FPGA Implementation of an Anonymization Algorithm

Niklas Kjellman, Rehenuma Tarannum

Axis Communications

Supervisor: Daniel Falk (Axis Communications)
Joachim Rodrigues (Lund University)

Examiner: Pietro Andreani (Lund University)

2020/06/26

Printed in Sweden
Tryckeriet i E-huset, Lund

Abstract

With the large amount of surveillance cameras in our public spaces there has been much discussion about their effects on privacy. *Axis communications* has an algorithm that tries to remedy this by making it hard to see who is in an image but still have it possible for a neural network to detect that there is a person present. This thesis explores how this algorithm needs to be adapted to be implementable on an FPGA that operates on the image stream directly from the sensor. Resulting in an anonymization that is much harder to turn off or bypass. The anonymization is performed by blurring the image with a box filter and edges are highlighted with a Sobel operator. These operations is performed in parallel and the edges from the Sobel operator is overlaid the blurred image from the box filter. The alterations made to the algorithm is tested in regards to how they affect the performance of *Axis communications* person detector.

The detector requires the images to look similar to the result from the reference algorithm. The implementation then need to replicate parts of the Image Processing pipeline in the camera system and its inverse. This is done by adding a demosaic and remosaic stage before and after the algorithm, so that both the in and out image is on Bayer form but the anonymization is performed on images in RGB form. The change from floating point calculations to fixed point arithmetic is also done to improve how implementable the design would be and to reduce area consumption. Down sampling is suggested and implemented, to reduce the size of the largest block the box filter. The amount of decimal points needed in the overlaying stage is also tested and the block is simplified. The problem of not retraining the detector is discussed and some solution that could improve hardware utilization without loosing detector performance is suggested.

Acknowledgments

Looking back at this project there are many people who have contributed to make it happen. First of all we would like to thank our supervisor at Axis, Daniel Falk for his guidance, positive energy and tireless dedication. Secondly we would like to thank Andreas Karlsson for his help with the FPGA and answering our every question regarding digital hardware. We would also like to thank our supervisor at Lund University Joachim Rodrigues, for his valuable input. Lastly we would like express a more general thanks to anyone else who have been involved in our thesis in any way, *Axis communication* in general, our friends and families.

Acronyms, Notation and Symbols

Acronyms

AMBA-AXI	**A**dvanced **M**icro **C**ontroller **A**rchitecture-**A**dvanced e**X**tensible **I**nterface: Bus architecture developed by *arm*.
ASIC	**A**pplication **S**pecific **I**ntegrated **C**ircuit: Integrated Circuit that is designed for a specific task.
B-RAM	**B**lock-**R**andom **A**ccess **M**emory: Dedicated RAM cells in and FPGA with a fixed capacity.
CPU	**C**entral **P**rocessing **U**nit: Processor that process the main instructions of a system.
CNN	**C**onvolutional **N**eural **N**etwork: A deep learning based neural network often used in object detectors.
DSP	**D**igital **S**ignal **P**rocessor: Hardware device-purpose built to speed up specific types of calculations.
FF	**F**lip **F**lop: A digital register.
FIFO	**F**irst **I**n, **F**irst **O**ut: A type of data queue.
FPGA	**F**ield **P**rogrammable **G**ate **A**rray: Hardware platform that can be re-programmed in the field.
HW	**H**ard**W**are: Specifically digital hardware, either in FPGA or ASIC form.
GDPR	**G**eneral **D**ata **P**rotection **R**egulation: Law of the European Union that regulates the handling of personal data.
IP	**I**ntelectual **P**roperty: In this context it refers to hardware blocks that have been developed by third parties and made available for use.
IPP	**I**mage **P**rocessing **P**ipeline: All of the processing done on an image in an Axis camer before it is ready to be viewed.
LUT	**L**ook **U**p **T**able: Array that hold pre-computed values. Can in FPGAs be used to model logic.

LUT-RAM	**L**ook **U**p **T**able-**R**andom **A**ccess **M**emory: Distributed RAM blocks, used to make a memory whose capacity is set during synthesis.
MUX	**Mu**ltiple**X**er: Block that selects output from a selection of inputs and a control signal.
RAM	**R**andom **A**ccess **M**emory: Hardware device that store data in a volatile way.
RGB	**R**ed, **G**reen, **B**lue: Refers to the RGB color model.

Notation

$0xFA23$	A hexadecimal number.
$10\,1010_2$	A binary number.
x_{16}	Variable x with 16 as subscript.
$f_1(x)$	Function f with subscript 1 and argument x.
$\mathbf{v}$	The vector $\mathbf{v}$. **Note** lower case.
$\mathbf{M}$	The matrix $\mathbf{M}$. **Note** upper case.
F	Variable F, **Note** not bold.
Δf	The differential of the function f.
$\nabla \mathbf{f}$	Vector differential operator or vector gradient of f.
$\mathbb{R}^N$	The set of real value numbers in N dimensional vector space.
$\|\|\mathbf{v}\|\|_x$	Magnitude calculation using the x norm.
$\lfloor x \rfloor$	The variable x is rounded down to the closest integer.
$\lceil x \rceil$	The variable x is rounded up to the closest integer.
$1 << 32$	The number 1 is bit shifted 32 bits to the left.
$1 >> 32$	The number 1 is bit shifted 32 bits to the right.

Popular Science Summary

A privacy friendly camera system

Despite of the wide use of surveillance camera for security purpose, over the time it has turned more controversial as a threat to privacy. If a privacy breach is abused it can even threaten people's security.

In our daily life we are constantly close to a camera. We use them in our phones and laptops to communicate and record memories, or in the watchful eye of a surveillance cameras to improve our security. In recent years a new usage of cameras have been popularized, where they are used as sensors in a computer-vision system. These can for instance, be used for counting the lengths of queues or detecting how many people enters and exits a room. With these sensor cameras, however, it is not always strictly necessary to be able to recognize a person, instead only to observe their presence. This thesis have aimed to improve and safeguard the privacy of the people being observed, the video stream from the camera is anonymized, which signifies the removal of details and feature of the person. This was done by blurring the entire image in the stream and re-highlighting the strongest edges. The blur is performing the anonymization by hiding the finer features of a person. The edges is re-highlighted to make it easier for a computer vision system to detect a person. This idea was implemented on an FPGA. An FPGA is a Digital hardware circuit and is therefore harder to re-configure compared to if it was implemented in software. The FPGA implementation is therefore a stronger anonymization, that people can trust as it will be hard to deactivate. This have the benefit of making it easier to have a computer vision system that complies with the European GDPR law, or that would make it more likely to get the required permissions to install a stationary camera.

When developing the anonymization filter it was discovered that the filter that adds the blur is a very large operation and requires a lot of hardware to be implemented. The level of blurring and the width of the image was the factor that scaled the hardware utilization the most. A solution for this, is to perform downsampling around the block. Downsampling is a method were the number

of data points are reduced. The amount of hardware reduction was proportional to fraction of an image that was skipped.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

Surveillance cameras, or sometimes known as CCTV(close circuit television) are being used for monitoring public and private spaces and observing people. It was first utilized in 1942 during world war two by German scientists to monitor rocket launches. CCTV used to only be connected to a local network system for the broadcasting of its video. The video was therefore limited over distance and who could see it. With the development of technology, the CCTV Cameras started to interconnect through IP-network systems. This made surveillance lower in cost and could be accessed from any part of the world with the correct access right. The amelioration in surveillance boosts up the security but also added other complexity like threats to privacy. The thesis is constructed aiming of privacy friendly camera that transforms the regular image into a indistinct image in hardware to safe guard the transformation. The following sections in this chapter states the background of the thesis along with its purpose and structure of conducting research.

