# Deblurring text
From theory towards an implementation

**François LANGE**
francois.lange.001@student.uni.lu

A thesis presented for the degree of
Bachelor in Computer Science

| | | |
|---|---|---|
| Local supervisor: | Dr. Ir. Hugo Jonker | `hugo.jonker@uni.lu` |
| Academic supervisor: | Prof. Dr. Sjouke Mauw | `sjouke.mauw@uni.lu` |

Faculty of Science, Technology and Communication
University of Luxembourg
Academic Year 2013 - 2014

# Contents

# List of Figures

**Abstract**

This report present the result of the internship of the bachelor in computer sciences of the university of Luxembourg concerning text deblurring. It explore the theory behind blurring and debluring, to finally describe the implementation of a text deblurring plug-in for GIMP.

Ce rapport présente le résultat du stage de fin d'étude du bachelor d'informatique de l'université du Luxembourg concernant le défloutage de texte. Sont d'abord présentées les principes théorique du floutage et du défloutage de texte, pour ensuite se concentrer sur l'implémentation d'un plug-in GIMP de défloutage de texte.

# Chapter 1

# Introduction

In January 2013, the Dutch newspaper NRC published [5] an article revealing that answers to the national primary school exam were sold online, before the passing of the aforesaid exam. To illustrate this claim, the article included a image of the answers. To prevent leaking exam answers, the newspaper had blurred the questions (cf. Figure 1.1).

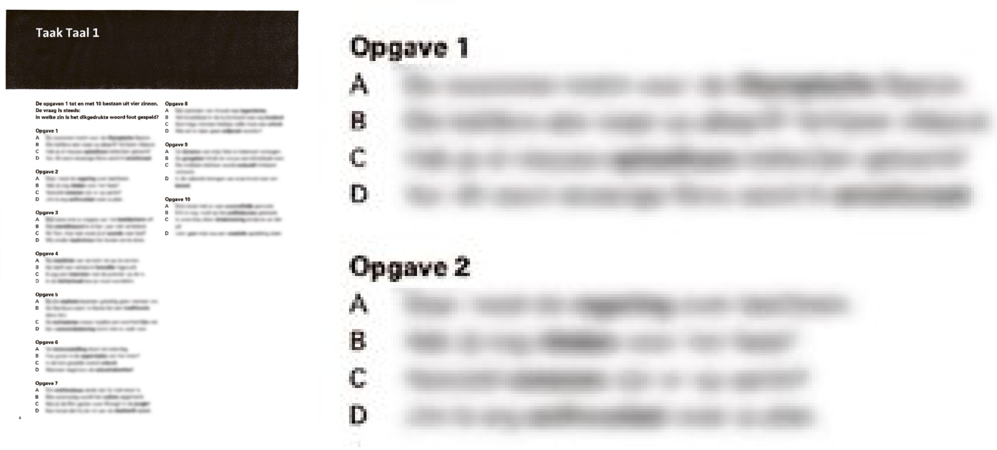

Taak Taal 1
De opgaven 1 tot en met 10 bestaan uit vier zinnen.
De vraag is steeds:
In welke zin is het dikgedrukte woord fout gespeld?
Opgave 1
A
B
C
D
Opgave 2
A
B
C
D

Figure 1.1: The blurred answer of the Cito exam (from [5])

Blurring is a common way to obscure parts of an image that should not be recognizable. It has become easier to use with the spread of digital image editing software such as Photoshop and Gimp. For example, Google automatically blurs faces of people in shots from Google Earth and Google Streetview to address privacy issues, and it will most likely expands this feature to other elements from images of their database.

The main concern of this report is: does blurring actually hide text? That is, is the textual information unrecoverable, or can a blur be "undone" so that the text may be recovered? In particular, if effective deblurring tools were available, Figure 1.1 would have been blatantly revealing exam questions – violating the newspaper's goal to preserve confidentiality of the questions.

This report describes the information and results discovered in the context of the *Deblurring Text* project. This project resulted in an examination of various deblurring methods, an in-depth study of the most promising approach to textual deblurring, and culminated in a partial implementation of this approach as a GIMP deblurring plug-in.

## Approaches to recovering blurred text

At the beginning of this research, a choice had to be made for the method on how to recover the text from a blurred image. Three different ideas were considered, as a general approach to achieve the recovery of a blurred text.

1. By image comparison.
2. Using line recognition to identify characters.
3. By processing a mathematical inverse of the blur process.

Researches were made concerning the potential efficiency and problems of these methods to be able to choose the most adapted among them. The following sections will describe the information gathered about them and the motivation of the choice.

## 1.1 Image comparison

The first possibility consist in a comparison between images of know blurred letters and the letters of the blurred text. This comparison allow to determine the blurred character and, by applying it to all the text, to recover the original text from the blurred one.

