# Programming Assignment 2 : Motion Deblurring

EE17B031

March 2021

## 1 Introduction

This assignment looks at imaging scenarios that lead to motion blur and two methods to reduce it.

### 1.1 Problem Description

Motion blur occurs when either the camera moves while capturing a static scene, or there is a moving object being captured, or both. Even if we assume knowledge of the blur kernel, deblurring is still hard with conventional photography. Considering 1D blur and the blur kernel to be a box, its representation in the frequency domain is the sinc function. A naive approach is to divide the frequency response of the blurred image with that of the transfer function and estimate the deblurred image by taking its inverse transform. The fundamental issue with this approach is that noise is amplified when the transfer function takes values close to zero. The added problem of a non-uniform blur is common when there are multiple moving objects in the scene apart from the moving camera. Segmenting and estimating the blur kernel for each object is a difficult task. We want uniform blurring throughout the image so that we can escape segmentation.

### 1.2 Implementation

The code has been implemented and tested to run on MATLAB R2020b. The helper functions and image files are assumed to be present in the same folder as the `main.m` file. Results to different sections can be obtained by executing individual sections in `main.m`, which have been separated using MATLAB's %% operator and named appropriately.

## 2 Motion Deblurring with Conventional Camera

We are given a clean image of a scene having $M$ rows and $N$ columns (`fish.png`). We capture the scene using a static conventional camera with the aperture kept

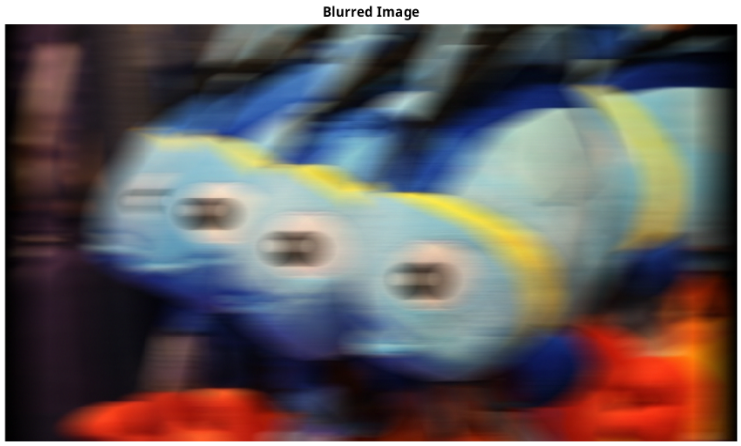


Figure 1: Blurred during conventional capture

open for the full exposure time. The exposure time is 52 seconds. For simplicity, we consider that the camera captures the scene at times $t = \{0, 1, ...50, 51\}$ (i.e. at 52 instants). At time $t = 0$, the image has zero translation. Then, the whole image as seen by the camera moves at 1 pixel per second to the right (horizontal translation).

## 2.1 Generating the Blurred Image

We generate the blurred image having $M$ rows and $N + 51$ columns captured by the camera. We generate translated versions of the image for $t = \{0, 1, ...50, 51\}$ , and average them. We add Gaussian noise of mean 0 and standard deviation 1 (with respect to the maximum intensity of 255) from `gaussNoise.mat`. The result can be seen in Figure 1

## 2.2 Blur Matrix

We form the blur matrix $\mathbf{A}$ of size $(N + 51) \times N$ corresponding to the horizontal translation described above. This matrix will be a Toeplitz matrix. A scaled version of the blur matrix can be seen in Figure 2. We display a scaled version for easier visualization.

