The Impulse Responses of Block Shift-Invariant Systems and its use for Demosaicing Algorithms

Yacov Hel-Or
Hewlett-Packard Labs Israel *

Abstract

Shift-invariant linear algorithms can be described completely by the algorithm's response to an impulse input. The so called *impulse response* can be used as filter kernels when the algorithm is equivalently implemented using convolution. This, however, is not true when the system is block-shift invariant, i.e. when invariance is only at repetitive locations. This paper describes a generalization of the impulse response for block shift-invariant systems. The proposed technique, takes any computer program which implements a linear block-shift-invariant algorithm, and produces its equivalent filter kernels. These kernels can then be applied efficiently by using convolution. This greatly reduces run time, especially when the convolution is implemented in hardware, or using the FFT.

This algorithm can assist in finding filter kernels for interpolating missing values in mosaic images acquired by digital cameras. Using this approach any algorithmic description for the demosaicing problem, which is linear and block shift-invariant, can be translated into actual filter kernels that can be applied efficiently.

*Address: HP Labs Israel, Technion City, Haifa 32000, Israel. Email for contact: renato@hpli.hpl.hp.com

1 Introduction

A color image is typically represented by three bands each of which is a matrix of values representing the responses of an array of photo-sensors to the viewed scene. The three bands are often referred to as red (R), green (G), and blue (B) according to their spectral sensitivity. Thus, three numbers are given at each matrix location composing a *pixel* value. Most CCD cameras, however, provide only a single value for each pixel due to their inability to position three photo-sensors at the same location. In these cases, the captured image is composed of three bands whose pixel values are partially sampled. The Bayer color filter array (CFA), given in Figure 1, is an example of a CCD array with a typical photo-sensor arrangement.

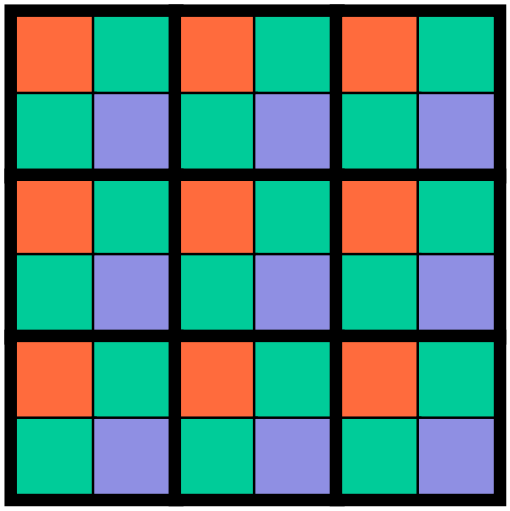

Figure 1: A typical arrangement of photo-sensors in a CCD array. This arrangement provides three color bands whose pixel values are partially sampled. Note that in this example, the number of green samples is twice that of the red and the blue. This corresponds to the varying spatial resolution of the human visual system at different spectral wavelength.

The *demosicing* problem deals with the reconstruction of a color image $I(x, y, c)$, where $c \in \{R, G, B\}$, from a partial sampling $D = S(I)$ of its pixel values. Here, S represents the sampling operator applied to the image I.

This reconstruction problem is, of course, an under-determined system since the solution space includes infinite images satisfying D. Several approaches were introduced to solve the demosaicing problem (see e.g. [3, 1, 2, 5, 4]). These methods assume some prior knowledge about the probability of the input images $P(I)$. The solution is then chosen such that it maximizes the *a posteriori* probability $P(I|D)$.

Regardless of the probability model chosen to describe the input space, some solutions are implemented using a finite set of linear operations (e.g.[5, 4]). These operations are efficiently implemented using an iterative or closed form scheme. An Example of such an algorithms might be:

1. Interpolate each missing data using a bilinear interpolation of neighboring values from the same color band.
2. Transform the RGB values to chrominance/luminance representation, using e.g. the RGB to YIQ linear transformation.
3. Spatially smooth the chrominance bands (bands I and Q) using Gaussian filtering.
4. Transform back from YIQ to RGB representation using a linear transformation.
5. Reset sampled values to their original values.
6. Iterate again from step 2 several times.
7. Sharpen the result using linear unsharp masking.