1.1 Motivation

The advancement in surveillance cameras made it prevalent in various sectors like crime prevention, traffic control, queue monitoring, industrial process monitoring, and many more. The extensive use of cameras is therefore responsible for capturing a large amount of identifiable information and details. If this data is abused then it can lead to a grave infringement of privacy as it is no longer a ”closed circuit”. Therefore the legislators in many countries have enacted strict regulations and require anyone who wants to install a surveillance camera to seek permission. Moreover, in Europe, the General Data Protection Regulation known as GDPR is one of the strictest privacy and security laws in the whole world. This regulation makes it mandatory for any company to maintain the privacy of data concerning people living in the European Union. This puts limitations on video surveillance for use-cases where the individual's recognizable features are not important. For instance in a queue monitor or when counting how many people have entered and exited a room. For these applications, it is enough to see that a person is present and what he or she is doing. The camera can be utilized more like a sensor for an

analytic system as the identity of the person mostly likely irrelevant. This case is getting more and more common nowadays. Anonymization in these use-cases would make it easier to comply with GDPR and improve the privacy of those in view of the camera.

This concern is substantial to the network surveillance camera producer *Axis Communications* (often referred as *Axis*). They are pioneers in network-attached products and the world's largest supplier of networked surveillance cameras. *Axis* developed an anonymization algorithm for use in their cameras. In the technological field, the act of anonymization on personal data means to sanitize recognizable features of a person so that the person can no longer be identified. This can especially be applied to images where the data to be made anonymous, consists of a person's recognizable features. They want to utilize this anonymized camera as an analytic camera that is more privacy friendly. To do so, they developed their in house detector that can detect people on an anonymized image stream. In *Axis* algorithm the anonymization was performed on a CPU. However, there is a problem when executing it on the CPU. The transformation of the image can be turned off by anyone with root access to the camera and its Linux system. It may put the anonymization in question. This is what the thesis aims to solve.

The focus of the thesis is to implement the algorithm in hardware so that the image conversion is done before reaching the CPU, which would make the transform much harder to disable or bypass. It would lead to a more strongly anonymized camera. To make the anonymization irretrievable hardware implementation is more feasible compared to alternative solutions like using a dedicated microprocessor. However, the requirement of a frame rate of 30 frames per second and a resolution of 1920 by 1080 would make the bit rate very high, which would not be suitable for the microprocessor to process efficiently. In contrast, a hardware implementation can perform it much faster. An ASIC would be the fastest a but they are very cumbersome and expensive to develop. As a prototype an FPGA implementation is preferable. Hence, for this thesis, the anonymization will be implemented on an FPGA. In addition, different hardware optimizations is explored in order to achieve a suitable implementation without compromising the detector's performance noticeably.

1.2 Objective

The main goal with the thesis is to: *Illuminate how hardware requirements and limitations will affect Axis' anonymization algorithm and the implementation of it.* To do this a number of sub-objectives have been defined.

1. Implement the predefined anonymization algorithm on an FPGA.
2. Iteratively communicate what parts of the algorithm that are restraining the hardware implementation and how the implementation restrains the algorithm. Based on the analysis, new suggestions and changes will be incorporated in the algorithm as well as in the hardware design.

3. Evaluate the accuracy loss of a deep-learning-based people detection algorithm (supplied by *Axis*), caused by the hardware implementation. It should be compared to the reference algorithm. This will be done by taking a set of images where the location of all the people in it is known and let the detector find people. The comparison between the detected and the actual people will be used to evaluate the performance of an algorithm.

1.3 Thesis Structure

The thesis follows the following chapters framework and their content is described below:

- **Introduction** Introduction chapter address the problem that this thesis aims to solve.
- **Reference Algorithm** This chapter briefly describes the operations and general mathematical approaches in the existing reference algorithm.
- **Detector Evaluation Process** Provides brief background of the computer vision and detection evaluation matrices. It also includes the detection methodology used in this thesis.
- **Required Adaptation of the Algorithm** Overview of adaptations that had to be made to make the algorithm implementable in hardware and still emulate the effects of the reference model.
- **Implementation** Describes how the algorithm is implemented in hardware. The implementation described in this chapter will be called *First Implementation*. The general design of each block and FPGA utilization of the block is discussed here.
- **Algorithmic Adjustments** This chapter focus on observing, describing and evaluating adjustments that can be made to the algorithm to improve hardware utilization. Some of the suggested improvements are also implemented and their final utilization is compared with the original.
- **Conclusion** The overall experience of the thesis and proposals to future work is discussed.

Chapter 2

Reference Algorithm

The anonymization algorithm, this thesis builds on was written at *Axis communications*. The chapter illustrates relevant background of image processing that the algorithim utilized. It accomplished with three main parts. A Sobel operator (also called Sobel filter), a box filter, and a merge function. The Sobel operator detects edges and highlight these (see more in section 2.2.2). The box filter blurs the image and obfuscates any recognizable features (see section 2.2.1). Lastly, the merge function blends the result from the two previous filters into a single image (see section 2.3). A simple block diagram of how the data flows through the algorithm and its components can be seen in Figure 2.1. The algorithm was workable for the RGB image color model and to execute the functionality of Sobel operator and box filter convolution was used.

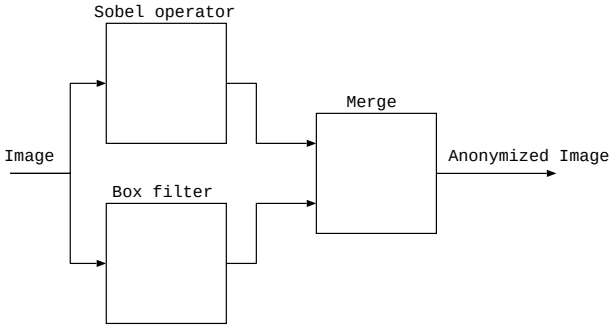

Figure 2.1: Block diagram of how the data flows through the reference algorithm and its major components.

2.1 Images and Color Representation

The most simple form when model an image digitally is to assume that all light have no specific color. Then a pixel in the image only stores the intensity or luminance of the light reaching the sensor in one small point. The resulting image

or frame in a video stream will look gray to a human. It is a simple representation and only one value is needed to be stored per pixel [1]. This model will be called a grayscale image.

There are many options available to represent color. The most common [1] and the one used in this thesis is the RGB model. In it, each pixel contains three luminance values to represent the light of colors; red (R), green (G), and blue (B). These three build a Cartesian coordinate system that contains most of the visible colors [1].