To achieve this, a database of self blurred letters would need to be build. The size of this database would depends of the number of elements that need to be recognizable, since the same letter typed in different fonts, put in italic or in bold would looks quite differently and thus needs its own sample for each of these case. In addition to this problem, different level or type of blur would also lead to an increasing size of the database to be able to handle different cases. It is a very high level approach since there is not any specific text properties or any image processing operation used in this method, except a image comparison and an extraction of the letters.

That is why this method has an obvious limitation, as each records of the database would corresponds to specific parameters of the character, which should all be considered since they impact the visual representation of the letter. So gathering a very large set of samples is needed, which would be difficult and cumbersome. Also a large database would require a lot of computational power, since each letters would be compared with each image of the database and image comparison by itself is not a trivial operation.

Another problem is that the letters have to be extracted from to text in order to perform a comparison. To achieve that we would need to find another method which would be able to recover a certain amount of information from the text.

## 1.2 Line recognition

Another approach is to detect the lines that compose a letter, to be able to differentiate them, as each letter have some traits which could be used to differentiate it from the others. For example distinguishing a D from an O by the straight line of the D. For this, we could use the canny edge detection which consists in:

- Finding the intensity and the direction of the variation of the value of the pixels i.e. the gradients of the image. And place these values in a table of the same size as the image.

  **Definition 1** *The gradient, in the case of image processing, represent a directional variation of the value of the pixels. It is commonly used in the field of edge detection, as an edge represent an area with a high variation of the value of the pixel. A value of the gradient of each pixels of a picture can be find by taking the derivative of the vertical and horizontal direction, respectively for the horizontal and vertical intensity of the gradient.*

- Setting to zero the value of a gradient if it is bigger than his directional neighbors. For example if the gradient *g1* is oriented in the North-South direction, it should be bigger than the gradient *gS* and *gN* respectively placed to the north and to the south of his position. This is called a non-maximum suppression of the gradients, because it erase the smaller local gradients. For example if the gradient in the center is oriented in the North-South direction like in figure 1.2 it will be suppressed because it is smaller than 44 and 36. If it is oriented in the East-West direction like in figure 1.3, it will not be suppressed, as 29 is larger than 20 and 18.

$$\begin{pmatrix} 33\rightarrow & 44\leftarrow & 16\uparrow \\ 20\leftarrow & 29\uparrow & 18\leftarrow \\ 10\uparrow & 36\rightarrow & 10\leftarrow \end{pmatrix}$$

Figure 1.2: Gradients map

$$\begin{pmatrix} 33\rightarrow & 44\leftarrow & 16\uparrow \\ 20\leftarrow & 29\leftarrow & 18\leftarrow \\ 10\uparrow & 36\rightarrow & 10\leftarrow \end{pmatrix}$$

Figure 1.3: Gradients map

- Then we should trace the edges through the images by choosing the larger gradients as start points and link together the neighboring gradients in the same direction. The threshold for the most significant gradients can be based on the gradients magnitude of the image, to avoid the problem of selecting non significant gradients or, at the opposite, missing the important ones.

Line recognition is widely used in Optical Character Recognition (OCR). OCR converts an image of a document into a text document. Google Books, for example, used OCR techniques to digitize large collection of books from all over the world. OCR has been widely used for scanning documents. Since scanned images may be blurred, document scanning stakeholders have supported research to address the effects of blur. One approach that promises interesting results is binarising [7].

**Definition 2** *Binarising is conversion operation which a non-binary image to a binary one. A binary image is a black and white image which pixel values are either 1 or 0.*

The problem with line detection as an approach to recovering text is that if the blur is too strong, it will be impossible to detect lines of the text. So an improvement of the image would be needed anyway, which leads us to the final idea.

## 1.3 Blur inversion

The final possibility concerned the mathematical operation of deblurring. It aims to operate the inverse of the blurring operation on the blurred image to recover the initial image. Intuitively, the blur does not destroy all the information, since the changes that occurs for each pixel in the image are dependent on the other pixels. Its a highly complex field but there was already a lot of researches done in this direction, especially concerning the blur of real image since it is a major issue in photography.

## 1.4 Conclusion

The image comparison method is not the best choice as it is “heavy” and dependent on another method. The success of a line detection approach depends on the intensity of the blur.

As such, the obvious choice is to continue in the direction of blur inversion, especially given the promising results achieved by existing research using this approach. In particular, the work by Cho et al. [2] seems to achieve the goals we are aiming for. Therefore, we chose to build forth on this work.

In the rest of this report, we first describe the theory of blurring in Chapter 2. Then, discuss the mathematical foundation necessary to understand the deblurring approach in Chapter 3. Next, we provide an in-depth discussion of the blur inversion approach proposed by Cho et al. in Chapter 4. Finally, we worked towards implementing the approach as a plugin for the open-source *GNU Image Manipulation Program* (Gimp) graphical editing software, which is described in Chapter 5.

# Chapter 2

# Blurring

Blurring is the common name of an image transformation that spreads colors and edges of the image, as illustrated in Figure 2.1. Blurring an image can can make the image totally unrecognizable, if the blur is strong enough. Blur can be caused by optical distortion, e.g. when moving the camera while taking a photo, or it can be added digitally later, by image transformation operation.