The algorithm described above indeed gives very good results after few iterations (typically 3-5 iterations). This algorithm is well understood, and it is efficiently described in an algorithmic manner. Moreover, since each step of this algorithm is a linear operation, the entire demosaicing process is linear. Therefore, the entire algorithm can be implemented efficiently using a convolution operation with filter kernels. The goal of the technique presented here is to find these filter kernels, given the algorithm.

Trivially, it is possible to describe the algorithm mathematically, and find the filter kernels using mathematical rules and derivations. However, this approach is complicated and requires specific mathematical derivations for each given algorithm. The suggested approach is general, and can be applicable for any given algorithm. Moreover, the procedure suggested does not require any elaborated knowledge or description of the algorithm. The algorithm can be regarded as a black box where the input and the associated outputs are available. There are only two requirements that must be satisfied:

1. The algorithm process must be linear, i.e., if $A\{D\}$ represents the results of applying the algorithm A to the input signal D then:
$$A\{D_1 + D_2\} = A\{D_1\} + A\{D_2\} \quad \text{and} \quad A\{\lambda D\} = \lambda A\{D\}$$
2. The algorithm is shift-invariant up to the input pattern repetition. That is, assuming homogeneity of the input data, the demosaicing process is shift-invariant at pixel positions having similar neighborhoods in the input mosaic pattern. The specific Bayer pattern described above has a repetative pattern in blocks of 2×2 (Figure 2). Thus, the demosaicing process of this pattern is shift-invariant, only if the shift is an integer number of blocks. We denote such a system as *block-shift-invariant*. A block-shift-invariant system with a block shift $\mathbf{s}$ satisfies:
$$A\{D(\mathbf{x} - n\mathbf{s})\}(\mathbf{y}) = A\{D(\mathbf{x})\}(\mathbf{y} - n\mathbf{s})$$

for any integer number n, and where, $\mathbf{x}$ and $\mathbf{s}$ are vectors whose dimensions are similar to the signal's domain. Note, that a block-shift-invariant system is not necessarily shift-invariant in the classical manner (see Figure 2).

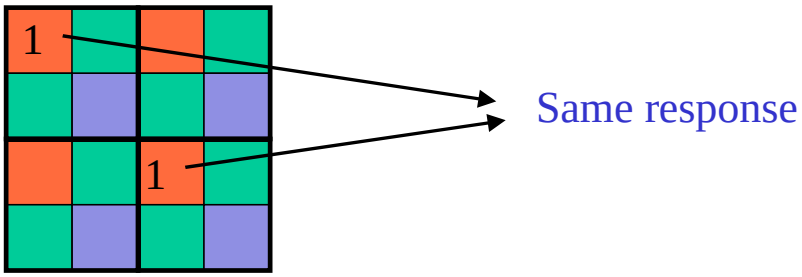

Figure 2: Block-shift-invariant system for the Bayer input image.

1.1 Determining Block-Shift-Invariant Kernels

In principle, a shift-invariant linear process can be implemented using an appropriate linear filter, i.e. a convolution with a filter kernel. In the case of a block-shift invariant linear system, as in the case of the proposed demosaicing process, a linear filtering system can be used as well, however the filtering is based on a set of linear filters rather than a single filter, each of which is associated with a particular location in the block. Thus, in the Bayer pattern depicted in Figure 1, 12 kernels are defined; for 3 different colors and 4 different locations. In this section a technique is described for determining the appropriate set of kernels. For clarity, the classical shift-invariant case is first described, and then extended to describe the block-shift-invariant case.

1.2 Shift-Invariant Case

The appropriate kernel for a given shift-invariant linear algorithm can be extracted by applying the algorithm to an impulse-function input as in Figure 3a (in the case of image filtering, the input image is a zero valued image with a single pixel having unit value). This is known as the *impulse response* of the algorithm. Due to shift-invariance of the algorithm, the response for any shifted impulse is the impulse response shifted by the the same amount (Figure 3b). To compute the kernel values, a single output pixel position is considered (Figure 3c). At that particular pixel location, the collection of impulse responses is determined for all possible impulse inputs (Figure 3d).

