# Camera Shake Deconvolution using Motion Sensors

Sebastian Sanso
ssanso@andrew.cmu.edu
Carnegie Mellon University
Pittsburgh, Pennsylvania, USA


**Figure 1: Camera deblurring setup with gyroscopes connected to an Arduino. Given a shaky image, the system is able to accurately predict the PSF and mitigate a lot of the motion blur present in the original photograph.**

## ABSTRACT

In this paper, I present a hardware method for deblurring images which are blurred as a result of camera shake. I use 3 gyroscope sensors to track the pitch, yaw and roll of the camera as it moves through an exposure. Using this gyroscope data, I infer an initial blur kernel which is already very accurate. I then use traditional optimization techniques to approach the non-blind convolution problem and solve for the original image. This deconvolution technique has the potential to be extremely accurate given that I track the camera motion directly. It is also computationally efficient, as the initial kernel estimation takes only a few seconds.

## 1 INTRODUCTION

Blurry images are often an undesirable result in photography. Blur often occurs in low-light conditions or in settings where it is necessary to use a low shutter speed. As a result, deblurring an image is a long studied problem which has proven to be generally challenging. Assuming that motion blur is shift-invariant, we can model it as a convolution of the sharp image with a blur kernel. If the blur kernel is known, then we can use non-blind deconvolution techniques to recover the non-blurred image as best we can. However, the blur kernel is not known in a lot of situations, which results in a blind deconvolution problem. Blind image deconvolution is an ill-posed problem, since the blurred image does not inherently have enough information to constrain the solution.

There are many ways to approach blind deconvolution. One of the most common methods is to use local statistics to generate image priors and therefore add more constraints to the problem [1]. The main issue with this approach is that it is computationally expensive and it may take many minutes to process one image. Modern approaches use convolutional neural networks to find a deblurred result. This approach seems to be very effective at the cost of training a potentially large model.

To address all of these issues, my approach introduced a hardware component that gives more information and therefore more constraints on the initial problem. By attaching 3-axis gyroscope sensor to any consumer camera, it is possible to measure the angular velocities along the $X$, $Y$, and $Z$ axes. Most smartphones have built in gyroscopes, so this method is actually very viable when compared to other deconvolution techniques such as adding a flutter shutter [5] or utilizing multiple cameras [4], [6]. My model assumes camera shake is composed mostly of rotations since the focal distance is often much smaller than the scene depth, but it is possible to add a 3-axis accelerometer to deal with translations as well.

Once the sensor data is measured, it can be used to generate an initial blur kernel which will already be fairly accurate. It is possible to improve the kernel even further with gradient descent techniques, but this significantly increases processing time. As my approach is designed to be primarily lightweight, this is avoided to reduce overhead.

## 2 DEBLURRING METHOD

My method can be partitioned into two phases. In the first phase, I read the gyroscope data and use it to generate a blur kernel. After that, I introduce the blurred image, and solve the non-blind convolution using a modified version of the Lucy-Richardson algorithm.

### 2.1 Blur Kernel Estimation

Image blur is often modeled by a convolution, assuming the image blur is spatially invariant. Specifically, we have:

$$B = L \otimes K + N \tag{1}$$

Where $\otimes$ is the convolution operator, $L$ is the latent image, and $N$ is the sensor noise at each pixel.

It is not the case, however, that image blur is always spatially invariant. Blur can be affected by camera translation, roll, pitch, yaw, focal length variation, and defocus. Using only gyroscopes, we can only hope to address roll, pitch, and yaw accurately. If we additionally add a 3-axis accelerometer to our hardware setup, we can hope to address camera translation accurately. However, adding an accelerometer implies that you have to integrate twice to get relative positional values, and double integration introduces a lot of undesirable drift. It is possible to counteract this with a drift compensation method [2], but I decided to avoid this method and assume that most camera shake blur is caused by camera rotation.