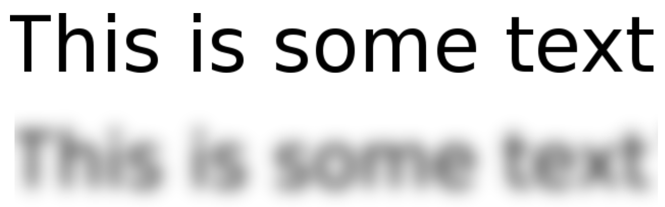
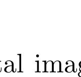
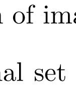


Figure 2.1: A text image before and after blur

## 2.1 Digital blurring





$$\begin{pmatrix} 1 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 1 \end{pmatrix} \qquad \underset{\text{red}}{\begin{pmatrix} 1 & 1 & 1 \\ 1 & 0 & 0 \\ 1 & 1 & 1 \end{pmatrix}} \underset{\text{green}}{\begin{pmatrix} 1 & 1 & 1 \\ 0 & 1 & 0 \\ 1 & 1 & 1 \end{pmatrix}} \underset{\text{blue}}{\begin{pmatrix} 1 & 1 & 1 \\ 0 & 0 & 1 \\ 1 & 1 & 1 \end{pmatrix}}$$

Figure 2.2: Representation of images into matrices form

A digital image is represented as a two-dimensional set of points – a matrix. Each element of the matrix represents a pixel, the smallest component of the image, and the value of this element represents the brightness of the pixel. For color images, each pixel is represented by a three matrices – one for each of the primary colors red, green, and blue. Combined, these matrices can represent any color, since any color can be represented by a mix of red, green and blue. Figure 2.1 illustrates these representations.

Blurring is processed by computing a local average of the value of the pixels and can be used to anonymise people faces or imitating the optical blur. In the remainder, we focus on the case when the user tries to obscure text in an image.

**Definition 3** *The kernel represent the filter that is used to operates the blurring operation, more explicitly the repartition of the weights that will have each neighboring pixels for the blur. It is often represented as a black* *pictures with white pixels as weighting values, the whiter a pixel is, the heavier the associated pixels will be for the blur. When we speak about artificial blurring, it is common to reduce the kernel to two of it's components since the most used ones are symmetric. These components are the size of the matrix, refereed as the radius and the distribution of the value inside of the matrix. For example when the box blur is mentioned, it represent a filter with the same value for each component, meaning that every pixel in the radius of the blur will weight as much as the others and that the final value of the pixel will be the average of the neighboring pixels in the radius of the blur.*

To perform this operation, we need to elaborate a so-called *kernel* i.e. a matrix that will characterize the computation for each pixel of the blurred image. The two parameters that characterize this computation are the size of this *kernel*, which will determine how many pixels are used in the averaging operation, and the distribution of the values in the *kernel*, which will determine the weight of each pixels in the averaging operation that will give the new value to the pixel.

With the following kernel,

$$y = \begin{pmatrix} 0.2 & 0.3 & 0.2 \\ 0.3 & 0.5 & 0.3 \\ 0.2 & 0.3 & 0.2 \end{pmatrix} \tag{2.1}$$

if we want to blur a image $x$ with the kernel $y$ (2.1) to get the blurred image $G$.

with $x_{0,0}$ as the upper left pixel, and $weight(y)$ as the sum of all the element in $y$ (2.5 in this case). To compute the value of the pixel $G_{1,1}$, we would make the following computation.

$$\begin{pmatrix} 0.2 \cdot x_{0,0} & 0.3 \cdot x_{0,1} & 0.2 \cdot x_{0,2} \\ 0.3 \cdot x_{1,0} & 0.5 \cdot x_{1,1} & 0.3 \cdot x_{1,2} \\ 0.2 \cdot x_{2,0} & 0.3 \cdot x_{2,1} & 0.2 \cdot x_{2,2} \end{pmatrix} \times \frac{1}{weight(y)} = G_{1,1} \tag{2.2}$$

Some typical blur that are widely used are the box blur and the Gaussian blur, based on the value repartition in the kernel. For the box blur all the values are the same and for the Gaussian blur, they are set from a two dimensional Gaussian repartition centered on the middle of the kernel.

The box blur is the simplest blurring function. It simply gives to all the pixels in the kernel the same weight. Thus this result in a image where every pixel has the average colour of its neighboring pixels.