Formally speaking, let $\delta(x-p)$ be an impulse input, positioned at location p, and the algorithm response for such an input be $R_p(y)$, i.e.

$$A\left\{\delta(x-p)\right\}(y) = R_p(y)$$

Due to the shift-invariance of the algorithm we have:

$$R_p(p+y) = R_q(q+y) \doteq \hat{R}(y) \quad \forall\ p,q$$

Similarly, the algorithm response to any shifted delta function $\delta(x-k)$ is:

$$A\{\delta(x-k)\}(y) = \hat{R}(y-k)$$

Now, the algorithm output $s_{out}(y)$ for a given signal input $s_{in}(x)$ can be calculated:

$$s_{out}(y) = A\ \{s_{in}(x)\}(y) = A\left\{\sum_k s_{in}(y-k)\delta(x-(y-k))\right\}(y) = \tag{1}$$
$$= \sum_k s_{in}(y-k)A\left\{\delta(x-(y-k))\right\}(y) = \sum_k s_{in}(y-k)\hat{R}(k)$$

where the third equality is due to the linearity of the algorithm. This result implies that the impulse response of a shift-invariant linear algorithm completely characterizes the algorithm; i.e. the output of the algorithm for any input signal can be calculated. This can be viewed as a kernel convolution where the convolution kernel is $\hat{R}(y)$. The question explored in the following section, is whether this property is valid also for block-shift-invariant algorithm, and if so, what are the kernels that should be used in such systems.

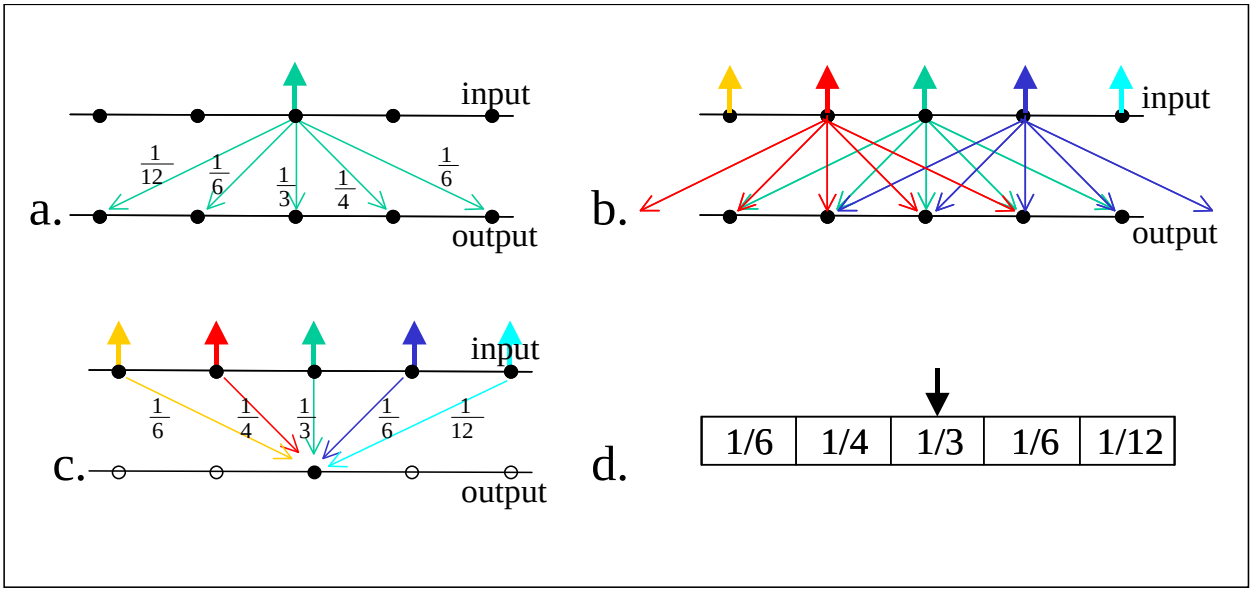

Figure 3: Finding the appropriate filer for a shift-invariant linear system. a) Applying the linear system to a Delta-function input. b) The system output for all possible delta-function inputs. c) A single output pixel position is considered. The system output at that pixel location is determined for all possible delta-function inputs. d) The kernel values are collected from the system outputs for the different delta-function inputs (arrow marks the filter origin).