## 2.3 Least Squares Deblurring

We deblur the noisy image using the least squares and the generated blur matrix $\mathbf{A}$. We peform the operation on each channel of the image individually. If the blurred channel is given by $y$, the estimate of the deblurred image chennel, $\hat{x}$ is

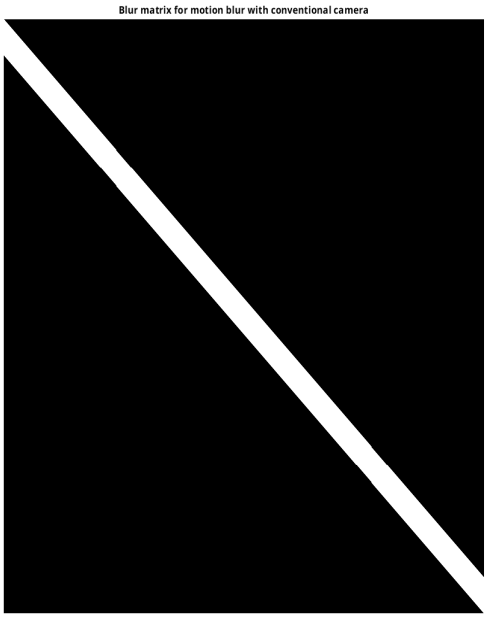


Figure 2: Blur Matrix Conventional Photography

given by

$$\hat{x} = y\mathbf{A}(\mathbf{A}^T\mathbf{A})^{-1}$$

where $\mathbf{A}$ is the blur matrix. The resulting deblurred image can be seen in Figure 3. The RMSE between the clean image and the deblurred image is 10.812395.

# 3 Motion Deblurring with Flutter Shutter

With the same setup as in previous section, we use a *flutter shutter camera* with code

$$1010000111000001010000110011110111010111001001100111$$

where each bit represents one second of the exposure time. The code is given in `codeSeq.mat`.

## 3.1 Generating the Blurred Image

We generate the blurred image with noise (same as for conventional camera) for the exposure code given above. The result can be seen in Figure 4.

## 3.2 Blur Matrix

We form the blur matrix $\mathbf{A}$ similar to what we had done for the conventional camera case. A scaled version of it can be seen in Figure 5.

