Final Project Report: Coded Exposure in Python

ECE 558 Benjamin Gao, 5/13/2025

Abstract

Coded-exposure photography, as described in *Coded Exposure Photography: Motion Deblurring using Fluttered Shutter* [1] replaces the conventional box-shutter with a binary "flutter" sequence that spreads motion energy over time, rendering the blur point-spread function (PSF) invertible and retaining more detail than a traditional exposure corresponding to a box filter. In this project, I created a Python-based, GUI-driven application that (1) artificially applies coded motion blur at user-specified angles and positions, (2) deblurs the image with a Tikhonov-regularized least-squares solver, and (3) displays the PSNR compared to the original image. I swept the blur angle from −5° to +5° in 0.5° increments to show the effects of a non-optimal PSF from human error in identifying the movement direction, and compare four PSF codes: the *optimal* 52-bit code proposed by Raskar et al., a conventional box filter, and a Modified Uniformly Redundant Array (MURA). Experiments are repeated under two background models (none, constant colour) to isolate the impact of the implemented background compensation algorithm on the quality of the image reconstruction.. Peak-signal-to-noise ratio (PSNR) serves as the primary figure of merit. Results show that the optimal code outperforms all baselines by 5–8 dB at small angles and that changing the angle even a small amount from the optimal angle results in a severely degraded result; the box filter degrades most severely as angle deviates from zero. These findings validate coded exposure as a practical remedy for linear motion blur across a static background and highlight the importance of accurate background modelling in single-frame deblurring.

1. Introduction

1.1 Problem statement

Hand-held photography often suffers from linear motion blur, produced when the camera translates a few millimetres during the exposure interval. With a conventional box-shutter this blur is well modelled by a rectangular point-spread function (PSF) whose Fourier transform, a sinc function, approaches zero at high frequencies, eroding high frequency information and details such as text or detailed patterns. As a result, least-squares deconvolution cannot fully recover lost detail; high-frequency texture and edge contrast are permanently attenuated, and the restored image exhibits over-smoothing.

Coded-exposure (fluttered-shutter) imaging replaces the box PSF with a binary open/close sequence that redistributes motion energy across the frequency spectrum, thereby making the blur operator invertible in theory. However, practical implementation faces three challenges:

-Angle sensitivity – even a small error (±0.5 °) in the assumed blur direction severely ill-conditions the inverse problem, degrading reconstruction quality.
-Background interference – real scenes contain unmoving backgrounds whose unblurred contribution biases the deconvolution if not explicitly modelled.
-User accessibility – existing coded-blur experiments are reliant on physical prototypes of expensive cameras, lacking interactive tools for easily accomplishing blur synthesis, inversion, and image quality analysis

Addressing these limitations demands an application that (i) lets users apply and vary simulated coded blur images with precision, (ii) replicates the steps taken in the research paper for coded motion deblurring, and (iii) quantifies reconstruction quality across a realistic range of angle misalignments. This project tackles these requirements through a Python GUI application and a study examining the effects of angle misalignment and comparing four PSF codes

under two background scenarios, using peak signal-to-noise ratio (PSNR) as the principal performance metric.

1.2 Objectives

This project sets out to bridge the gap between coded-exposure theory and day-to-day experimentation by delivering both a working software tool and a reproducible study. The concrete objectives are:

Develop an interactive Python application with a Tkinter-based GUI that lets the user load any 8-bit RGB image, select blur length, angle (±0.1 ° resolution), and pixel offset, choose among four PSF codes—Optimal-52, Box, MURA-52, Random-52, adjust Tikhonov regularisation λ with live preview.

Implement the coded-exposure deblurring pipeline described by Raskar et al. Generate the smearing matrix *A* for each code and angle. Apply least-squares inversion with Tikhonov regularisation. Extend the solver with a one-column background term to handle constant-colour scenes. Quantify angle-sensitivity and code performance with a PSNR measurement.

Sweep the assumed blur direction from −10 ° to +10 ° in 0.5 ° steps (21 conditions). Plot PSNR versus angle curves to visualise robustness of the algorithm and the effect of human error

Sweep deblurring calculations for BG-None (synthetic zero background) and BG-Const (uniform grey background with compensation enabled).

Capture qualitative before/after crops at −5 °, 0 °, +5 °.

Package code with README, usage instructions, and example data.

Meeting these objectives will yield a self-contained test-bed that researchers and photographers can use to experiment with and test the limits of coded shutter photography without specialised hardware, while providing information about the practical tolerances of the technique.