The Gaussian blur is one of the most used blur functions. Its characteristic is to weigh pixels in the kernel based on a Gaussian distribution centered on the center of the kernel. To build this kernel, we can use two-dimensional Gaussian function.

$$Gaus(x, y) = \frac{1}{2\pi\sigma^2} \cdot e^{\frac{x^2+y^2}{2\sigma^2}}. \tag{2.3}$$

With the standard deviation of the Gaussian function and $x$ and $y$ the position in the kernel relative to the center. So we have the standard deviation and the blur radius as parameters to instanciate a specific Gaussian blur. But since the value of the Gaussian distribution is almost equal to zero when the distance from the center of the function is greater than $3\sigma$ (at a distance of $3\sigma$, the value is already 500 times smaller), it is not necessary to have a kernel larger than $\lceil 6\sigma \rceil \times \lceil 6\sigma \rceil$ . One of the reasons that this function is interesting for artificial blurring is that we can compute the blur average first horizontally with a one dimension Gaussian kernel, defined by

$$Gaus(x) = \frac{1}{\sqrt{2\pi\sigma^2}} \cdot e^{\frac{x^2}{2\sigma^2}} \tag{2.4}$$

and then reapply another Gaussian kernel vertically to the result image, and this will result with the same blur as with the two dimensional function with a significant reduction in computing time. There exist also other specific blur, like the motion blur, that try to copy the effect of movement, or the pixelizing blur that reduce the number of pixel in the image.

**Definition 4** *[6, p. 100] A convolution between two functions $f$ and $g$, defined on the same domain, can be represented as a product $h$ which represent on each point of itself the value of the integral of $f$ around this point, pondered by $g$ around the origin. In image processing, this is known as a neighborhood operator, because for images, which can be represented as discrete two dimensional functions, this operation is based on a weighted average of the neighboring pixels of $f$ based on $g$. An interesting property of this operation is that a function* *convolved with an impulse signal $\delta(i, j)$, an image which every pixels is equal to zero except at the origin, will reproduce the function without any alteration.*

With a formal point of view, if we consider the image as a function, we can represent the blurring process as a convolution between the function of the source image and the kernel, which result is the function of the blurred image. Convolution is an operation that forms a new function from two other functions defined on the same domain, commonly represented with the operator *. In the case of blurring an image $f$ with dimensions $M \times N$ and a blurring function $h$ with dimension $m \times n$ we can write the convolution as

$$g = f * h = \sum_{i=0}^{m} \sum_{j=0}^{n} f(i, j)\ h(x - i, y - j) \tag{2.5}$$

Convolution can also be transformed into a simple product with the help of the Fourier transform, as the Fourier transform of the product of convolution is equal to the product of the Fourier transform of the source functions. This gives us the following, with $F$, $G$, $H$ being the Fourier transform of $f$, $g$, $h$, respectively:

$$G = FH \tag{2.6}$$

Of course, given the discrete nature of a digital image, the Fourier transform used is the discrete Fourier transform.

# Chapter 3

# Deblurring

As explained previously, the subject of this internship concern the possibility of reversing the blurring process to recover text. This section will introduce and describe different concepts that allow to achieve this goal.

## 3.1 Inverse filtering

Since blur can be represented as a convolution between the original picture and the kernel, it seems reasonable to think that the process is reversible. This is known as a deconvolution. The first idea is to solve the same equation as the blur with a different unknown, the source image instead of the blurred image. This method is known as inverse filtering. With $H(\omega_1, \omega_2)$, $G(\omega_1, \omega_2)$ and $F(\omega_1, \omega_2)$ being respectively the Fourier transform of the kernel $h(x, y)$, the blurred image $g(x, y)$ and the original image $f(x, y)$, we have:

$$F(\omega_1, \omega_2) = \frac{G(\omega_1, \omega_2)}{H(\omega_1, \omega_2)} \tag{3.1}$$

A problem that arises is that $H$ can be equal to 0 for some value, so we can apply a threshold to limit the value of H in this case. This would leads to:

$$F(\omega_1, \omega_2) = \begin{cases} \frac{G(\omega_1, \omega_2)}{H(\omega_1, \omega_2)} & \text{if } H(\omega_1, \omega_2) > \gamma \\ \frac{G(\omega_1, \omega_2)}{\gamma} & \text{otherwise} \end{cases} \tag{3.2}$$

But this method have two significant limitation:

First, if noise has been introduced in the picture after the blur, it will be considerably amplified by the deblur operation. This phenomenon is caused by the frequency distribution of the noise, which is larger in the higher frequencies than in a natural image. Since these high frequencies are reduced by the blur, the deblur operation tends to amplify them to recover the original image. Thus, even a tiny amount of noise would make the picture unrecoverable via inverse filtering. This is not a problem if the noise addition is prior to the blur since it will act as an noise filter.

The second problem is that we need to know the kernel that has been used for the blur, which is not the case if want to recover the text from a picture that we did not blur ourselves.

### 3.1.1 Wiener filter

One approach to solve the noise problem is to use the Wiener filter [9]. It is a formula found by Norbert Wiener in 1942 that operates an inverse filtering, which has the additional feature of minimizing the effect of noise. It is given by

$$F(\omega_1, \omega_2) = \left( \frac{1}{H(\omega_1, \omega_2)} \frac{\mid H(\omega_1, \omega_2) \mid^2}{\mid H(\omega_1, \omega_2) \mid^2 + K} \right) G(\omega_1, \omega_2) \tag{3.3}$$

with $K$ as an approximation of the signal-to-noise ratio. This is one of the most optimal approach if there is no assumption concerning the properties of the image that we try to deblur, like the prevalence of certain frequencies, and given we know the blurring kernel.