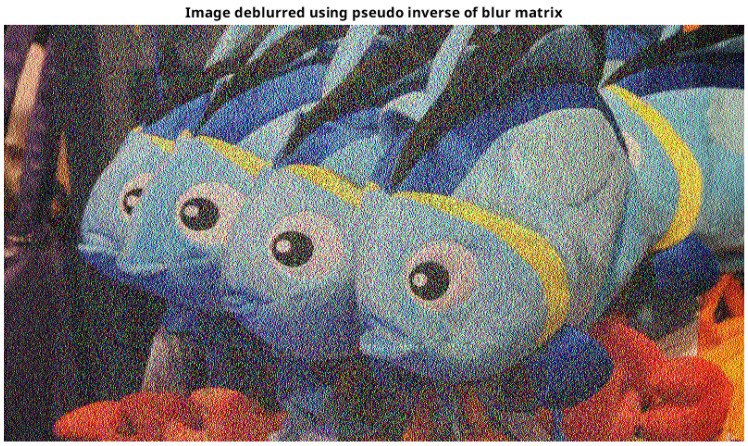


Figure 3: Least Squares Deblurring for Conventional Capture

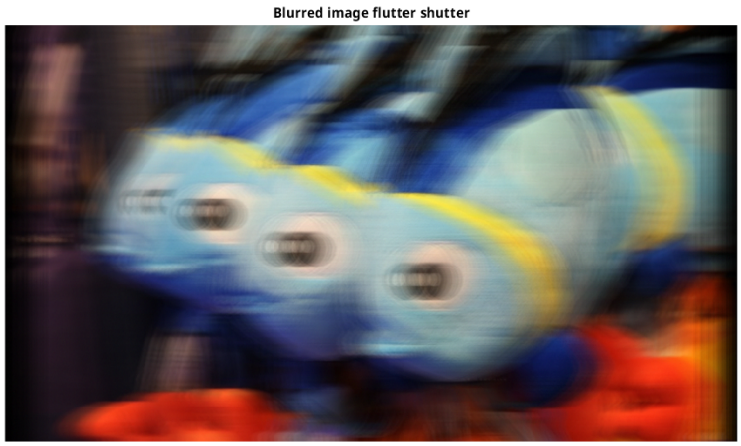


Figure 4: Blurred during flutter shutter capture

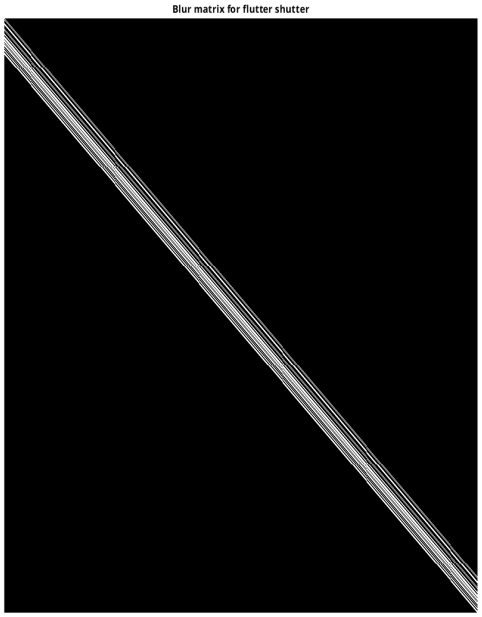


Figure 5: Blur Matrix Flutter Shutter

### 3.3 Frequency Responses

We compare the frequency responses of the DFTs of the conventional and flutter shutter codes. We use the first column of the respective **A** matrices. The magnitude in decibels is plotted in Figure 6. The frequency response of flutter shutter is clearly more desirable. It does not have dips due to zeroes and does not show a significant reduction in magnitude at higher frequencies. This means the image obtained using flutter shutter is easier for deblurring compared to the conventional case. Due to the absence of zeroes, we also do not expect the noise in the deblurred image to be amplified like a conventional camera.

### 3.4 Deblurring in the Absence of Noise

We perform deblurring of the images when noise is not added for the conventional camera and flutter shutter. The results can be seen in Figures 7 and 8. We can make two important observations.

- The deblurred image corresponding to the conventional camera is much better now than the case when noise was added. This can be expected as the noise is absent and no longer gets amplified and ruins the image.
- Flutter shutter is still better. We can still see prominent speckles in the conventional camera deblurred image, which are totally absent in the case of flutter shutter.

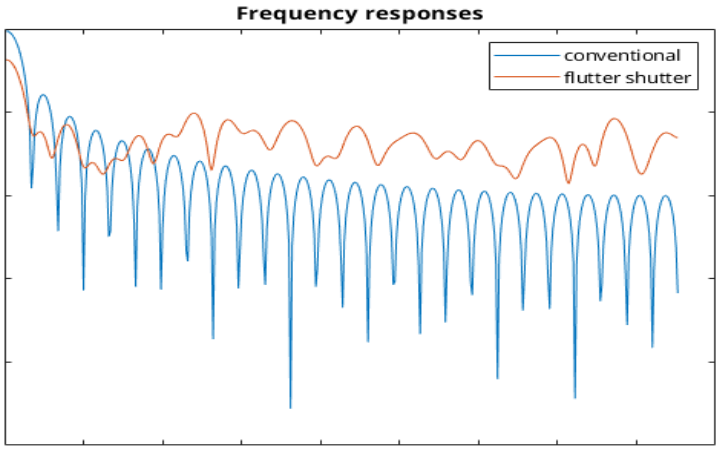


Figure 6: Comparing Frequency Responses


Figure 7: Least Squares Deblurring for Conventional Capture in the Absence of Noise


Figure 8: Least Squares Deblurring for Flutter Shutter in the Absence of Noise

# 4 Deblurring with Motion-Invariant Photography

We are given images of the top view of a road scene (`background.png`) and a foreground moving object (`redcar.png`).

## 4.1 Static Camera

We assume static camera and the car's velocity as k pixels per second to the right (horizontal translation) relative to a static camera, where k$\in \{1, 2, 3\}$. We generate the blurred image captured for an exposure time of 52 seconds. We first create individual frames for each second, i.e. 52 frames. We apply translation corresponding to that frame on the foreground object and merge it with the background to create each frame. For merging, all pixels with non-zero values in the foreground image are assigned foreground pixel values; they are assigned background pixel values otherwise. We average the 52 frames at the end. The resulting images can be seen in Figures 9, 10, and 11.