1.3 Block-Shift-Invariant Case

In a block-shift-invariant linear system, the entire signal domain is tiled by a collection of similar block shapes (see, for example, Figure 4). A canonical block consists of N indexed

grid positions $p_1 \cdots p_N$. Define the function $g(p)$ that maps a general grid position p to its position index, i.e. $g(p) \in \{1, \cdots, N\}$. In Figure 4, for example, $g(p) = g(p + ks_1 + ls_2)$ for any integers k and l. Applying the algorithm to an impulse input located at position p, results in the output response:

$$A\{\delta(x-p)\}(y) = R_p(y)$$

However, due to the block-shift invariance of the algorithm we have:

$$R_p(y+p) = R_q(y+q) \doteq \hat{R}_{g(p)}(y) \quad \text{if } g(p) = g(q)$$

Therefore, we have a set of N different impulse responses: $\hat{R}_i(y)$, $i = 1 \cdots N$, where

$$\hat{R}_i(y) = R_{p_i}(y + p_i)$$

Given this set of N impulse responses, the algorithm output to *any* shifted impulse can be deduced:

$$A\{\delta(x-k)\}(y) = \hat{R}_{g(k)}(y-k)$$

Similar to the classical shift-invariant system, the algorithm output $s_{out}(y)$ for a given signal input $s_{in}(x)$ can be calculated from the set of $\hat{R}_i(y)$:

$$s_{out}(y) = A\ \{s_{in}(x)\}(y) = A\left\{\sum_k s_{in}(y-k)\delta(x-(y-k))\right\}(y) =$$
$$= \sum_k s_{in}(y-k) A\left\{\delta(x-(y-k))\right\}(y) = \sum_k s_{in}(y-k)\hat{R}_{g(y-k)}(k)$$

The last equation states that, similar to the classical shift-invariant systems, the set of impulse-responses of the block-shift-invariant algorithm $\hat{R}_i(k)$ entirely characterizes the algorithm, i.e. the output of the algorithm for any given input signal can be calculated. However, due to block-shift-invariance, the algorithm output is calculated similarly only for those positions y having the same position index $g(y)$. Therefore, there are N different convolution kernels $H_i(k)$, $i = 1..N$, where:

$$H_{g(y)}(k) \doteq \hat{R}_{g(y-k)}(k)$$

To calculate the algorithm output at position y, the kernel $H_{g(y)}(k)$ is applied in the following manner:

$$s_{out}(y) = \sum_k s_{in}(y-k) H_{g(y)}(k)$$

As explained above, the number of kernels required is equal to the size of a block. Figure 5 shows a 1D example where the block size is two. In this case, 2 filters are required to implement the linear system. The filters can be found by applying the algorithm to two impulse inputs as in Figure 5a. To compute the filter kernel values at a single output pixel position, the collection of system outputs at that pixel location is determined for all possible impulse inputs (Figure 5b).

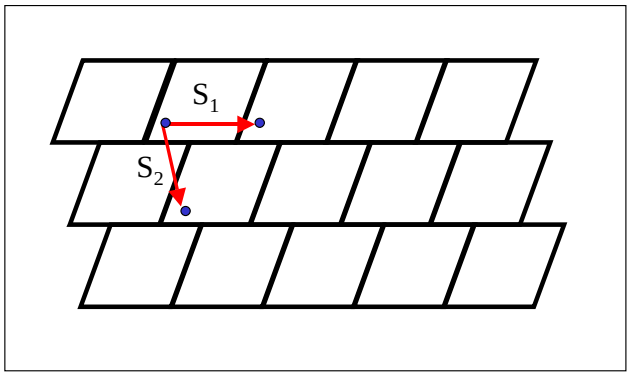

Figure 4: A signal domain is tiled by a collection of similar shaped blocks. The shift from a pixel position p_1 to a similar block-relative position p_2 can be expressed as $p_2 = p_1 + ks_1 + ls_2$ where k and l are integers.

1.4 Block Shift-invariant Kernels for Demosaicing

As described above, the Demosaicing algorithm is a block-shift invariant linear system applied on a 2D Bayer Pattern which is 2×2 repetitive. Thus, the block size of the linear system is 2×2. The input is a single 2D array of sensor inputs. The output of the process consists of three 2D arrays representing the Red, Green and Blue contents of a color image. Thus, in practice. the Demosaicing process can be viewed as 3 independent block-shift invariant linear systems (receiving the same input and producing the R, G or B arrays). Following the discussion above, the Demosaicing process can be implemented using 12 filters (3 filters for the R, G and B outputs for each of the 2×2 block positions (see Figure 6).