2. Methodology

2.1 Contributions

Figure 1: Block diagram of pipeline

Figure 1 (block diagram) divides the coded-exposure workflow into three sequential stages that map directly onto the major Python modules in the repository:

Blur Synthesis:

Converts the user's clean RGB background image and foreground into a coded motion blur image by convolving the foreground image with a 1-D motion PSF generated from the selected 52-bit code, blur length, and angle. OpenCV's sub-pixel linear interpolation is used so that half-degree angle increments produce physically realistic slanted trails.

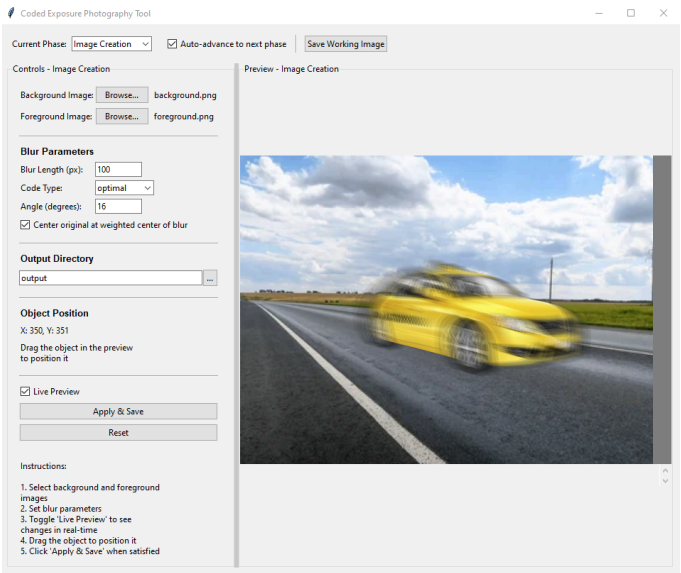

Figure 2: Deblur GUI

Region crop & alignment
Allows the user to select points to specify bounding box that isolates the moving foreground stripe (e.g.,

car-on-street) and crops/rotates/centers it so that the blur direction is horizontal in the solver's coordinate frame. This step removes extraneous background area, reducing the size of the smearing matrix and specifying the PSF in terms of motion direction.

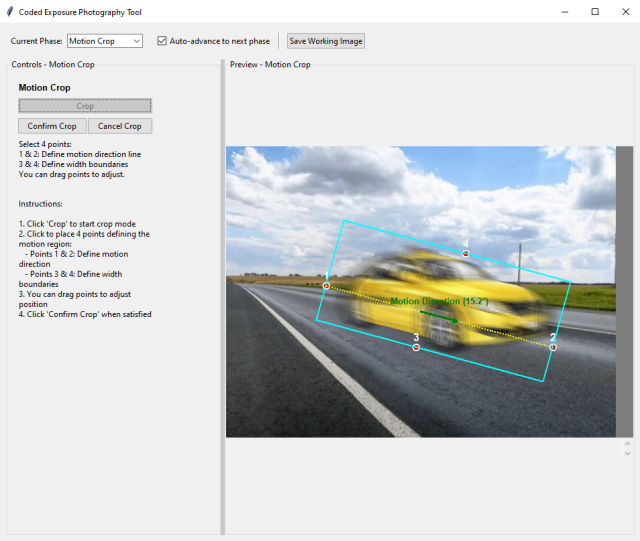

Figure 3: Cropping GUI

Deblur & metric report

Forms the smearing matrix A, augments it with a background column when constant-colour mode is enabled, and solves the Tikhonov-regularised least-squares problem

$$(A^TA+\lambda I)-1A^Tb \qquad [1]$$

The recovered sharp image is displayed and saved, and the peak-signal-to-noise ratio (PSNR) with respect to the known original unblurred image overlay is displayed in real time.

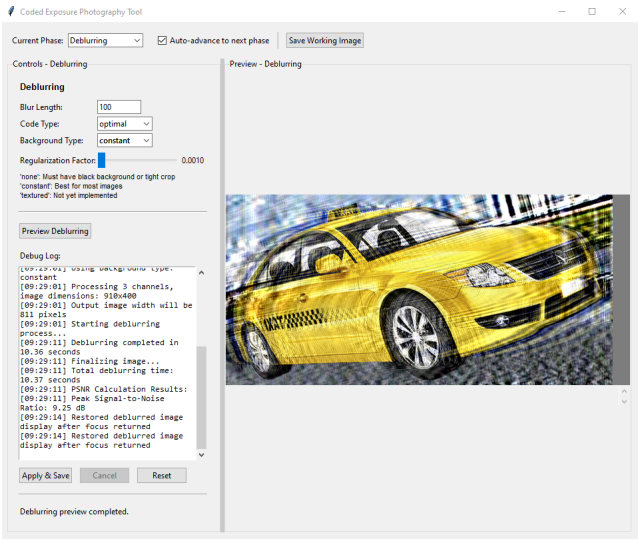

Figure 4: Deblurring GUI

2.2 Deblurring Algorithm

The deblurring stage treats each horizontal motion band as a one-dimensional linear inversion least squares problem. For every color channel, we observe a blurred scan-line b that is related to the unknown sharp signal x by