## 4.2 Motion-Invariant-Photography

We consider the same scene captured using motion invariant photography (i.e. the camera also moves during the exposure). The static background's translational values due to camera motion at each second are given in `CameraT.mat`. We generate the blurred image captured in this case with both camera and object motions for an exposure time of 53 seconds (`CameraT` has 53 entries in it and not 52). We use the following methods.

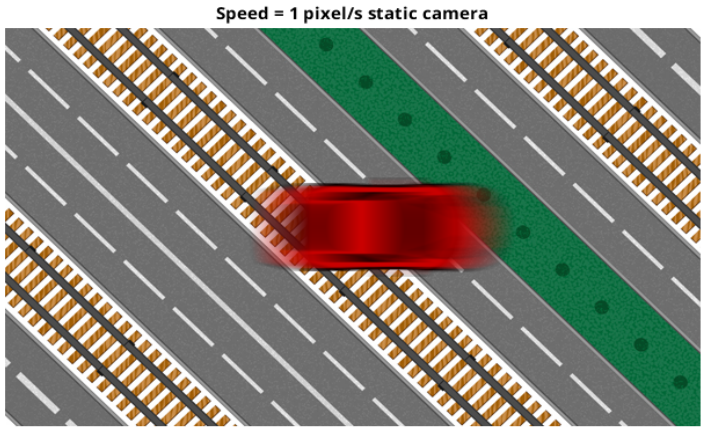


Figure 9: Car moving at 1 pixel/s with respect to a static camera

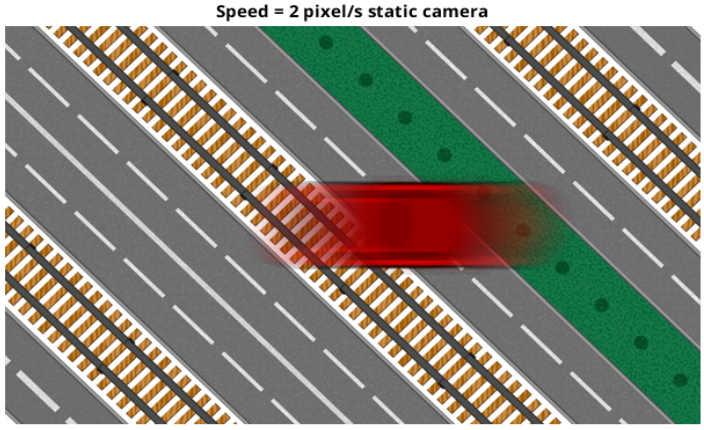


Figure 10: Car moving at 2 pixel/s with respect to a static camera

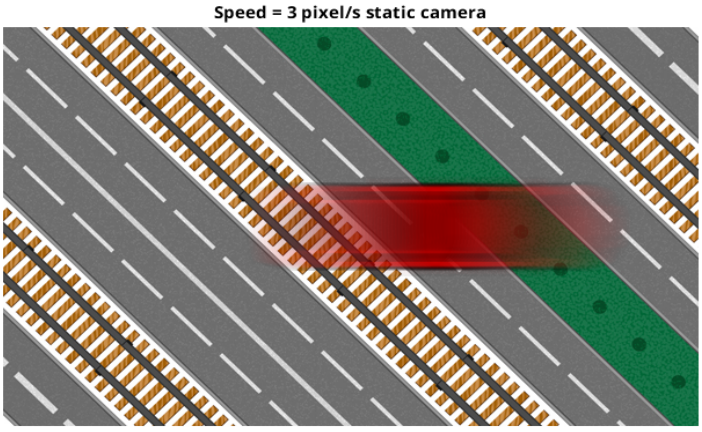


Figure 11: Car moving at 3 pixel/s with respect to a static camera

- We create individual frames for each second: We apply camera translations to the background image and the relative translations to the moving object separately. We merge them to obtain a frame. The relative translations of the moving object with respect to the moving camera can be obtained by *relative translation = static background translation due to the camera - object translation.*
- As the *static background translation due to the camera* values are non integer, we use bilinear interpolation and target-to-source mapping to generate the translated frames.
- We average all the 52 frames to obtain the blurred image

The resulting images can be seen in Figures 12, 13, and 14.