2 Demonstrated Results

In order to demonstrate the proposed technique, two demosaicing algorithms were considered. The first algorithm, is the trivial bilinear interpolation for the Bayer pattern. The 12 kernels that were generated by the proposed technique, are shown in Figure 7. In the figure each row has 4 kernels, for block positions $(1,1), (1,2), (2,1)$, and $(2,2)$, from left to right. The rows show the kernels for the Red, Green, and Blue images, from top to bottom. The second algorithm considered, is that given in Section 4. The kernels that were generated for this algorithm are shown in Figure 8, in the same order. These kernels were applied to a mosaiced image that is shown in Figure 9a. Figure 9b shows the resulting image after applying the bilinear kernels. Figure 9c is the result after applying the kernels generated for the second algorithm.

3 Conclusions

A technique for generating the appropriate kernels for block-shift-invariant algorithms was proposed. These kernels can be calculated without specific knowledge about the algorithm, i.e the algorithm can be regarded as a black-box. The advantage of the proposed technique is that the kernel derivation is automatic and does not require complicated mathematical derivations. A hardware device that can apply convolution kernels to an image, can be adjusted efficiently to perform any proposed block-shift-invariant and linear algorithm.

Acknowledgements

The author would like to thank Renato Keshet for his assistance in writing this paper and for helpful discussions.

4 References

[1] D. R. Cok, *Reconstruction of CCD images using template matching*, Proc. IS&T's Annu. Conf. ICPS, 1994, pp. 380–385.

[2] Yacov Hel-Or and Daniel Keren, *Demosaicing of color images using steerable wavelets*, Tech. Report HPL-97-104, HP Labs, August 1997.

[3] R. Kimmel, *Demosaicing: Image reconstruction from color CCD samples*, EVVC (1), 1998, pp. 610–622.

[4] D. Taubman, *Generalized wiener reconstruction of images from colour sensor data using a scale invariant prior*, Proc. of the International Conference of Image Processing, 2000.

[5] D. Taubman, *Generalized image demosaicing and enhancement (GIDE)*, Tech. Report HPL-97-104, HP Labs (internal), Vancouver, 1996.