### 3.1.2 Blind deconvolution

The solution to this problem is to perform a blind deconvolution. This is a set of methods that applies a deconvolution with an initial kernel, and then refines the kernel estimation by analyzing the quality of the deblurring. These steps are then repeated with the refined kernel until the moment when the quality of the deblurred picture is considered good enough. Epshtein et al. [5] developed a method of this kind. Its implementation is explained in the implementation section.

## 3.2 Artifacts and limitation

Current deconvolution methods show good performances if there is no noise and the blur kernel is well enough modeled. But this is not always the case. In Figures 3.1 and 3.2 is an example of a simple deconvolution (with the real kernel known) of a deblurred picture with and without noise. The first image of the figure shows the original non-blurred image, when the other is the result of a simple deconvolution.

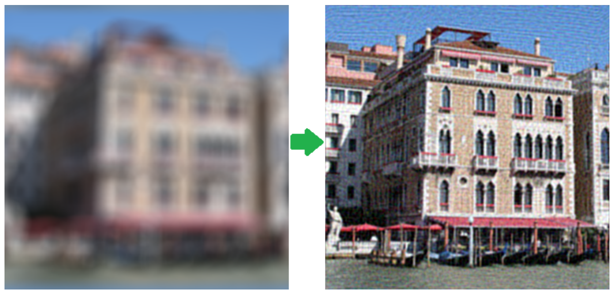


Figure 3.1: Result of an inverse filtering without noise (from [9])

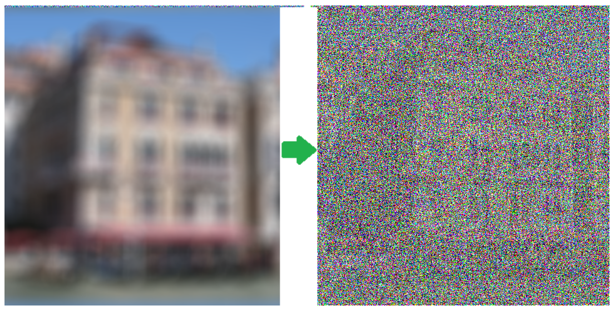


Figure 3.2: result of an inverse filtering with noise added to the blurred picure (from [9])

As we can see, the recovery becomes pretty bad with the noise addition. This make noise limitation factor for deblurring

Another drawback is the emergence of ring artifact in deblurred image, as illustrated in Figure 3.3

According to [8] this kind of artifact appears near strong edges of the images. It has significantly less impact than the noise on the final result of the picture.

## 3.3 Text deblurring properties

As explained in [1], text images have some specific properties that can render a good deblurring algorithm, designed to deblur natural pictures, quite ineffective. If we consider the gradient distribution of the pictures, i.e. how many small and large gradients are the image , we can see that the repartition of these values is wider for a non-blurred natural image than for its blurred version. This is explained by the fact that the blurring operation reduces the value of the higher gradients by smoothing the pixel colors. For a text image,

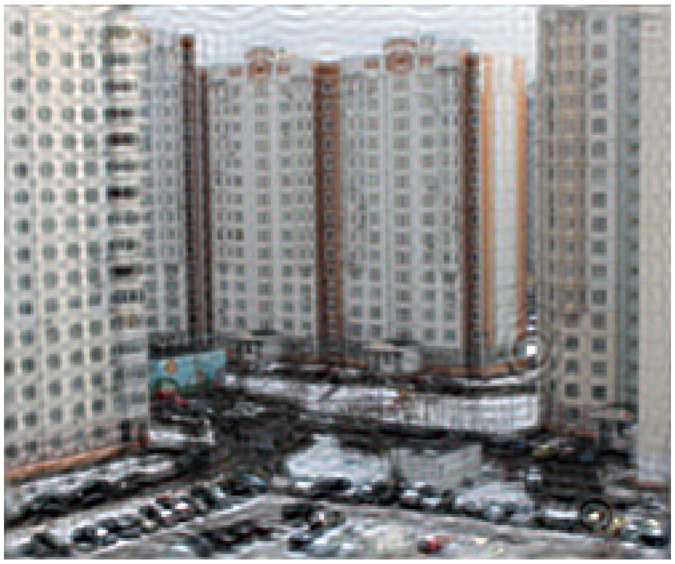


Figure 3.3: Example of ringing artefact occuring with a deblurring from a natural blur (from [9])

the repartition of the gradients is quite similar to the repartition in the blurred natural image, with a very little amount of large gradients, it is due to the fact that there is much less detail in the background of the text. This is the main reason that motivates the authors of this publication to focus on a text specific deblurring method.

