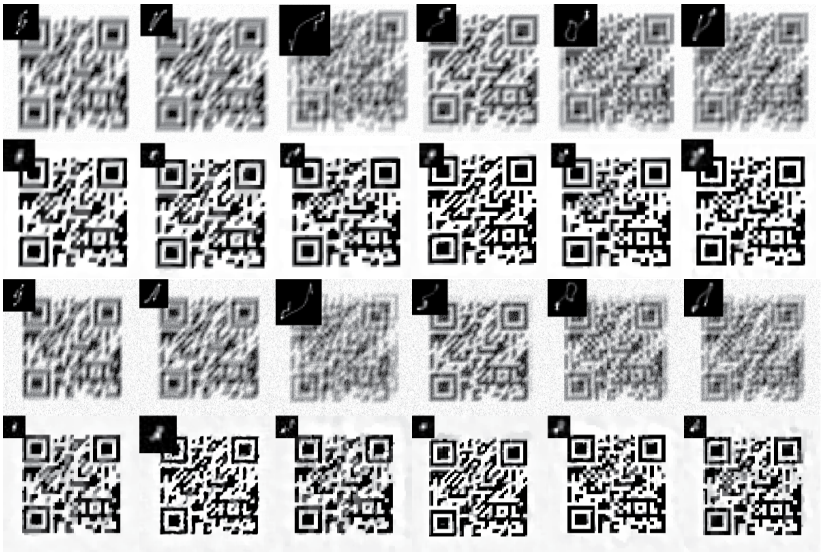
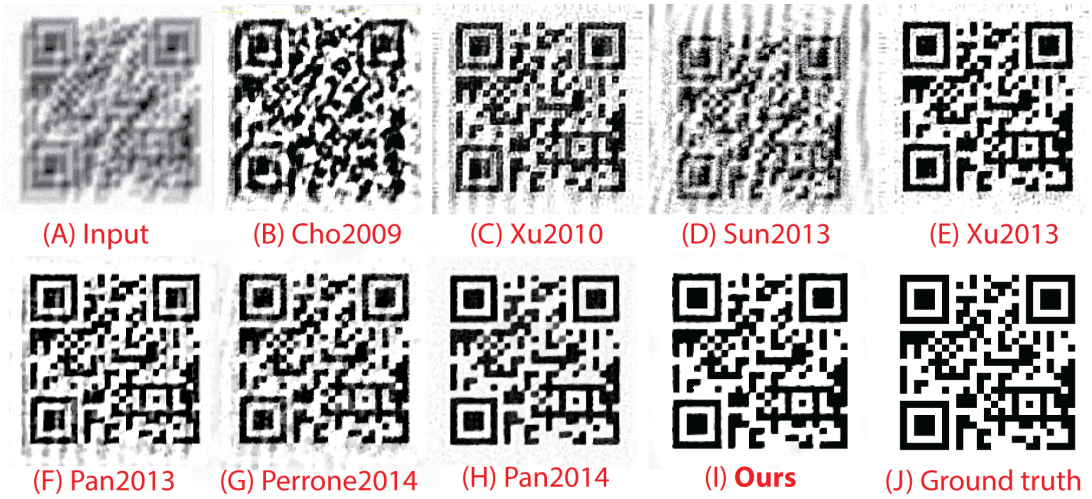


Figure 5: Top two rows: Removing synthetic blurs from images (0.1% synthetic noise). Bottom two rows: Removing synthetic blurs from images (also added 1% synthetic noise). The odd rows show the input images and ground truth kernels. The even rows show the output images and kernel estimates when first decoded. Kernel size indicates the scale level. The kernels are from the blur test set from Levin2009.



	A Input	B Cho2009	C Xu2010	D Sun2013	E Xu2013	F Pan2013	G Perrone2014	H Pan2014	I Ours2015	J Truth
binary	-	C++/GPU	C++/GPU	Matlab	C++/GPU	Matlab	Matlab	Matlab (*C++*)	C++	-
runtime	-	0.481s	0.955s	217.730s	1.049s	133.8s	171.898s	12.736s (*9.691s*)	1.765s (**0.614s**)	-
decoded	-	no	no	no	**yes**	no	**yes**	**yes**	**yes**	-

Figure 6: Comparison of blind deconvolution algorithms on a synthetically blurred QR code.

Removing real motion blur

These figures show more examples from the 83 smartphone images restored by our algorithm (Figure 6 in the paper).