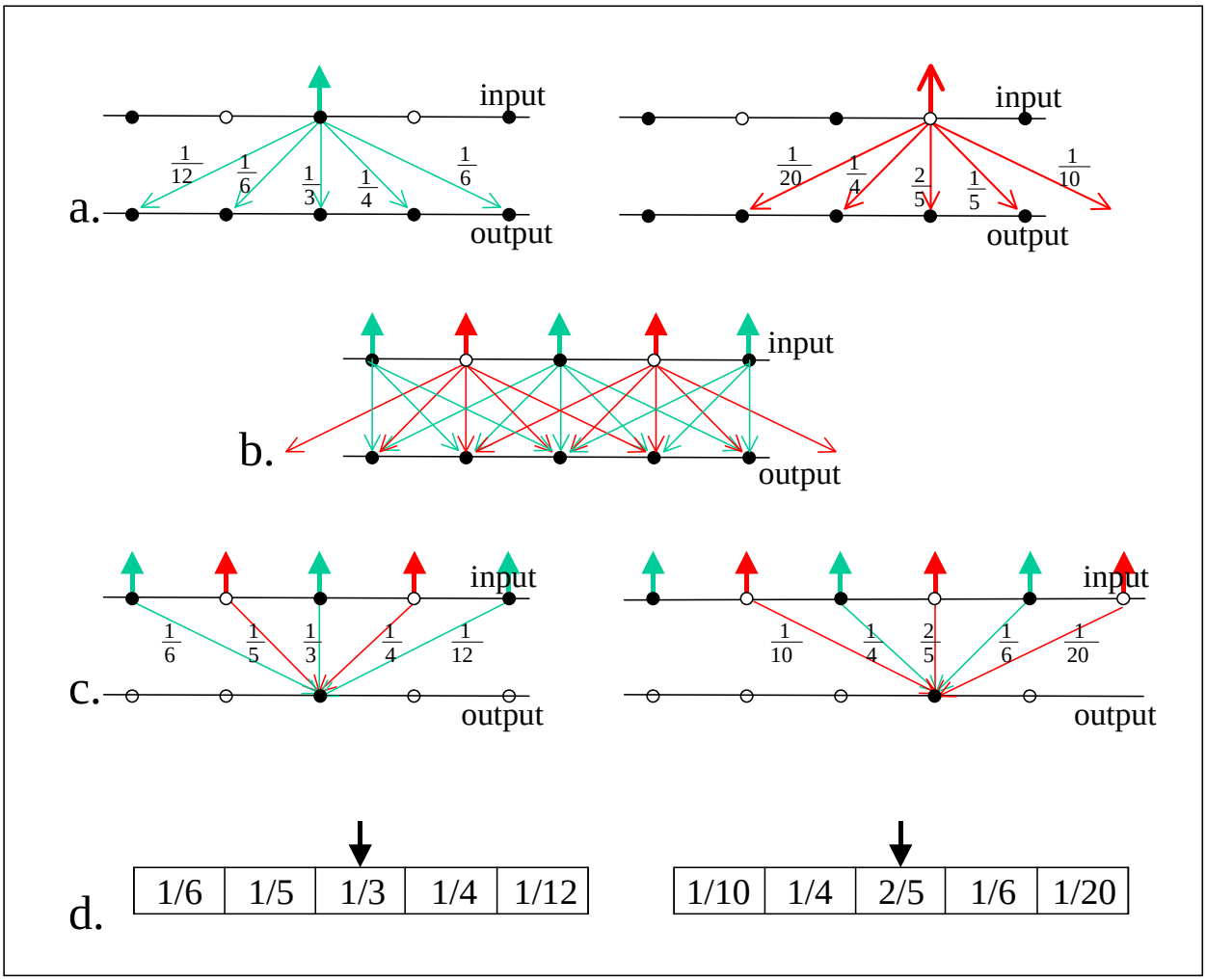

Figure 5: Finding the appropriate filter for a shift-invariant linear filter. a) Applying the linear system to an impulse input. b) The system output for all possible impulse inputs. c) A single output pixel position is considered. The system output at that pixel location is determined for all possible impulse inputs. d) The filter kernel values are equal to the system response for the different impulse inputs (arrow marks the filter origin).