To transfer an RGB color model image to a grayscale image, the three luminance values need to be weighted together [2]. The library *openCV* (from `opencv.org`) uses the following transform from $\mathbb{R}^3 \rightarrow \mathbb{R}^1$ to scale the values to better represent the human eyes ability to see different colors [1].

$$Gray\,pixel = Red \cdot 0.299 + Green \cdot 0.587 + Blue \cdot 0.114 \tag{2.1}$$

Humans are best at observing green and its weight is therefore the largest [1]. Blue is less observed and therefore the smallest [1]. The function is applied to each RGB pixel in an image to make a grayscale image.

2.2 Spatial Filters and Convolutions

Spatial filters aim to preform filtering on a pixel in regards to its neighborhood and a weight matrix [1]. The weight matrix is often called *kernel* or mask [1], but for this thesis kernel will mostly be used. A general kernel $\mathbf{G}$ can be seen in the following Equation 2.2, where $\omega_{(x,y)}$ are the kernel weights, n is the number of rows and m is the number of columns in the matrix. The sum of all of the weights equal to $n * m$, the division is meant to normalize the kernel so that the element sum of $\mathbf{G}$ equal to 1 and this would ensure that the average pixel brightness is retained [1].

$$\mathbf{G} = \frac{1}{n \cdot m} \begin{bmatrix} \omega_{(0,0)} & \omega_{(0,1)} & \cdots & \omega_{(0,m-1)} \\ \omega_{(1,0)} & \cdot & \cdots & \omega_{(1,m-1)} \\ \cdot & & & \cdot \\ \cdot & & & \cdot \\ \cdot & & & \cdot \\ \omega_{(n-1,0)} & \cdot & \cdots & \omega_{(n-1,m-1)} \end{bmatrix} \tag{2.2}$$

The kernel $\mathbf{G}$ is used by sliding it over the image and element-wise multiplying the weights with the values in the image and take the sum of the result. A general function on how to calculate a filtered pixel $g(x, y)$ can be seen in the following Equation 2.3 and 2.4, where $f(x, y)$ is the pixel to be filtered, $\omega_{(s,t)}$ is the kernel values from Matrix 2.2. $a = \lfloor \frac{n}{2} \rfloor$, $b = \lfloor \frac{m}{2} \rfloor$ and both n and m are odd:

$$g(x, y) = \frac{1}{n \cdot m} \sum_{s=-a}^{a} \sum_{t=-b}^{+b} \omega_{(s,t)} f(x + s, y + t) \tag{2.3}$$

If n and m are even:

$$g(x,y) = \frac{1}{n \cdot m} \sum_{s=-a+1}^{a} \sum_{t=-b+1}^{+b} \omega_{(s,t)} f(x+s, y+t) \tag{2.4}$$

These two calculations is very similar to the concept of convolutions [1] and is only valid for a one or two-dimensional space. So it either needs to operate on a grayscale image or on all three color planes in an RGB image in parallel, so that Equation 2.4 and 2.3 only would calculate $g_{red}(x,y)$, $g_{green}(x,y)$ or $g_{blue}(x,y)$. At the edges when the kernel covers indexes outside of the image, the pixel values need to be estimated. There are many methods of extrapolating around the edges, but the simplest is to just assume that those values are zero [1]. This method is called zero padding [1]. Using it will make the rim of the image darker but with an image resolution that is much larger than the kernel size, the effect is negligible.

2.2.1 Box Filter

To smooth out color transitions in an image and remove high frequency content in the spatial domain, a low pass filter can be used [1]. A simple low pass filter is an averaging filter, often called a *box filter*. It takes an average of the values around a pixel and uses that as the new pixel value to get a smoother and blurrier image [1]. It is a spatial filter with all of the weights in Matrix 2.2 set to 1. The filter that Rafael and Richard [1] describes is over the values in a grayscale image. Nevertheless, the filter can be applied to all three color planes separately to have the same effect.

2.2.2 Sobel Operator

An edge in an image is a location where there is a change in the luminance values [1]. The most ideal edge is two mono-colored fields that are located next to each other. In real images there is generally more of a gradual transition over some pixel widths worth of space. What both of these examples have in common is that the color gradient is non zero at this transition and zero for the rest of the image. For the ideal transition, the gradient is only non-zero at the thin edge, whereas for the gradual transition, the gradient will be non zero for the entire edge. For an example of a straight edge and its gradient see Figure 2.2. A gradient edge and its gradient can be seen in Figure 2.3. For the gradients, a higher magnitude is indicated with a darker color.

To calculate the gradient of the function f, either the right or the left derivative [3] can be estimated. In the discrete plane the left can be as closely as possible estimated with $\frac{\Delta f_x^{left}}{\Delta x} = \frac{f(x)-f(x-1)}{\Delta x}$ and the right $\frac{\Delta f_x^{right}}{\Delta x} = \frac{f(x)-f(x+1)}{\Delta x}$. However, both of these estimations are biased in a specific direction. The mean of the two are un-biased and therefore a better estimation. It is calculated with the symmetric derivative [4]:

Figure 2.2: The top field is how an ideal straight edge looks. The lower its gradient. The transition is immediate between two mono-colored fields next to each other.

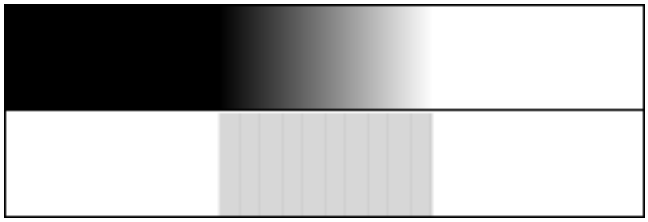

Figure 2.3: The top filed is how a more realistic edge looks. The lower is its gradient. It generally have a gradual transition between two mono-colored fields next to each other.

$$\frac{\Delta f_x^{sym}}{\Delta x} = \frac{f(x+1) - f(x-1)}{\Delta x} \tag{2.5}$$

Gradient estimations can be performed with spatial filters. The following two kernels [5] calculates the gradients in Equation 2.5. The resulting index is in the middle of the kernel and the first kernel estimates the gradient in the x direction:

$$\mathbf{G}_x = \begin{bmatrix} -1 & 0 & 1 \end{bmatrix} \tag{2.6}$$

And the second kernel is in the y direction:

$$\mathbf{G}_y = \begin{bmatrix} -1 \\ 0 \\ 1 \end{bmatrix} \tag{2.7}$$

OpenCV describes these two Kernels (2.7 and 2.6) as a Sobel operator [5]. In other literature the kernels that are generally mentioned as Sobel operators are the following two kernels:

$$\mathbf{G}_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \tag{2.8}$$

$$\mathbf{G}_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix} \tag{2.9}$$

These kernels also achieve some smoothing and noise reduction, as well as finding the gradient [1].