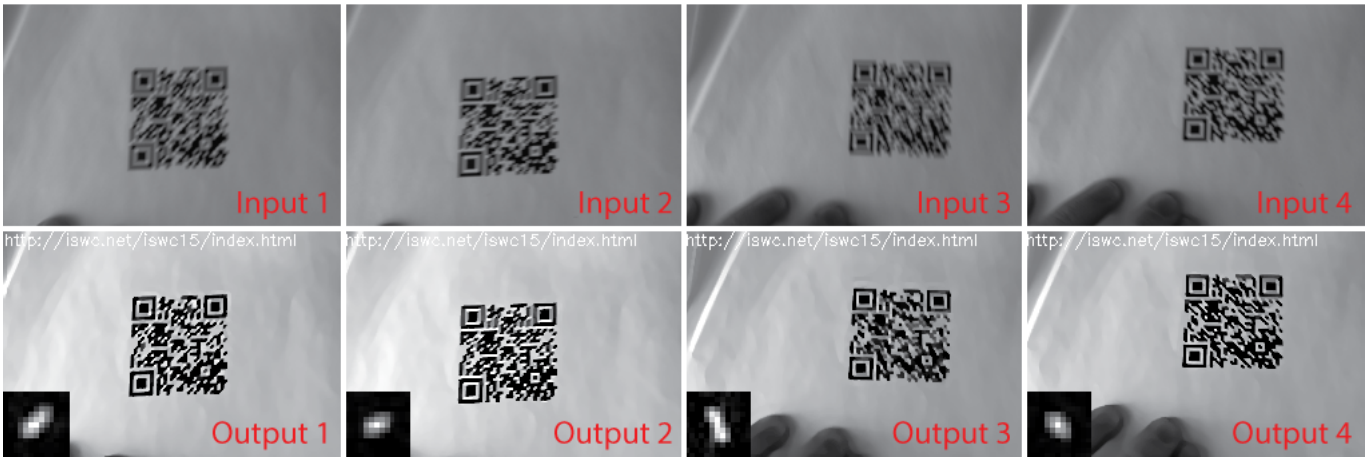


Figure 7: Removing real motion blur from QR code images. The decoded content (ISWC2015 URL) is written in the top of the images.

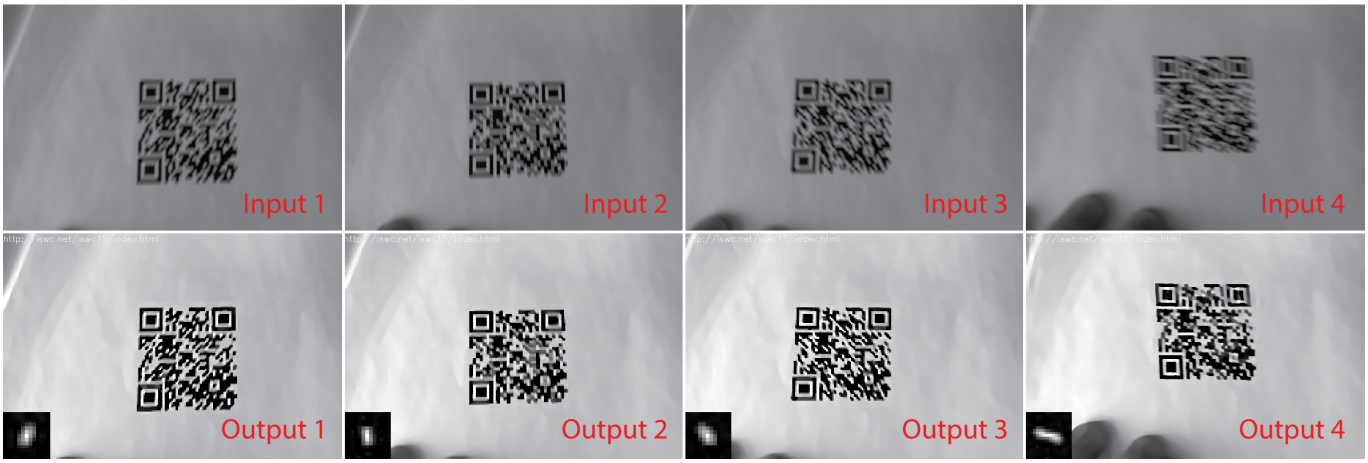


Figure 8: Further results of removing real motion blur from QR code images.