Chapter 4

# Practical deblurring using Cho et al.'s approach

In this chapter, we investigate and explain the mathematical details of the algorithm proposed by Cho et al. [2] to deblur text.

## 4.1 Cho et al.'s main algorithm

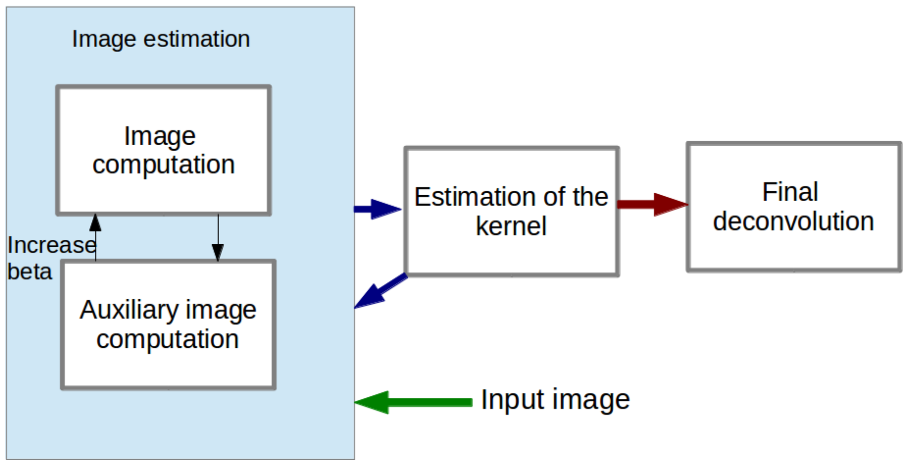


Figure 4.1: Graphical representation of Cho et al.'s algorithm [2]

### 4.1.1 General operation

This algorithm is an implementation of the blind deconvolution, which was presented in the previous section. As shown in Figure 4.1, this consist of an iteration of steps that are repeated to alternatively approximate the kernel and the deblurred images. And then a final deconvolution with the last kernel estimation is processed to recover the original image.

## 4.2 Text image properties

The main characteristic of this method is that it use some specific properties of text images to improve the efficiency of the deblurring operation.

These properties are respectively a high contrast between the characters of the text and the background, and a high uniformity of their color. To make their methods more general, the authors have decided to not assume that the background was uniform. Thus, the algorithm remain valid for text images with a complex background.

To enforce these properties into the processing of the images, the authors decided to include an auxiliary image in the optimization computation of the images. This image is generated with the Stroke Width Transform, which is an image operator used to detect text into natural images.

**Definition 5** *Epshtein et al. [3] describe a method that allow detection of the text inside of a image, with the help of a new kind of image operator, the* stroke width transform *(SWT). This detection is based on the characteristic of the texts to maintain an approximatively constant stroke width, in opposition with other elements in the image, where the stroke variance is bigger. It is processed via the following steps*

1. *We initialize all the elements of the SWT to $\infty$*
2. *We compute the directional gradient of each pixels of the image.*
3. *Then from each gradients $g$ we follow a line in the direction of this gradient and we search for a gradient $f$ with the same intensity but in the opposite direction*
4. *If we effectively found a gradient $f$ with this condition, we set the SWT value of all the pixels between $f$ and $g$ as equal to the distance between $f$ and $g$, if the SWT value is not smaller than this distance.*

*The first step to perform the SWT is to compute the directionnal gradient of each pixels of the image.*

## 4.3 Estimation of the image

The image estimation step is itself composed of an alternate computation of the deblurred image and the auxiliary image.

The computation of the original image $l$ is done by solving

$$l = \mathcal{F}^{-1}\left(\frac{\overline{\mathcal{F}(k)}\mathcal{F}(b) + \beta\mathcal{F}(a)}{\overline{\mathcal{F}(k)}\mathcal{F}(k) + \beta\mathcal{F}(1) + \lambda_l\overline{\mathcal{F}(\nabla)}\mathcal{F}(\nabla)}\right), \tag{4.1}$$

where the functions $F()$, $F^{-1}()$ and $overlineF()$ denote the Fast Fourier Transform (FFT), the inverse FFT and the complex conjugate of the FFT, respectively. The terms on which we apply these operations are the kernel $k$, the blurred image $b$, the auxiliary image $a$ and the gradients of the recovered image $\nabla$. $F(1)$ is a particular case as it represent the FFT of the delta function, which is equal to 1 everywhere. The delta function is defined as equal to zero everywhere but at the origin and to have a derivation over it's domain, from $-\infty$ to $+\infty$, equal to 1. Finally, $\lambda 1$ is a weighting constant and $\beta$ is a weighting variable that is incremented after each image estimation. The purpose of $\beta$ is to reduce the importance of the auxiliary image at the beginning of the process, since it may not be initialized well.