If an edge is parallel to a gradient kernel it will not be discovered by that gradient kernel, since there is no change in luminance in that direction. For instance the y-gradient kernel would not find the edges in Figure 2.3 or 2.2. It is therefore useful to use both the x and y gradients in a two dimensional vector gradient and take the magnitude of them [1]. The following Equation 2.10 describes how the two dimensional gradient $\nabla\mathbf{f}$ should be assigned and its magnitude ∇f:

$$\nabla\mathbf{f} = \begin{bmatrix} \Delta f_x \\ \Delta f_y \end{bmatrix}, \ \nabla f = ||\nabla\mathbf{f}|| \tag{2.10}$$

When calculating the magnitude of a two dimensional vector $\mathbf{v}$, the distance that the elements x and y build is the hypotenuse between point A and C in Figure 2.4 [6]. The most exact way of calculating this is with the *euclidean norm* (or 2-norm) [6]. The two-norm of the vector $\mathbf{v}$ can be expressed mathematically as below:

$$||\mathbf{v}||_2 = \sqrt{x^2 + y^2} \tag{2.11}$$

In some cases the 1-norm or the ∞-*norm* [6] can be used to trade accuracy for reduced computational complexity. The 1-norm simply adds up the distances of x and y together [6]. It will lead to a larger magnitude then sought after unless one of the distance is 0, which will give an exact result. This can very intuitively be seen in Figure 2.4. It has the following Equation:

$$||\mathbf{v}||_1 = |x| + |y| \tag{2.12}$$

The ∞-norm only takes the maximum value of the two distances [6]. So it will generally be shorter then the other two (see Figure 2.4), unless one of x or y is 0, which will also lead to an exact result.

$$||\mathbf{v}||_\infty = max(|x|, |y|) \tag{2.13}$$

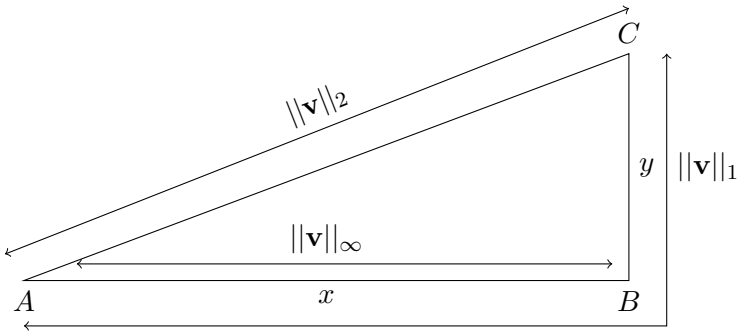

Figure 2.4: Triangle used to demonstrate norm calculations of the vector $\mathbf{v} = [x, y]$.

2.3 Merging the Two Images

The merge overlays the Sobel edges on top of the filtered image. It does this by using the Sobel image as an alpha channel and perform an alpha scaling between the box filtered image and an image that is completely white, where white means that all pixels have its maximum value assigned to it. For instance, if a regular 8-bit RGB image is used then $max\,bit\,value = 2^8 - 1 = 255 = 0xFF$. The alpha channel is deciding how opaque [7] the box filtered image will be. The opaqueness is used to make the filtered image transparent when an edge is detected and firm when there is no edge. This will keep the color of the filtered image but the edges will be highlighted in white. Alpha scaling in one pixel follows the following Equation 2.14 :

$$pixel_{out} = (1 - \alpha) box\ filter\ pixel + \alpha * max\,bit\,value \tag{2.14}$$

The α value is calculated by $\alpha = \frac{Sobel\,pixel}{max\,bit\,value}$, which will make the α value to be between 0 and 1 [7]. Simplifying the Equation 2.14 leads to the following Equation 2.15 :

$$pixel_{out} = (1 - \frac{Sobel\,pixel}{max\,bit\,value}) box\ filter\ pixel + Sobel\,pixel \tag{2.15}$$

This Equation 2.15 eliminates the need for the white image, and the Sobel edges can be used directly. Since the Sobel image is a grayscale image and the box filtered one is a RGB, the edges need to be assigned equally to all three color planes. The resulting image is therefore in RGB form.

2.4 Configuration

The reference algorithm was configured to use RGB images as in- and output, with 8 bits per color in a pixel. It utilized the *openCV* functions to perform the filtering

and image color model conversion. The Sobel filter expected a grayscale image as input and it performed the color conversion as described in the Equation 2.1. The Sobel used the gradient kernels in Equation 2.7 and 2.6 and for the magnitude calculation of gradient vectors, the euclidean norm was chosen in the *openCV* function. For the box filter, a kernel size of $30x30$ was specified to optimize the blur to people 6 meters away from the sensor.

Chapter 3

Detector Evaluation Process

Apart from the implementation of the anonymization algorithm, this thesis focuses on an analogy between the hardware implementation and the reference algorithm both from a human vision and computer vision perspective. Human vision comparison is made later on the implementation chapter 5 that shows the visual comparison of the hardware implantation in contrary to the software implementation of the reference algorithm. Computer vision is an artificial intelligence that enables computers to identify, measure and classify objects in a digital image or video. It discerns the object as well as processes the information and response of what it sees. Detection is one of the most important use cases of it. A detector uses different computer vision techniques to determine the presence of a particular item in the image or video from a common generic set of objects that it is trained for (for example a human, car or a dog). The detector used in this thesis builds on a deep learning technique with a convolutional neural network to perform the detection. For better understanding the performance of a detector, this chapter explains necessary concepts in evaluation matrices and a brief methodology to perform it. Improving this detection algorithm is beyond the scope of this thesis. However, comparing the detector's performances between the reference algorithm and the hardware-implemented algorithm are noteworthy.

3.1 Evaluation Parameters

In general, no system is irreproachable and it is always hard to determine in a concrete way how good or bad a system is. Several aspects need to be considered when it comes to measuring the performance of a detector. Therefore the evaluation depends on what use-cases the detector is used for [8]. Several evaluation matrices are being used to analyze performance. For detection, a set of images with annotated objects are introduced to the system to train it. A test data-set that is not part of the training set is then run by the system to observe how well it can detect objects. To visualize the performance, a binary output representation of detection and annotated objects are considered. It is represented below in a $2x2$ matrix called "confusion matrix" (see Figure 3.1). It shows all four possible

outcomes of a detection mentioned in the "Computer and Information Security Handbook" [9].

- True Positive (TP): Detection and actual case both are positive. A person is present and detected. So, the detection is correct.
- True Negative (TN): Detection case is negative when the actual case is also negative. It means the detection says no person is present and it is accurate. The detector used in this thesis did not calculate the true negative. Because, all of the pixels that do not consist of a person and the detector acknowledges it correctly, would be considered as a TN. In general, the number of pixels that do not contain a person by far surpasses the number of pixels with a person. Therefore, the quantity of TN cases would be very high which can mislead the evaluation metrics. To avoid this phenomena TN was ignored.
- False Negative(FN): The detected case is negative, which implies no person is detected whereas the actual case says positive. It means the prediction is inaccurate.
- False Positive(FP): The actual case contains no person where the detector detects a person.