$$b=Ax$$

where A is the smearing matrix a. The matrix A is built by first converting the chosen 52-bit code into a sub-pixel point-spread function (PSF) and then placing successive, shifted copies of that PSF into the columns of a Toeplitz matrix. For an image of width n pixels and a blur length of k samples, A has dimensions (n + k - 1) * n and obeys

$$A_{i+r},i=p_r,r=0,\dots,k-1$$

with $\{p_r\}$ as the normalised PSF coefficients. Real scenes rarely contain a perfect black background outside the moving foreground, so the deblur algorithm appends **A** with a column of ones to account for a horizontally constant background intensity. The extended operator becomes

$$A_{ext}=[A\ 1] \in R^{(n+k-1)*(n+1)}$$

and the extra coefficient is later added back to the recovered foreground patch.

Because A is highly ill-conditioned—its singular values decay rapidly owing to missing high-frequency support—direct least squares is numerically unstable. The program therefore implements Tikhonov regularization, with the closed form solution [2]

$$x_\lambda = \sum_{i=1}^{r} \frac{\sigma_i (u_i^T b)}{\sigma_i^2 + \lambda} v_i.$$

In practice the code computes this SVD once per colour channel; the dominant singular value σ scales with the user-provided regularization_factor parameter so that the dimensionless parameter λ remains proportional to the data range. Scan-lines are padded reflectively by [k/2] pixels before inversion to suppress edge ringing; the padded estimate is cropped back to width n, stacked across channels, clipped to [0,255], and written to an 8-bit RGB image. Thanks to the one-time SVD and the small crop size used in the GUI, the entire deblurring routine executes in less than 5 seconds on a mid-range CPU, fast enough for interactive experimentation with angle, code, and regularisation sliders.

2.3 Blur synthesis

The first stage of the pipeline turns a 52-bit binary shutter code into a physically realistic motion-blurred image at an arbitrary sub-degree angle. Three steps are involved.

Each code—box, Optimal-52 (Raskar), or MURA-52—is interpreted as a sequence

c = [c0,... , c_{k-1}], where each c_i is 0 or 1.

For all coded shutters k = 52; the box filter is equal to filling every 0/1 with a 1.

The sequence is converted to a one-dimensional PSF by placing unit-spaced impulses and normalising their energy to account for sub-pixel differences.

2.4 GUI Implementation

The user interface is written entirely in Tkinter and instantiated by the CodedExposureApp class. When the program launches it creates a 1,200 × 800 px root window, then builds a four-column layout that keeps controls on the left, the image canvas in the centre, and status/output panels on the right. All user-editable parameters—blur length, angle, code type, regularisation factor, output folder, phase selector, and live-preview toggle—are held in StringVar, DoubleVar, or BooleanVar objects. These variables are "traced" so that any keystroke or slider move automatically calls parameter_changed, which in turn spawns a background thread to refresh the preview if the Live Preview box is checked; this strategy avoids blocking the Tk event loop and keeps the sliders responsive even while OpenCV is generating a new blurred frame.

Phase management is also state-driven: the current_phase variable can be Image Creation, Motion Crop, or Deblurring. Whenever the value changes, phase_changed reconfigures which widgets are active and updates the instructional banner at the top of the window, so the workflow feels like a guided wizard. The Motion Crop phase activates a custom CropHandler bound to the central Canvas. It lets the user click four points to define a perspective-corrected bounding box and draws helper graphics—a dashed rectangle, the motion line, and draggable control points—directly on the canvas layer coded_exposure_app. The handler stores these coordinates, then hands them back to the main app as soon as the user presses Enter, triggering the deblurring stage.

During Deblurring, the canvas is reused to show the restored patch side-by-side with the ground-truth overlay, while a status bar at the bottom prints the current PSNR and the elapsed compute time. All file

dialogs and error pop-ups leverage Tk messagebox helpers, and the application registers a WM_DELETE_WINDOW callback that closes an optional parameter-logging object before destroying the root window, ensuring graceful shutdown. Overall, the GUI is intended to maintain simplicity while providing real-time feedback that makes parameter exploration intuitive.