## 4.3 Comparision

The following differences can be observed between the cases of static camera and motion invariant photography.

- As expected, for the static camera, the background is sharp, whereas the moving car is blurred. Whereas in the case of motion invariant photography, both the car and the background are blurred.
- The static camera image suffers non-uniform blurring. Whereas the photo captured using motion invariant photography has uniform blur throughout.

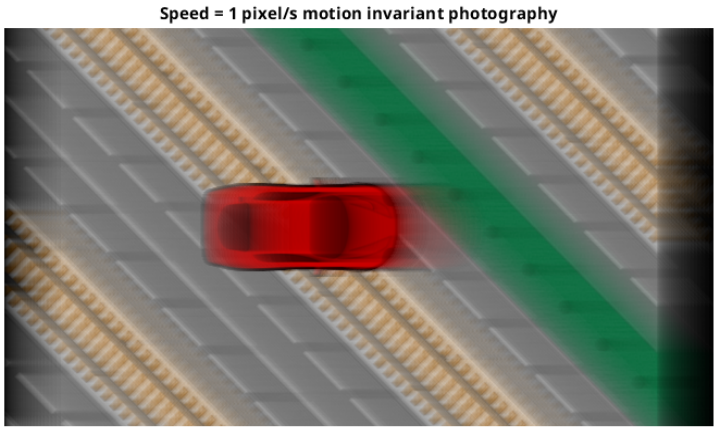


Figure 12: Car Moving at 1 pixel/s Motion Invariant Photography

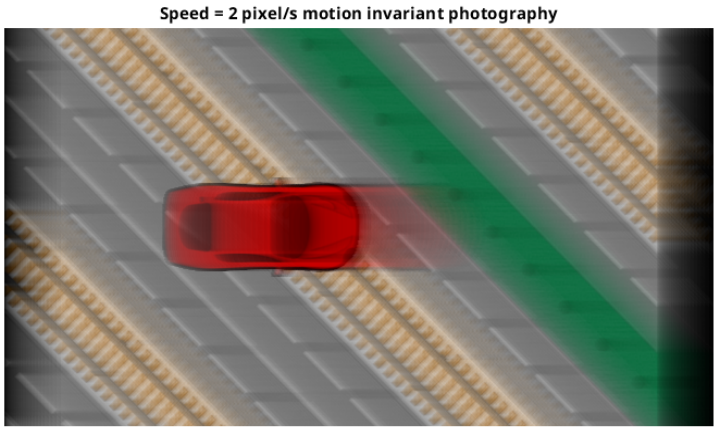


Figure 13: Car Moving at 2 pixel/s Motion Invariant Photography

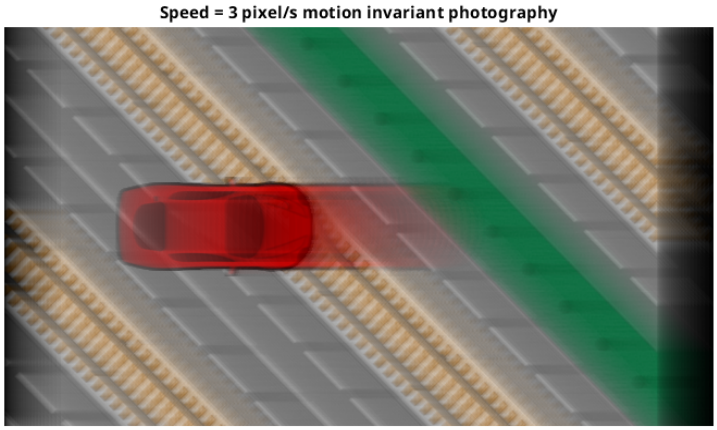


Figure 14: Car Moving at 3 pixel/s Motion Invariant Photography

- When we move to higher velocities for the car, the amount of motion blur increases in the case of static camera capture. Whereas in motion invariant photography, we see the amount of blurring remains approximately constant.
- Using the formula given in the assignment PDF for the car's relative motion, we notice that the car appears to move towards the left in motion invariant photography. This is opposite to that of the static camera case.