If a shaky camera takes K samples as it moves, it will project K points in 3D space to points $(x_k, y_k)$ on a 2D plane. We can introduce homogenous coordinates and this becomes:

$$(x_k, y_k, 1)^T = P_k(X, Y, Z, 1)^T \tag{2}$$

where $P_k$ is the camera projection matrix at sample time $k$. Note that for estimating image blur, it is only necessary to estimate relative positions and not absolute positions.

At each position, we project the sensor data onto the image plane. Therefore, we can decompose this movement into rotation and translation matrices using the following planar homography:

$$(x_k, y_k, 1)^T = [C(R_k + \frac{1}{d}T_k N^T)C^{-1}](x_0, y_0, 1)^T \tag{3}$$

where C is the camera intrinsics matrix, $R_k$ and $T_k$ are the rotation and translation matrices for time $k$, $d$ is the depth, and $N^T$ is the unit normal vector to the image plane. Since we are neglecting translations this becomes

$$(x_k, y_k, 1)^T = [CR_kC^{-1}](x_0, y_0, 1)^T \tag{4}$$

The camera intrinsics matrix can be measured using the classical checkerboard technique. However, we cannot expect the user to calibrate the camera before every photo. Therefore it is assumed that the intrinsics matrix is dominated by the focal length, so that

$$C = \begin{bmatrix} \ell_f & 0 & 0 \\ 0 & \ell_f & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{5}$$

Once we have each $(x_k, y_k, 1)^T$, we can find the blur kernel, or equivalently by calculating the PSF (point spread function). We know the PSF is the trajectory of each point across the image plane. In order to estimate this, I assume that each point on the estimated trajectory is normally distributed in order to eliminate gyro noise.

$$h[m, n] = \frac{1}{K}\sum_{k=1}^{K} F(m - x_k, n - y_k) \tag{6}$$

where F is the two dimensional Gaussian distribution. Since gyroscope movement is uncorrelated we can split this into a product of two independent Gaussians:

$$h[m, n] = \frac{1}{K}\sum_{k=1}^{K} F_1(m - x_k) * F_2(n - y_k) \tag{7}$$

All that is left to is to find the Gaussian distributions and the rotation matrices $R_k$. This will be described in the following sections.

## 2.2 Estimating the Gaussian kernels

A Gaussian function requires a mean and a variance. I estimate these parameters by measuring the gyroscope data for a still camera. This is similar to the method employed by Lee et al. [3]. The values of the gyroscope data can be modeled by a Gaussian as seen below: Therefore, I can take the mean and variance of this data for both

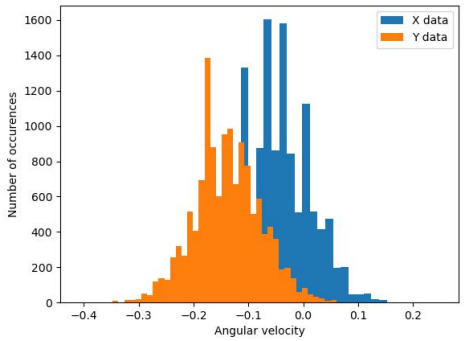


**Figure 2: Histogram of gyroscope data captured when the camera is still. Note that the data approximately follows a Gaussian distribution.**

the x and y axes to compute $F_1, F_2$.

## 2.3 Calculating $R_k$

A gyroscope measures angular velocity, not angular position. To get the angular position, we must integrate the gyroscope readings with initial value $\theta_0 = \vec{0}$. This can be done discretely as follows:

$$\theta_k = (\omega_k * \Delta t) + \theta_{k-1} \tag{8}$$

where $\theta_k$ is the angular position vector at step $k$, $\omega_k$ is the angular velocity vector from the gyroscope at $k$, and $\Delta t$ is the time difference between two gyroscope samples. Figure 3 shows a sample result after integrating the angular velocities.