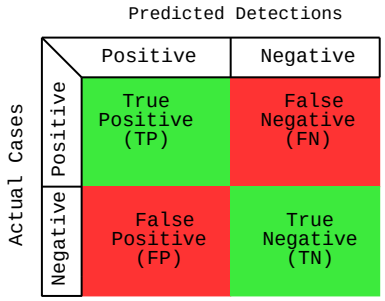

Figure 3.1: Confusion matrix

Using the confusion matrix, different evaluation matrices can be formulated. As True negative cases are not well defined the evaluation matrices used by *Axis* detector are briefly discussed below based on the reference of "Computer and Information Security Handbook" in chapter six [9]:

- Precision: It describes the accuracy of detection among all detected instances. A detector that does not use more detections then it is certain to be correct will have a higher precision. A detector that finds less TP can have higher precision than one that finds more, but that also finds more FP. High precision might not mean that it is finding everything. In Figure 3.2 it can be visualized with the amount in the overlap out of the right circle.

$$Precision = \frac{TP}{TP + FP} \tag{3.1}$$

- Recall: It provides information on how accurate the detection is out of total relevant instances. A detector that finds as many TP as possible will have a high recall. Even if it also finds many FP. A High recall value might not mean that it is finding the right objects. In Figure 3.2 it can be visualized with the amount in the overlap out of the left circle.

$$Recall = \frac{TP}{TP + FN} \tag{3.2}$$

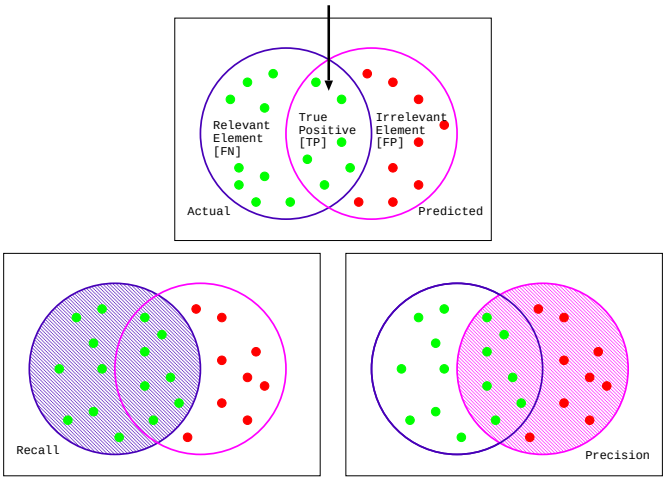

Figure 3.2: Visualization of precision and recall

- F1-Score: It gives an idea of detection accuracy combining precision and recall. A detector with both high recall and precision will find as many TP as possible and only a few FP. The F1-score is a weighted harmonic mean and it is a pessimistic mean calculation, where the smallest of the two will have more effect on the score. This is intended to lower the score if one drops even if the other increase by the same amount. [10]

$$F1\text{-}Score = \frac{2 * TP}{2 * TP + FP + FN} \tag{3.3}$$

To be able to better evaluate the performance of the detector, its use-cases need to be known. It will affect how the evaluation matrices are interpreted. To illustrate it, a system with high precision can still be ineffectual. Precision only measures the number of correct predictions out of the total predictions, not necessarily the success of a system. For instance, a diagnosis machine diagnoses cancer or not cancer. If it correctly detects 10 cancer patients among 100 patients, where 40 patients had cancer. then precision will be at 100%. Even though the machine itself is very faulty and would have a recall of only $\frac{10}{40} = 25\%$. Hence, a single measuring metric is not always effective to describe the performance of a detector. Therefore, other parameters also need to be taken into account.

3.2 Detection Evaluation Methodology

The detector's accuracy depends on the correct localization and recognition of the object. To evaluate the detector's performance a large set of 1216 images where peoples location was annotated, is anonymized by both the reference algorithm and a bit-exact model of the implementation written in the programming language C . The detector is then run on both sets of images and the results are evaluated. To compare the quality of the algorithm from a detection standpoint the precision, recall and F1 score matrices are utilized. The detector's function is in this thesis considered as a black box that deliverers the number of TN, FP and FN. The following Figure shows a simple block diagram of the detector testing methodology:

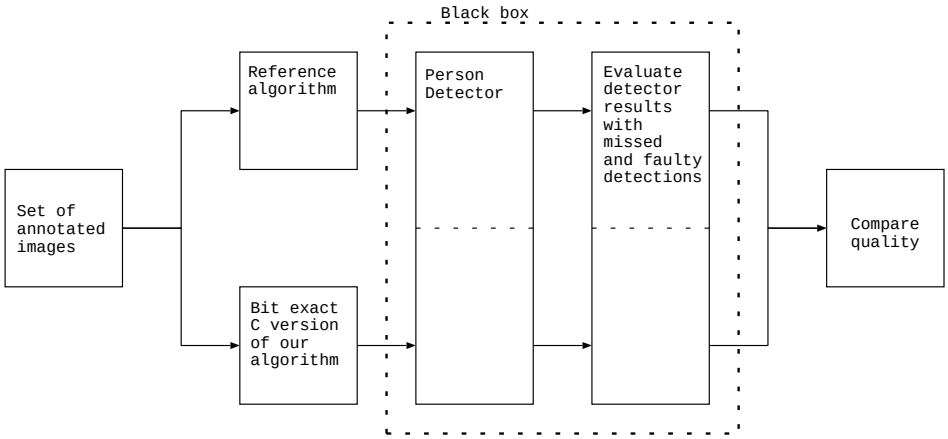

Figure 3.3: Structure of the detector quality tests.

Chapter 4

Required Adaptations of the Algorithm

In a regular digital camera a lot of processing is done on an image before it is ready to be viewed by a human. In this thesis, the term Image Processing Pipeline (IPP) will be used to describe these steps. For the camera used in this thesis the most drastic step is when the image from the sensor is taken from Bayer form (see section 4.1) to an RGB image. After passing through the IPP the finished image is available to the camera's CPU and can be sent out on the network for further use. In this thesis both the IPP and the CPU is integrated in the same *Artpec* chip, developed by *Axis Communications*.

The development platform used to realize this thesis consist of an image sensor, a *Xilinx* FPGA and an *Artpec* chip. A block diagram of this can be seen in Figure 4.1. The FPGA is placed directly after the image sensor and it is here that the algorithm will be implemented. The image sensor delivers pixel values 12 bits wide and two pixels per clock cycle to the FPGA. To adapt to the datarate Unfolding and Digital Signal Processing (see section 4.3) are used. The software that is synthesizing and placing the design for the FPGA is *Xilinx's Vivado 2016.3*.

The reference algorithm took an image in the RGB model after the IPP. This transform and the anonymization could be modeled as two transforms. One that performs the IPP processing ($\mathbf{T}_{IPP}$) and another to actually perform the

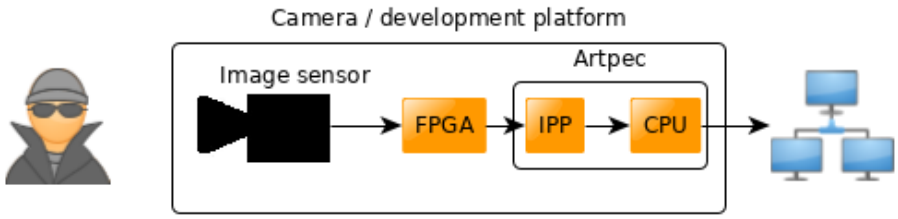

Figure 4.1: Block diagram of the development platform used to realize the anonymization.