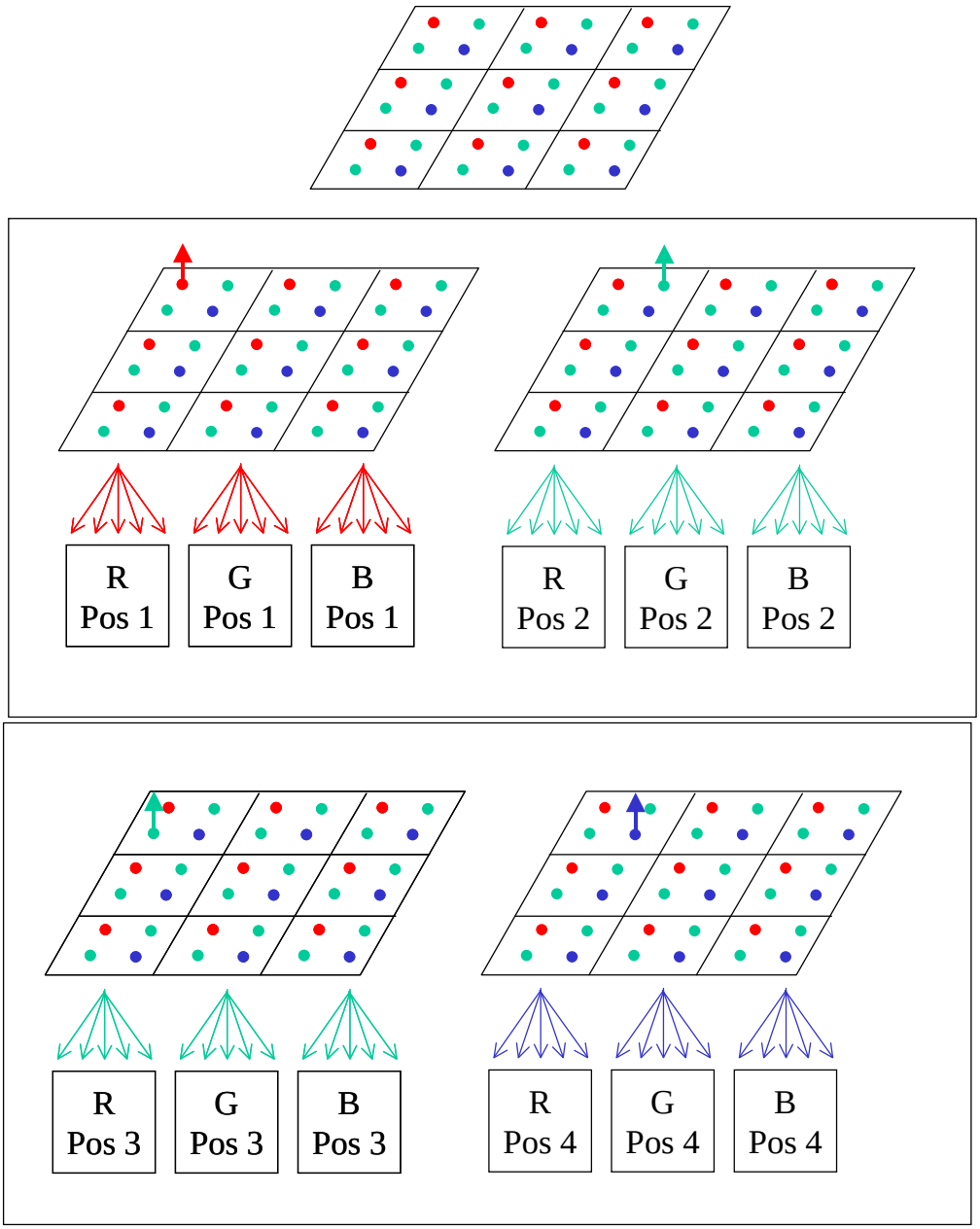

Figure 6: The Bayer pattern in a digitized image consists of a repetitive 2×2 patterns. Using a linear demosaicing algorithm, each impulse input produces 3 output images: Red, Green, and Blue. Therefore, for each output image, 4 kernels are required, and a total of 12 kernels are needed for the 3 output images.