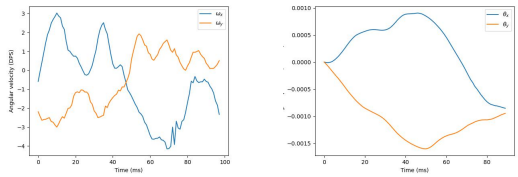

**Figure 3: The left image shows angular velocity read from the gyroscope. On the right, we see the result after integration.**

## 2.4 Non-blind Deconvolution

Once we have an estimate for the PSF, we can apply non-blind deconvolution techniques in order to recover the deblurred image. To solve the new deconvolution problem, I employed a modified Richardson-Lucy algorithm. Traditional Richardson-Lucy uses an

iterative approach to recover the original image. The update step is:

$$u_{i+1} = u_i \cdot \left(\frac{B}{u_i \otimes K} \otimes K^{\dagger}\right) \tag{9}$$

Where $K^{\dagger}$ is the flipped PSF. However, in the traditional Richardson-Lucy, a lot of ringing is prevalent. Yuan et al. [7]. claim that this is due to high contrast edges, so they use a gain-controlled Richardson-Lucy to counteract this. I adopt a similar procedure. My modified update step is

$$u_{i+1} = I_{\text{gain}} \cdot \left[u_i \cdot \left(\frac{B}{u_i \otimes K} \otimes K^{\dagger}\right)\right] \tag{10}$$

$$I_{\text{gain}} = 1 - \alpha + \alpha \cdot c \sum \|\nabla B^l\| \tag{11}$$

where $\sum\|\nabla B^l\|$ is the gradient of the blurred image at the $l$-th step of the Gaussian pyramid, and c is a normalizing constant so that $I_{\text{gain}} \leq 1$. Overall, this modified Richardson-Lucy does a decent job at de-ringing the resulting image, but ringing is still strong for images that are extremely blurred. The results are further evaluated in section 4.

## 3 HARDWARE DESIGN

Now I describe the hardware setup that I used in order to capture movement data. I used the Fermion ICG 20660L module, which combines a 3-axis gyroscope and a 3-axis accelerometer for maximum simplicity. The 3-axis MEMS gyroscope is configured to ±125° since the total angular velocity during a shake does not typically exceed 50°. This breakout module is connected to an Arduino Uno R3 board. The Arduino board connects to my computer via a wired connection and logs sensor data into a text file. I additionally connected a button to the system in order to control when the data logging starts and stops. This entire system is glued onto a flat piece of cardboard and the camera is attached to it. Figure 4 shows my final setup.

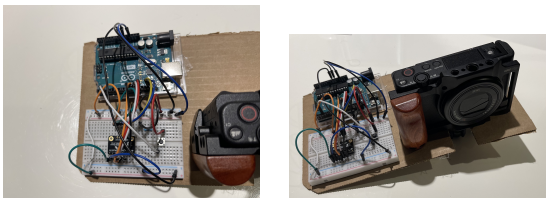

**Figure 4: The left image shows my breadboard configuration. The right image shows the final setup with the camera attached**

## 4 RESULTS

I now present my results. I ran my algorithm on three separate images, each with varying levels of blur. All of the images are taken with a point light source aiming down at it so it is possible to infer the original PSF from certain reflections for comparison. Figure 5 shows each of the original blurry images along with the inferred true PSF.

In figure 6, we can see the projected point trajectories and estimated PSFs. In general, the estimates are extremely close to the true

**Figure 5: Three initially blurred images which I used as input into my algorithm.**

PSFs shown in figure 5. There are some minor discrepancies in the path, which can be due to noise from the gyroscopes. Additionally, since the camera shutter was not directly connected to the Arduino board, I had to manually select the sensor data corresponding to each camera shake. Some of the path discrepancy may also be due to minor errors when manually selecting the sensor data.