anonymization ($\mathbf{A}_{anon}$). An Equation for the transformation can be seen below:

$$\mathbf{Image}_{anonymized} = \mathbf{Image}_{sensor}\, \mathbf{A}_{out} \tag{4.1}$$

$$\mathbf{A}_{out} = \mathbf{T}_{IPP}\, \mathbf{A}_{anon} \tag{4.2}$$

When the algorithm is moved directly to the sensor data $\mathbf{T}_{IPP}$ and $\mathbf{A}_{anon}$ effectively switch places. Since the transforms are not commutative $\mathbf{A}_{anon}$ needs to be changed to $\mathbf{A}^*_{anon}$. To be able to replicate the reference algorithm, $\mathbf{A}^*_{anon}$ has to perform an estimation of the effects of the IPP and its inverse $\mathbf{T}^{-1}_{IPP}$:

$$\mathbf{A}_{out} = \mathbf{T}_{IPP}\, \mathbf{A}_{anon} = (\mathbf{T}_{IPP}\, \mathbf{A}_{anon}\, \mathbf{T}^{-1}_{IPP})\, \mathbf{T}_{IPP} = \mathbf{A}^*_{anon}\, \mathbf{T}_{IPP} \tag{4.3}$$

$$\mathbf{A}^*_{anon} = \mathbf{T}_{IPP}\, \mathbf{A}_{anon}\, \mathbf{T}^{-1}_{IPP} \tag{4.4}$$

The transform of the IPP is very complex and has many steps, all of which might not be active in all situations. It is therefore necessary to estimate it. In this thesis, it will be approximated to only be the most drastic step of taking the data from Bayer form to RGB form.

4.1 Bayer Image Model

The Bayer image model is an image representation that is common in many digital cameras [7]. It represents an image with one value per pixel, but with each pixel only representing one color in a pattern. The pattern used by the sensor in the development platform can be seen in Equation 4.5, where each pixels color is given as R for red, G for green or B for blue. The sensor has a color filter before each pixel to only sample the luminescence value of that specific color [7]. It has a higher density of green pixels since the human eye is more sensitive to green light than red or blue [7].

$$\begin{bmatrix} B_{00} & G_{01} & B_{02} & G_{03} & . & . & . \\ G_{10} & R_{11} & G_{12} & R_{13} & & & \\ B_{20} & G_{21} & B_{22} & G_{23} & & & \\ G_{30} & R_{31} & G_{32} & R_{33} & & & \\ . & & & & . & & \\ . & & & & & . & \\ . & & & & & & . \end{bmatrix} \tag{4.5}$$

4.1.1 Demosaic

The act of transforming a Bayer model image to an RGB image is generally called demosaicking [11]. There are many methods to perform this and some of them are compared by Ramanath [11]. One of the simpler methods, that is also used in the IPP, is to perform bilinear interpolation. It estimates the missing color information by taking an arithmetic mean of the pixels in its direct vicinity (eight pixels surrounding it), with the same color as the one sought after [11].

For instance, to calculate the RGB values in $(1,1)$ in Matrix 4.5, the red value already exist, but the pixels Green and blue need to be determined. The green is calculated by averaging the four green values around the pixel:

$$G_{11} = \frac{G_{01} + G_{10} + G_{21} + G_{12}}{4} \tag{4.6}$$

Blue is in the same manner:

$$B_{11} = \frac{B_{00} + B_{02} + B_{20} + B_{22}}{4} \tag{4.7}$$

For the next pixel $(1,2)$, green is present, red and blue is calculated with:

$$R_{12} = \frac{R_{11} + R_{13}}{2} \tag{4.8}$$

$$B_{12} = \frac{B_{02} + B_{22}}{2} \tag{4.9}$$

This kind of pattern then continues for the rest of the image. This method can be seen as using four different spatial filters. The kernels would contain ones at the indexes where they need to take pixel data from, zeros from the rest and then divide the result by the number of ones, normalizing the kernel. To build an RGB pixel it needs to take pixels from the filtered images in regards to the Bayer pattern. Equation 4.6 describes a *Plus*, 4.7 a *Cross*, 4.8 a *Horizontal* line, 4.9 a *Vertical* line and the present color pixel is the *Mid* pixel. The following Table shows what filter pixels will be used for the top left corner. The same pattern repeats over the entire image:

Table 4.1: What spatial filters are used to build an RGB pixel from Bayer data. A visualization of the filters can be seen in Figure 5.6.

Index	Red	Green	Blue
(0,0)	Cross	Plus	Mid
(0,1)	Vertical	Mid	Horizontal
(1,0)	Horizontal	Mid	Vertical
(1,1)	Mid	Plus	Cross

The edges of the image need to be handled by extrapolating or zero padding. This thesis will use zero padding.

4.1.2 Remosaic

The reverse act of demosaicking is remosaicking. It filters an RGB image to Bayer image color form. This topic is seldom discussed in literature, so none have been referenced in the making of this stage. The simplest method to remosaic is just to pick out the colors in the Bayer pattern from the RGB image:

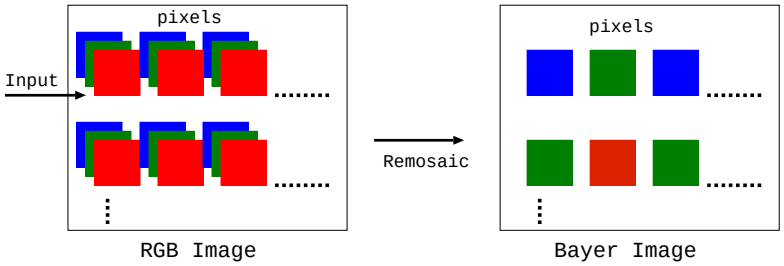

Figure 4.2: The basic color multiplexing concept to form Bayer image from an RGB image

The marked colors on the left-hand side are the colors that will generate the Bayer image on the right-hand side. The remosiac transform $\mathbf{T}_{remosaic}$ intends to be the inverse of the demosaic. This method will therefore fulfill the following Equation where $\mathbf{I}_{eye}$ is the identity matrix and $\mathbf{T}_{demosiac}$ is a bilinear interpolation.:

$$\mathbf{T}_{demosaic}\,\mathbf{T}_{remosaic} = \mathbf{I}_{eye} \tag{4.10}$$

Since demosaic is performing upsampling and remosaic downsampling, there exist at least one transform $\mathbf{A}$, where $\mathbf{A}$ is not the identity transform, that fulfill Equation:

$$\mathbf{T}_{demosaic}\,\mathbf{A}\,\mathbf{T}_{remosaic} = \mathbf{I}_{eye} \tag{4.11}$$

For instance, the transform that sets the green RGB value to zero in pixel $(1, 1)$, will be ignored since that index is reserved for a red pixel. And if a white line (all colors is set to max value) is drawn through green only pixels. It will be interpreted as a green line after the remosiacking stage. It is a drawback that it introduces this kind of color noise. However, due to its simplicity and that it fulfills Equation 4.10 it will be used anyway.

4.2 Number Representation

The reference algorithm was designed with rational numbers in mind. It used python's floating-point representation of the *IEEE-754* standard. In digital hardware, however, floating point representation is complex and and its standard [12] is extensive. According to the book *Digital Signal Processing* [13] floating point operations is hard to implement in hardware [13]. Third-party IPs from FPGA vendors like *Xilinx* can be used to make the implementation easier. Yet they are still large and slow. Simulating these IPs can also be a problem when a simulator that is not from the specific vendor is used. This is the case for this thesis. *Xilinx* floating-point IPs for the FPGA in the development platform, is not possible to simulate in the environment available for this thesis.