After this, the auxiliary image is build with

$$a_i^* = \begin{cases} a_i^P & \text{if } \beta \mid l_i - a_i^P \mid^2 < d_{MAX} \\ li & \text{otherwise} \end{cases}, \tag{4.2}$$

where $a^P$ represent an ideal image satisfying the text image properties described previously. It is computed by applying the Stroke Width to the previously refinned image $l$. $\beta$ is the weighting variable described previously and $d_{MAX}$ is a constant larger than the maximum pixel value of $l$.

## 4.4 Estimation of the kernel and final deconvolution

Finally, after computing a better estimation of the original image, the algorithm make an estimation of the kernel that was used for the blur operation. This estimation is obtained by solving

$$E(k) = \sum\nolimits_{\partial^* \in \theta} \omega(\partial^*) \mid \partial^* b - k * \partial^* a \mid^2 + \gamma \mid k \mid^2 \tag{4.3}$$

After this, the previous step are repeated with the improved kernel and image to improve the quality of the recovery. But the result of the paper have shown that the first iteration of this process already achieve a good deblurring quality, as shown Figure 4.3.

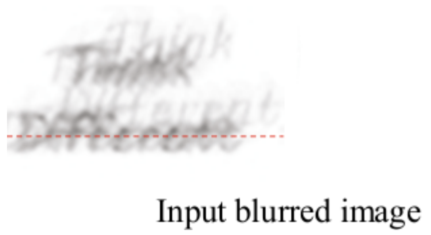


Figure 4.2: Original blurred image (from [2])

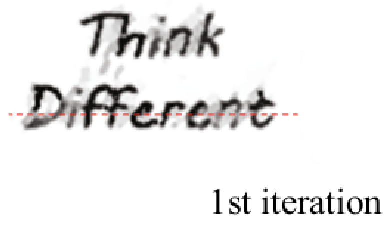


Figure 4.3: Result of the deblur operation with 1 iteration (from [2])

When the kernel recovery is considered to be good enough, a final deconvolution is processed to process the best deblur image possible from the estimated kernel. This deconvolution is achieved by solving

$$E(l) = \mid b - k * l \mid^2 + \rho T(l) \tag{4.4}$$

The two previous equation are energy minimization problems. This problematic was not deepened during this internship and is left for potential continuation of this work.

## 4.5 Limitations of Cho et al.'s approach

The results achieved by this approach look really good. However, this deblurring algorithm does not work very well under certain conditions.

As explained before, noise is a common limitation for deblurring, because it is amplified by a deconvolution. As we can see in Figure 4.4, the algorithm does not work well if a large amount of noise is added in the picture. Furthermore, if characters touch each other, the algorithm fails too.

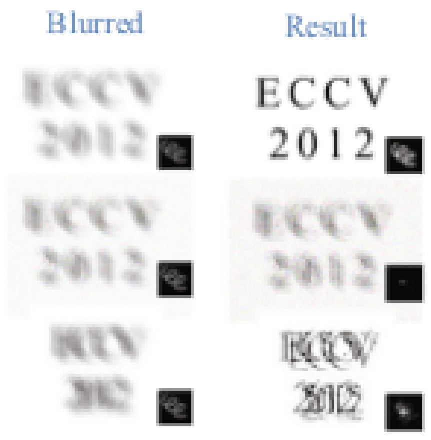


Figure 4.4: Impact of noise and collision of the characters on the deblurring (from [2])

# Chapter 5

# Implementation

In this section will be explained my choices for the implementation of the previously described method in the form of a gimp plug-in.

## 5.1 GIMP plug-in

GIMP, for Gnu Image Manipulation Software, is an open-source software similar to the well known Photoshop. It allow the user to edit images via a large set of tools, like brushes and filters. The plug-in is usable in the form of a filter that process the whole image opened in GIMP or a selected part of this image.

A Gimp plug-in is a software that can process images, but that is used within GIMP and thus relies on it to get input and to displays the image result. It can be written in C, Scheme, Python, or Perl, but for this project my choice has been to code the plug-in in C as it is the language among these which I am the most familiar with. It is also the most used language in GIMP plug-in for computing intensive algorithm . To be usable for Gimp a plug-in must implement several functions, which are

```
 void run ();
 void query();
```

and include and link the GIMP library.

When Gimp is started, it look after plug-ins in predefined folders and ask to identify themselves by calling the query() function of each of them. With this, the software will know the name, the place in the menu and the parameters of each plug-in.

After this, when the plug-in is needed the software call the run() function which implement the image processing logic of the plug-in. There is three important function called during its runtime in the respective order

```
IplImage* get_image (GimpDrawable *drawable);
IplImage* deblur (IplImage* input_img);
void set_image(IplImage* output_image, GimpDrawable *drawable);
```

`get_image()` purpose is to transform the input image in the form of a GimpDrawable to an IplImage, which is the standard image object for openCV.

`set_image()` on the other hand, transform back a GimpDrawable to an IplImage, to be able to send back the result of the deblurring.

The logic behind these functions is to simply access to the value of each pixel of the picture from the source data and then to copy this value at same pixel position in the output data.