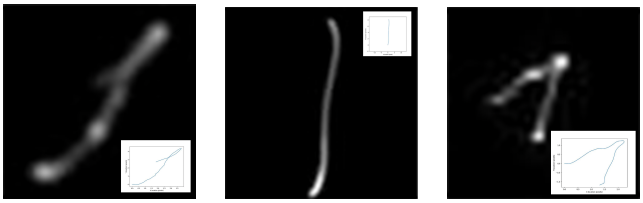

**Figure 6: These are the point trajectory estimations and PSFs for the images shown in figure 5.**

Once I obtain an estimated PSF, the deblurred image is recovered using a gain-controlled Richardson-Lucy deconvolution. Figure 7 shows the final deblurred results for each image. All images show clear improvements in sharpness while also introducing ringing and other artifacts. For example, the authors of the book in the first photo are clearly legible when they were not in the blurred picture. The third photo is the best result, as it sharpens up the slightly blurred image without producing many artifacts. Both the first and

second photos suffer greatly from ringing due to a number of factors. First, it is known that the Richardson-Lucy algorithm produces ringing due to the Gibbs effect. Even though I used a gain-controlled version, the ringing did not entirely go away. Additionally, it is possible that my sensors had too much noise, which caused noisy PSF estimations. This is known to cause weird edges to appear after deconvolution.


**Figure 7: Resulting images after running my full processing pipeline**

## 5 DISCUSSION AND FUTURE WORK

In this report, I presented my blind deconvolution approach. This involved the use of inertial sensors in order to accurately track camera motion throughout each exposure. The benefit of my approach when compared to purely software approaches is that I generate more data which will make an extremely difficult task a bit easier.

The biggest issue with the results produced by my method is that they have too many artifacts. This is due to a number of factors. First, as discussed in section 4, the Richardson-Lucy algorithm will generate ringing due to the Gibbs effect. Although I tried to tackle this with a gain-controlled Richardson-Lucy, unfortunately a lot of the artifacts still remained. Sensor noise could have also contributed to the artifacts, although it may have been mitigated because I used a Gaussian PSF estimation.

Another issue is that my approach for selecting gyroscope data which corresponded to camera shake was not automated. As a result, I had to manually select angular velocity data that corresponded to the camera motion. This worked out decently, but the process was extremely tedious and inefficient. Additionally, it also might have introduced some noise into the PSF estimation which was then propagated after running the deconvolution algorithm. In the future, this could be addressed by wiring the shutter button directly to the Ardunio, so that the logging starts when the shutter is depressed.

Finally, my pipeline is somewhat slow and work needs to be done to speed it up. Richardson-Lucy makes my algorithm extremely slow, as it usually takes at least 20 seconds. This is because of the iterative nature of the optimization. My choice to use Richardson-Lucy was predicated by the fact that it may help with de-ringing. However, given that ringing is still prevalent, it may have been better to use a direct optimization approach such as a Wiener deconvolution which takes only a few seconds at maximum.

While the results are not perfect, my approach is definitely still effective in many cases. It works best when there is not a lot of blur or when the camera motion is more continuous. Even given an image with high blur, the deblurred image usually has legible text and is much sharper. Given the fact that gyropscopes are common on phones, I believe that my approach is definitely a viable method for deblurring images.

## REFERENCES

[1] Fergus, R., Singh, B., Hertzmann, A., Roweis, S. T., and Freeman, W. T. (2006). Removing camera shake from a single photograph. *ACM Transactions on Graphics*, 25(3):787–794.

[2] Joshi, N., Kang, S. B., Zitnick, C. L., and Szeliski, R. (2010). Image deblurring using inertial measurement sensors. *ACM Transactions on Graphics*, 29(4):1–9.

[3] Lee, E., Chae, E., Cheong, H., and Paik, J. (2014). Fast motion deblurring using sensor-aided motion trajectory estimation. *The Scientific World Journal*, 2014:1–7.

[4] Nayar, S. and Ben-Ezra, M. (2004). Motion-based motion deblurring. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 26(6):689–698.

[5] Raskar, R., Agrawal, A., and Tumblin, J. (2006). Coded exposure photography: motion deblurring using fluttered shutter. *ACM Transactions on Graphics*, 25(3):795–804.

[6] Tai, Y.-W., Du, H., Brown, M. S., and Lin, S. (2008). Image/video deblurring using a hybrid camera. In *2008 IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–8.

[7] Yuan, L., Sun, J., Quan, L., and Shum, H.-Y. (2007). Image deblurring with blurred/noisy image pairs. *ACM Transactions on Graphics*, 26(3):1.