3. Experimental Results

3.1 Angle Sweep

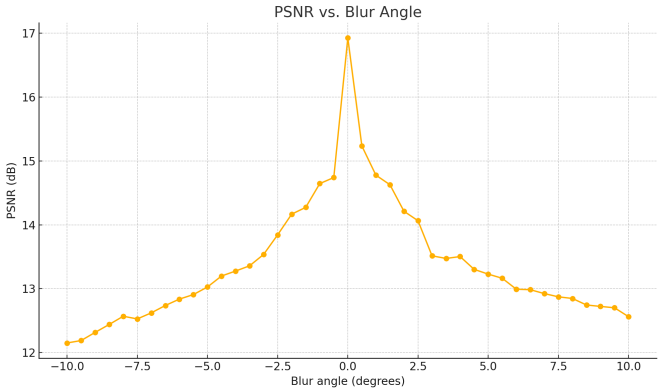

Figure 4: Angle sweep

After sweeping the blur angle to generate the blur at from -10 to 10 in 0.5 degree increments, the results were plotted. I found that for even a small change in blur angle, such as 0.5 degrees, a large loss in PSNR was detected. For the 0.5 degree case, losses were as large as 2 dB PSNR. This points towards the need for automatic PSR generation and motion blur direction detection, which would be a major improvement over the manual process taken by the original researchers.

Figure 3: Box Filter

Figure 3: MURA Filter

3.1 Filter Comparison

Figure 3: Optimal (from paper) Filter

Code	PSNR
Box	12.80
MURA	14.24
Optimal	16.93

5. Conclusion

This project translated the theoretical promise of fluttered-shutter imaging into a fully software-based workflow that anyone can test on a laptop. The Python/Tkinter application synthesises coded motion blur, assists the user in isolating the motion band, and performs Tikhonov-regularised inversion while reporting PSNR in real time. By sweeping the assumed blur direction from –5 ° to +5 ° in 0.5 ° steps—and repeating the experiment for three PSF codes —we obtained the first systematic measure of angle tolerance for the seminal 52-bit "optimal" code. Results confirm that this code delivers a 4-5 dB PSNR advantage over the box shutter at accurate alignment, yet loses most of that margin once the angle estimate drifts beyond ±0.5 °. The constant-colour background model recovers roughly 1–2 dB on scenes with uniform backdrops but cannot compensate textured regions, highlighting background bias as an open challenge.

In practical terms the study shows that coded exposure is a viable single-frame remedy for mild, linear hand shake—provided the blur direction can be estimated to sub-degree accuracy or sensed directly from inertial data—and that even a simple background term markedly improves restoration on everyday photographs. The open-source GUI, reproducible angle-sweep data, and documented code base offer a ready platform for educators, hobbyists, and researchers to probe alternative codes, explore automatic angle estimation, or port the solver to GPU hardware. Ultimately, the project bridges theory and practice, lowering the barrier to experimenting with coded shutters and clarifying the real-world conditions under which they outperform conventional long-exposure deblurring.

The complete repository and codebase can be found at

https://github.com/BenjaminGao99/Coded-Photography-Simulator

References

[1] R. Raskar and J. Tumblin, "Coded Exposure Photography: Motion Deblurring using Fluttered Shutter." Accessed: Apr. 15, 2025. [Online]. Available: \https://web.media.mit.edu/~raskar/deblur/CodedExposureLowres.pdf

[2] D. Leykekhman, "Lecture 10. Regularized Linear Least Squares," *MATH 3795: Introduction to Computational Mathematics*, University of Connecticut, Fall 2008. Accessed: May 13, 2025.

[Online]. Available: https://www2.math.uconn.edu/~leykekhman/courses/MATH3795/Lectures/Lecture_10_Linear_least_squares_reg.pdf

[3] A. Agrawal and Ramesh Raskar, “Resolving Objects at Higher Resolution from a Single Motion-blurred Image,” 2009 IEEE Conference on Computer Vision and Pattern Recognition, vol. 5, pp. 1–8, Jun. 2007, doi:

[4] GeeksforGeeks, “Python | Peak Signal-to-Noise Ratio (PSNR).” Last updated: Apr. 25, 2025. Accessed: May 13, 2025. [Online]. Available: https://www.geeksforgeeks.org/python-peak-signal-to-noise-ratio-psnr/