## 4.4 PSF for Motion-Invariant Photography

We plot the PSF of the blurred image for the case of motion-invariant photography. If $x$ is the camera translation, the PSF weight at $x$ is $\frac{1}{\sqrt{ax}}$ where $x \in \{0.1, 1, 2, ..52\}$. We use 52 instead of 51 as given in the assignment to ensure that the shape of the blur matrix generated using it is valid for the blurred image. A scaled version of the PSF is shown in Figure 15.

## 4.5 Deblurring the Blurred Image

We generate the blur matrix using the PSF defined in the previous section. A scaled variant of it is shown in Figure 16. The deblurred image is shown in Figure 17.

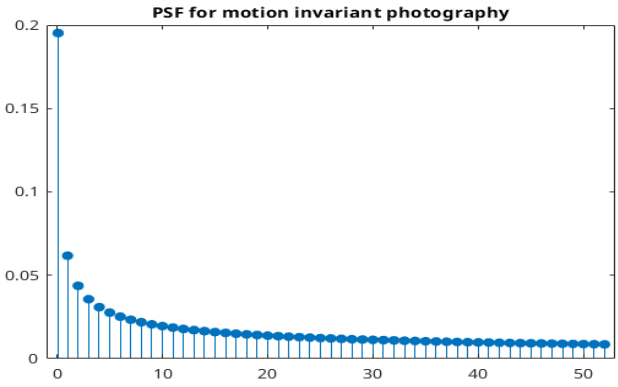


Figure 15: PSF for motion-invariant photography

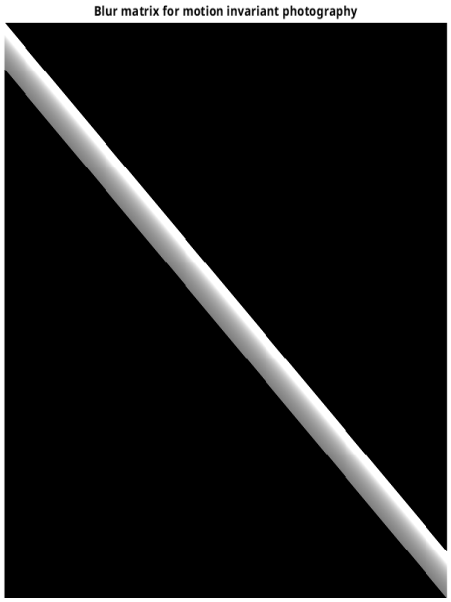


Figure 16: Blur Matrix Motion-Invariant Photography

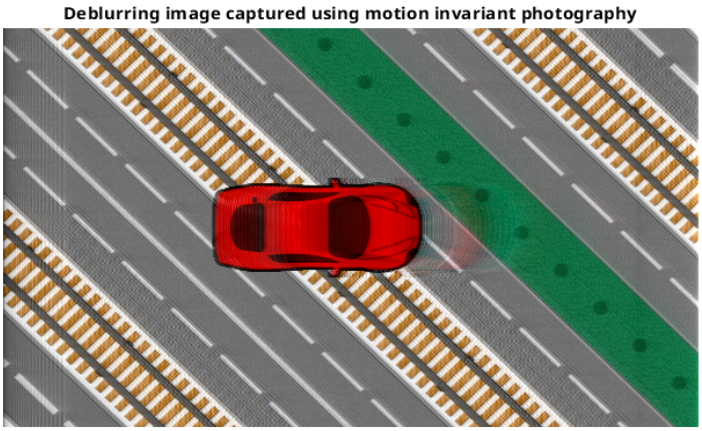


Figure 17: Least Squares Deblurring for Motion-Invariant Photography