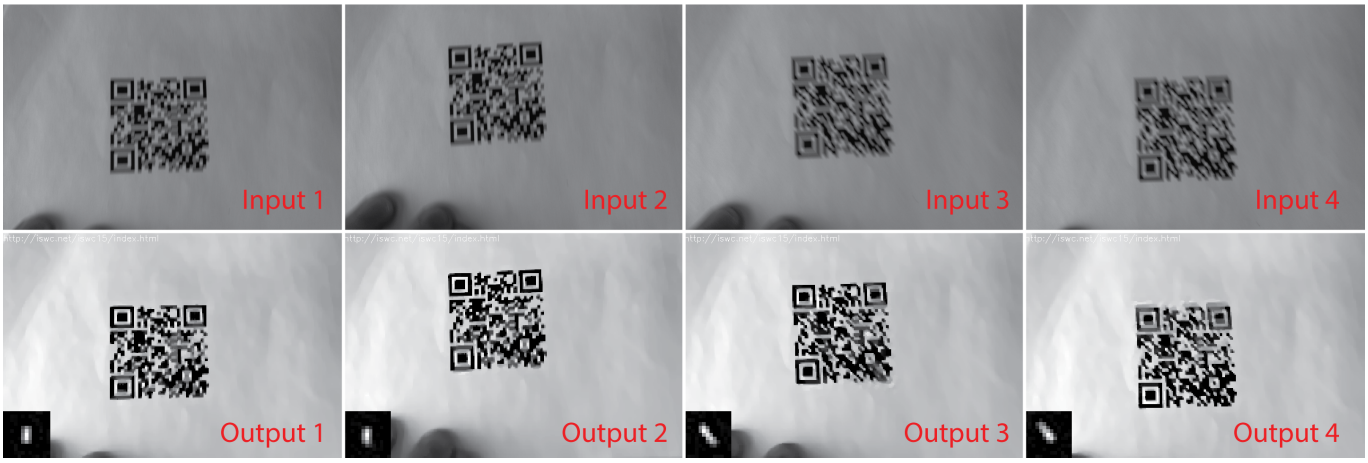


Figure 9: Further results of removing real motion blur from QR code images.

ADDITIONAL EXPERIMENTS

Initial kernel choice

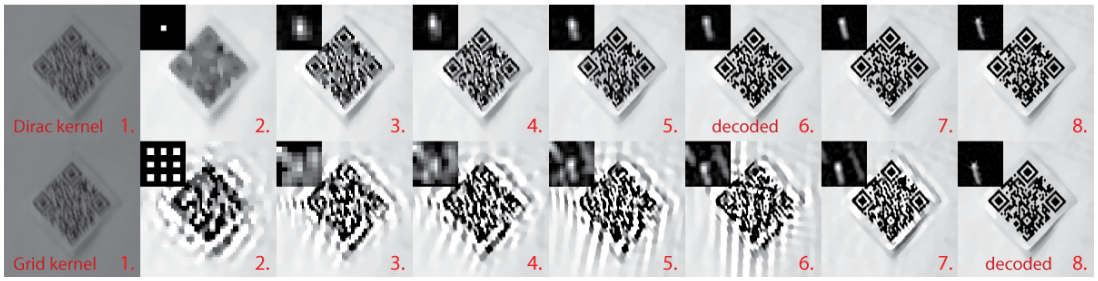


Figure 10: Illustration of image and kernel refinement over 8 iterations using a single peak or a grid of peaks as starting kernel. In this example, both initial kernels lead to a correct solution, but our grid kernel converges slower. However, the grid kernel is able to restore very large blurs that are otherwise unsuccessful with the peak kernel (see next figure). The grid example also illustrates how disconnected kernel noise gets removed during the process. The image is 300×300, the kernel is 33×33 pixels, the decoding took $0.719s$ and $3.107s$, respectively. The blurry image was taken with a smartphone.

Figure 11: Further examples illustrating image and kernel refinement using a single peak or a grid of peaks as starting kernel. In these examples, codes were not recognized using the peak initial kernel, but our grid kernel is successful in removing large blurs.

Defocus blur and upscaling blur

So far, we have focused on decoding motion-blurred codes only, but the QR properties remain the same under other types for blur as well. In our future work, we will investigate the adaptations required in kernel regularization to allow different non-sparse shapes. Here, we briefly show promising preliminary results of our experiments in removing synthetic defocus blur and synthetic upscaling blur.

Removing synthetic defocus blur

In theory, it is possible to restore slightly defocused codes until the blur is smaller than the module size in the code. Figure 12 left shows a synthetically defocused example using a 178×178 image and a 33×33 Gaussian kernel with standard deviation 3. The image is successfully decoded in $0.318s$, however, some gray artifacts are visible at dense black and white areas of the code. Note the Gaussian-shaped estimated kernel.

Reading tiny codes

Blind deconvolution is also an important step in super resolution algorithms where a downsampling filter needs to be estimated and inverted. The close connection in the mathematical models suggest that our algorithm might be suitable for super resolving tiny QR codes. We have performed a simulation to justify this (see Figure 12 right). In a photo editing software we downscaled a QR code with nearest neighbor interpolation so that one symbol corresponds to only one pixel, and upscaled it to 300×300 pixels ($12\times$) again, using bilinear interpolation. The upscaled code is blurry and not readable. Our algorithm is able to restore and read the code after only two iterations. Note the square shape of the estimated blur kernel in the bottom right corner.

Figure 12: Left: removing synthetic defocus blur. Note the Gaussian-shaped estimated kernel. Right: Reconstructing a bilinearly 12x upscaled code. Note the square-shaped estimated kernel.