### 5.1.1 Performance

However, at the beginning of the plug-in implementation this process was really slow, up to 30 seconds for a simple image of $400 \times 400$ pixels. This was caused by the accessing method to the pixels of the GimpDrawable, which required to call a specific function for each pixel. I solved this issue by using a function to get a full row of pixels from the GimpDrawable and then accessing the pixels value from this row. This improved significantly

### 5.1.2 Debugging

The debug of a Gimp plug-in is intrinsically difficult since it doesn't run alone and need Gimp to be executed. The advice given to developers on the Gimp developer wiki is to start Gimp from the terminal to be able to detect print statements and to use them to detect the bugs.

## 5.2 Deblur Function

When the plug-in has successfully converted the image from the GimpDrawable format to IplImage, the function Devblur() is called.

```
Deblur(image){
1.Initialization of the different images used for the deblurring
2.Initialization of the FFTW elements
3.Start of the deblur algorithm
4.Final deblur of the image
}
```

### 5.2.1 OpenCV

OpenCV is an open source library used to manipulate images. It is used in this software to process the pictures at the pixel level by accessing ab array of char representing the pixels value of the image.

### 5.2.2 FFTW

FFTW [4] for "Fastest Fourier Transform in the West" is a free C library for computing the discrete Fourier transform of any size or dimension. It was chosen for its efficiency and simplicity of usage. The reason why it need an initialization outside of the deblur algorithm is that before applying the FFT, the library require to create an object called plan where we fix the size and the dimension of the input and the direction of the transform (choosing between the transform or the inverse). This creation take several seconds and would slow the algorithm if a plan is created again each time a FFT is needed. Since the size of the transform is always the size of the picture, two plan are created, one for the FFT and one for the inverse that take place in the image refinement equation.

### 5.2.3 Deblur algorithm

After the initialization of the different images and the FFT elements, begin the core of the algorithm, that can be described as follow

```
while(i< number of iteration){
  while(beta<5){
    Original image refinement
    Auxiliary image refinement
    beta=beta*1.5
  }
  kernel estimation
  i+=1
}
final deconvolution
```

When this process is over the final image is returned.

## 5.3 Original image refinement function

The computation of the original image is the application of the equation 4.1. The first part of this function consist to apply a FFT to each elements of the equation, for this we convert the image from `IplImage` of openCV to the `fftw_complex` of the FFTW. Then we apply the operations between elements as

## 5.4 Auxiliary image refinement

The computation of the auxiliary image require two step. First we need to build an ideal image by applying the SWT to the original image computed previously via the function

```
 function_SWT()
```

And then we can apply the equation 4.2 to build the auxiliary image from the original image and the ideal image which is done with the function

```
 check_equ9()
```

To apply the SWT, a first attempt has been made to use an open-source library named CCV which already implement it. Unfortunatelly, the lack of documentation of this library has made its use and comprehension very difficult. After several failed attempt to include its implementation of the SWT into the plug-in the decision was took to give up with this library and to try to self-implement the SWT.

## 5.5 Completion of the plug-in

To complete this plug-in, it will be necessary to complete both the original and auxiliary image refinement functions since they are not completely functionnal. And also to understand the theory of energy minimization to be able to implement the equations 4.4 and 4.3 into the functions

```
kernel_refine()
final_deblurring()
```

.

# Bibliography

[1] Xiaogang Chen, Xiangjian He, Jie Yang, and Qiang Wu. An effective document image deblurring algorithm. In *Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on*, pages 369–376. IEEE, 2011.

[2] Hojin Cho, Jue Wang, and Seungyong Lee. Text image deblurring using text-specific properties. 2012.

[3] B. Epshtein, E. Ofek, and Y. Wexler. Detecting text in natural scenes with stroke width transform. In *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*, pages 2963–2970, June 2010.

[4] Matteo Frigo and Steven G. Johnson. The design and implementation of FFTW3. *Proceedings of the IEEE*, 93(2):216–231, 2005. Special issue on "Program Generation, Optimization, and Platform Adaptation".

[5] NRC Handelsblad. Cito-toets 2013 te koop via Marktplaats. http://www.nrc.nl/nieuws/2013/01/29/nrc-handelsblad-cito-toets-2013-te-koop-via-marktplaats/, January 2013.

[6] Szeliski Richard. *Computer visions, algorithms and applications*. Springer, 2011.

[7] Mauritius Seeger and Christopher Dance. Binarising camera images for ocr. In *Document Analysis and Recognition, 2001. Proceedings. Sixth International Conference on*, pages 54–58. IEEE, 2001.

[8] Qi Shan, Jiaya Jia, and Aseem Agarwala. High-quality motion deblurring from a single image. In *ACM Transactions on Graphics (TOG)*, volume 27, page 73. ACM, 2008.

[9] Vladimir Yuzhikov. Restoration of defocused and blurred images. http://yuzhikov.com/articles/BlurredImagesRestoration1.htm, 2012.