To replace the floating point operations, fixed-point arithmetic is used. It represents numbers with a fixed number of bits and what value each bit represents is always the same [14]. All values to the left of the decimal point represent

$2^{distance\ to\ dot}$ and right $2^{-distance\ to\ dot}$. The following example shows how the value 10.3125 can be represented with 8-bits and no sign bit:

$$1010.0101_2 = 10.3125_{10} \tag{4.12}$$

The fixed point values also have a constant distance between each other [14]. The distance is the value of the least significant bit or 2^{-N} where N is the number of fractional bits, whereas the distance between floating-point numbers is not constant [12]. It is large for large values and small for small [12].

4.3 Unfolding

Unfolding is a structured way to achieve parallel processing [15]. It allows a process to calculate multiple output values each clock cycle [15]. It is performed on a digital signal process which is briefly explained below:

4.3.1 Digital Signal Processing

In digital signal processing, algorithms are non-terminating and described as functions of previous inputs, outputs and constants [15]. A spatial filter is not non-terminating and will end when a full image is processed. Since the images contain a lot of pixels, the filter's datapath can be modeled in a non-terminating manner and then made terminating with some extra control logic. The gradient in Equation 2.5 could be written as $y(n) = x(n+1) - x(n-1)$ where n is the time instance, $y(n)$ is the output signal and $x(n)$ is the input. Since time instances ahead of time $(n + [\geq 1])$ is not possible, a delay to $y(n)$ needs to be added. A more implementable Equation would be:

$$y(n) = x(n) - x(n-2) \tag{4.13}$$

The datapath representation of this Equation can be seen in the following Figure 4.3:

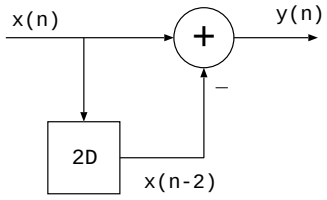

Figure 4.3: Datapath for the gradient function in a digital signal process

To make this datapath handle the edges in an image, add a MUX before the input of the adders. Then use it to set the signals to zero if they are outside of the image.

4.3.2 Unfolding

When performing unfolding more output functions are added [15]. The number of outputs is decided by the unfolding factor J [15]. They would be marked from $y(n)$ to $y(n+J-1)$. The same applies to the inputs with $x(n)$ to $x(n+J-1)$. In this case signals with $n+[\geq 1]$ is allowed as long as they exist as in or out signals [15].

To get a minimum number of delays and a correct signal assignment the functions need to be modified [15]. The n should be replaced with Jk and the signals should be re-written so that all have a format like:

$$y(J(k-d)+p) \tag{4.14}$$

Where d assigns the number of delays and p decides which input or output node to take the delayed signal from. To continue the previous example with the gradient function and an unfolding factor of $J = 3$. It would lead to the following three Equations:

$$y(n) = y(3k) = x(3k) - x(3k-2) = x(3k) - x(3(k-1)+1) \tag{4.15}$$

$$y(n+1) = y(3k+1) = x(3k+1) - x(3k-1) = x(3k+1) - x(3(k-1)+2) \tag{4.16}$$

$$y(n+2) = y(3k+2) = x(3k+2) - x(3k) = x(3k+2) - x(3k) \tag{4.17}$$

They would translate to the following datapath in Figure 4.4:

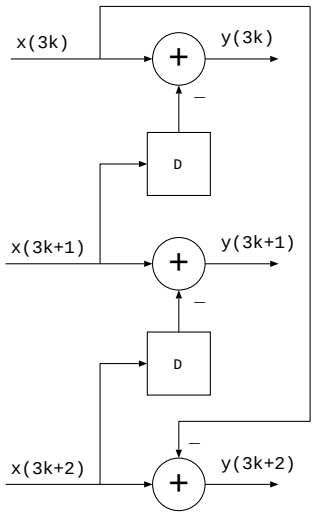

Figure 4.4: Unfolded gradient function in a digital signal process with an unfolding factor of $J = 3$.

Unfolding increases the throughput of a process as many times as the unfolding factor, without changing the number of registers present in the circuit [15]. This is the main reason it is interesting to this thesis. It is needed to accommodate the sensor's two pixel values per clock cycle.

Chapter 5

Implementation

The sensor that is used with the development platform, delivers an image with $1920x1088$ pixels and 35.6 frames per second. It transfers data with the MIPI interface over an LVDS link. How this works is outside of the scope for this thesis. The MIPI interface was translated to an *Axis* internal standard in a deserialization block that provides an output of two pixels per clock cycle. The internal standard is a master-slave protocol with a one clock cycle transfer where the slave can put the master on hold if it can not receive data. Similar results can be achieved with the AMBA AXI standard but the *Axis* internal protocol has been chosen for its simplicity.

The sensor dispatches the image with rows consisting of active pixels and horizontal blanking. The active pixels is the image data and the horizontal blanking is the time interval between the active data. It can be visualized as a break in the data delivery. The horizontal blanks can be utilized to introduce small delays without reducing the frame rate.

To accommodate for the move of the algorithm to before the IPP and to estimate Equation 4.4, Figure 2.1 needs to be extended with a demoasic block before and a remosaic block after. The Sobel module also needs to be elaborated with a block that takes the RGB image and transforms it into a grayscale image. A FIFO based on a dual-port RAM will be added to decouple the timing of the two parallel paths from each other. This FIFO will allow the Sobel operator to sit and wait for data from the box-filter while the split stream from the demosaic stage can keep feeding data to both paths. The FIFO needs to be big enough to not cause a lockup. If it gets full while the Sobel is waiting for the box filter to produce a pixel, it will block the demosaic module from producing more and the Box filter can not continue. The FIFO therefore needs to have a depth of at least:

$$\left\lceil \frac{size_{Box\ filter} - size_{Sobel\ operator}}{2} Row_{width} \right\rceil = \lceil 13.5 \cdot Row_{width} \rceil \tag{5.1}$$

Where $size_{Box\ filter}$ and $size_{Sobel\ operator}$ are the kernel sizes 30 and 3 respectively and Row_{width} is the width of the image. Both paths need to hold the same amount of rows before they can start delivering values. However, it can be divided by two

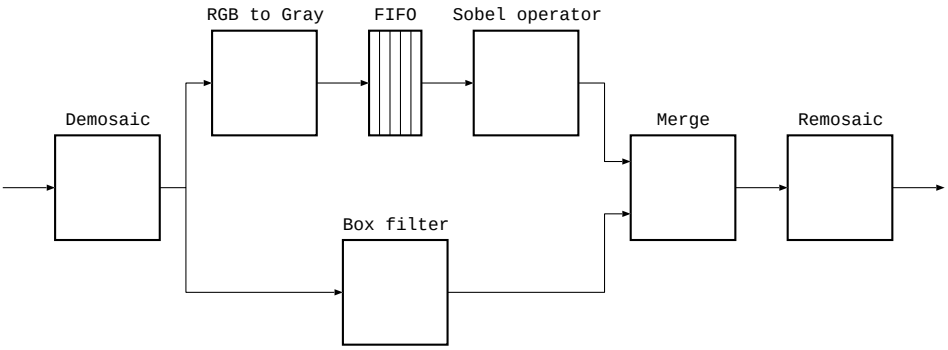

Figure 5.1: Block diagram of the first implementation of the algorithm.

since the spatial filters only need half the amount of rows before the first values are delivered. This is because, at the topmost row half of the kernel will cover pixels outside of the image. It is, therefore, this size that is needed to hold the excess data that does not fit in the Sobel operator before the box filter is ready to send. When both blocks are ready to transmit, the pixel pair will be consumed at the same time by the merger. A simplified block diagram can be seen in Figure 5.1.