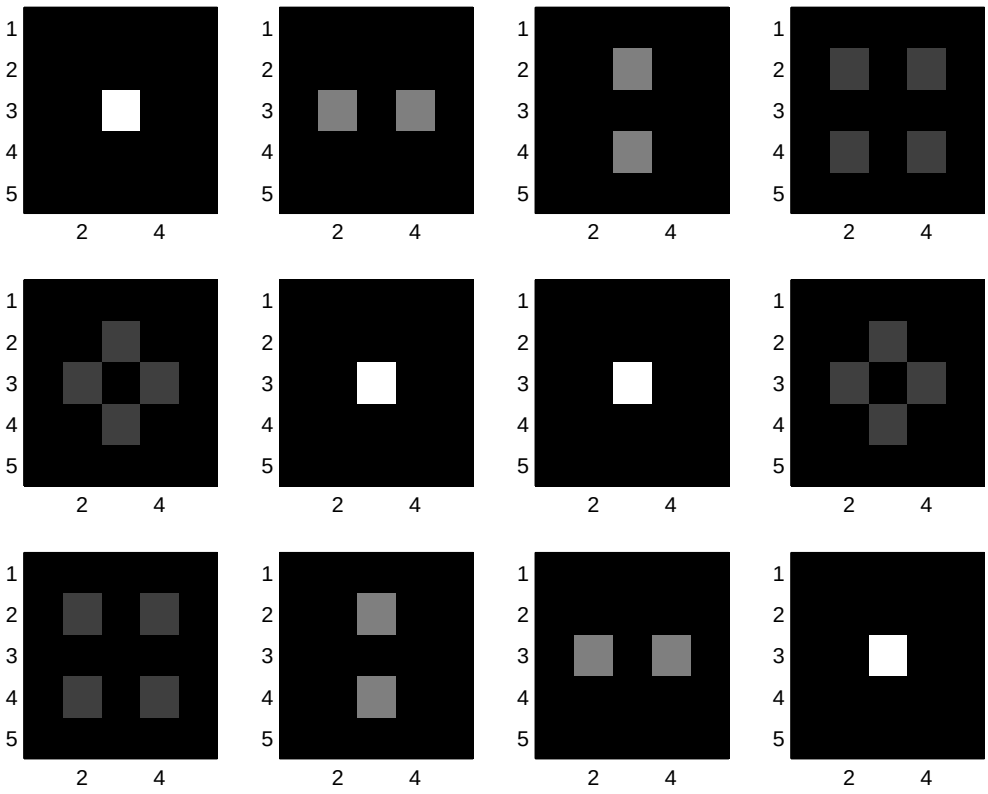

Figure 7: The kernels that were calculated for a bilinear algorithm. At each row 4 kernels are given, for block positions (left to right) $(1,1),(1,2),(2,1)$, and $(2,2)$. The rows show the kernels for (from top to bottom) the Red, Green, and Blue images.

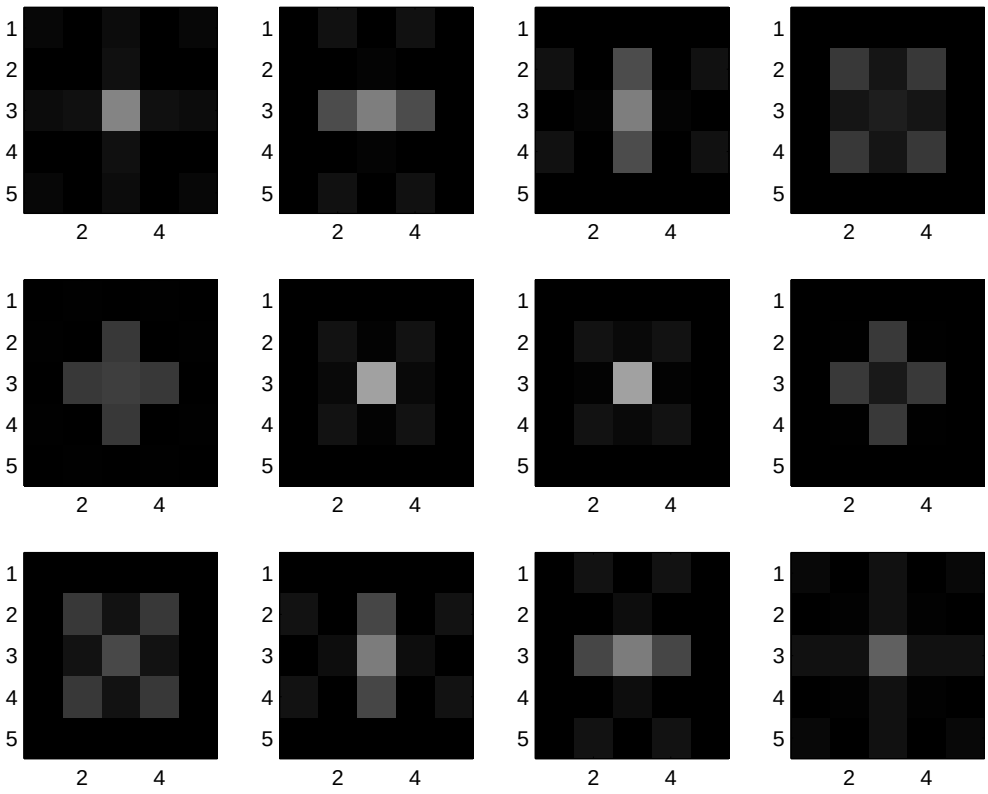

Figure 8: The kernels that were calculated for the demosaicing algorithm proposed in Section 4. At each row 4 kernels are given, for block positions (left to right) $(1,1),(1,2),(2,1)$, and $(2,2)$. The rows show the kernels for (from top to bottom) the Red, Green, and Blue images.

a.

b.

c.

Figure 9: a: The original image from which a mosaiced image was sampled. b: After applying the bilinear kernels to the mosaiced image. c: After applying the kernels of the second algorithm