5.1 Sub-matrix Generation

The Sobel operator, demosaic block and the box filter, all build on spatial filters and need to operate on a small sub-matrix out of a full image. The sensor delivers pixels row by row. Therefore, to build a sub-matrix a whole number of rows need to be stored. With a sub-matrix of the size $3x3$, the following Figure 5.2 shows that one whole and two partial rows is needed to fully represent a sub-matrix. The dashed line is the contents of the shift register, the bigger box represents the actual image and the smaller box is the sub-matrix.

When the next pixel is delivered the sub-matrix moves one step to the right. In general, if the index is q then it would be $x(n-q)$ with respect to $x(n)$. With an image width of 1920, the same sub-matrix would have the shift-register pixel indexes that can be seen in Figure 5.3.

The small box in the middle of the Figure 5.3 is the pixel that is operated on and the location where the result should be stored after a kernel multiplication. As a result, the input and output location in the sub-matrix are not aligned.

When performing unfolding on the image stream, the only thing that changes is the names of the indexes. To adapt with data rate, two sub-matrices need to be taken out at the same time. The number of indexes per row is halved but there are two pixels with the same index. This causes the Row_{width} in Equation 5.1 to be halved. To avoid the confusion while expressing, one is feed from $x(n)$ with a given

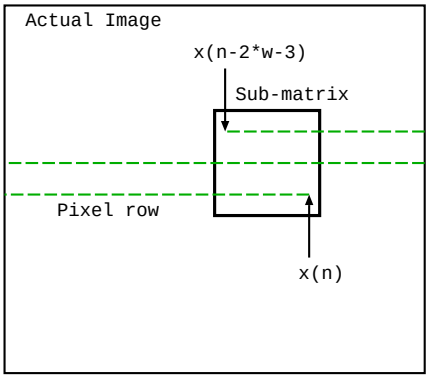

Figure 5.2: Sub-matrix of an image in a shift register. The latest pixel is $x(n)$ and the oldest one is $x(n - 2 * w - 3)$ where w is the image width.

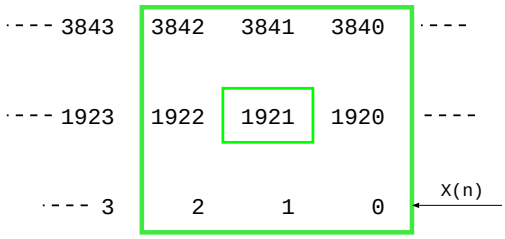

Figure 5.3: Representation of a 3x3 sub-matrix of an image with shift register indexes.

sub-index of 0, the other one is from $x(n + 1)$ with the sub-index 1. Figure 5.4 displays how the indexes are assigned for the two sub-matrices, where the index is d and the sub-index is p in $x(2(k - d) + p)$. The bigger left most box is the matrix for $y(2k)$ and the bigger right most box is for $y(2k + 1)$. Both the smaller boxes in the middle represent the pixels for which the two sub-matrices are utilized respectively.

At the edges of an image where some indexes are outside of the image, the values of those are not valid. They instead need to be replaced depending on what edge extrapolation method used. With zero-padding, they are replaced with zero. The contents of the shift register should not be changed but a MUX should replace the values to the output function.

The indexes of the shift register outside of the sub-matrix are mapped to LUT-RAM by *Vivado*. To reduce the utilization of these, the indexes can instead be implemented with dual-port RAM blocks, which will be mapped to B-RAM by *Vivado*. With all of the indexes in registers and LUT-RAM the design does not fit in the FPGA. This utilizes 345% of the available LUT-RAM blocks, which is not feasible. In contrast, by using the dual-port RAM for all spacial filters, no LUT-RAM is utilized but 57% of the B-RAM is used for the spatial filters. A

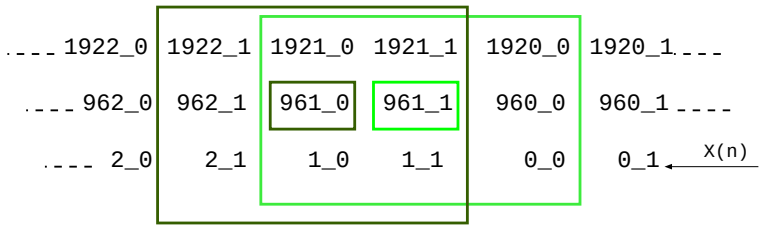

Figure 5.4: Representation of two 3x3 sub-matrices of an image with shift register indexes and two input signals.

significant amount of the RAM is occupied for the Box filter. If the box filter would use dual-port RAM, the design would utilizes 55% of B-RAM for the box filter 15% of LUT-RAM for the other two spatial filters.

5.2 Sobel Operator

The Sobel operator is implemented with the vector gradient $\nabla \mathbf{f}$ consisting of the two gradients in Equation 2.6 and 2.7. By using the shift register sub-matrix concept, the kernels needs to be scaled up to 3x3 around the center pixel in the sub-matrix. The spatial filtering can then be modeled with the following two functions. Where $w = 1920$ is the width of the image, g_x and g_y are the gradients in the direction of the x-axis and y-axis respectively.

$$g_y(n) = x(n) - x(n - 2 * w) \tag{5.2}$$

$$g_x(n) = x(n - w - 1) - x(n - w + 1) \tag{5.3}$$

To accommodate for two pixels per clock cycle, they can be unfolded to the following four Equations:

$$g_{y0}(2k) = x(2k) - x(2(k - w)) \tag{5.4}$$

$$g_{y1}(2k + 1) = x(2k + 1) - x(2(k - w) + 1) \tag{5.5}$$

$$g_{x0}(2k) = x\left(2\left(k - \frac{w}{2} - 1\right) + 1\right) - x\left(2\left(k - \frac{w}{2}\right) + 1\right) \tag{5.6}$$

$$g_{x1}(2k + 1) = x\left(2\left(k - \frac{w}{2}\right)\right) - x\left(2\left(k - \frac{w}{2} + 1\right)\right) \tag{5.7}$$

Where $\nabla f_0 = |||[g_{x0}(n), g_{y0}(n)]||| $ and $\nabla f_1 = |||[g_{x1}(n+1), g_{y1}(n+1)]|||$. The square root operation needed for the 2-norm is very hard to implement in hardware. It could be done numerically using Newton's method [6]. It iteratively tries to estimate x where the value to take the square root of x_0 equals x^2 [6], or in other

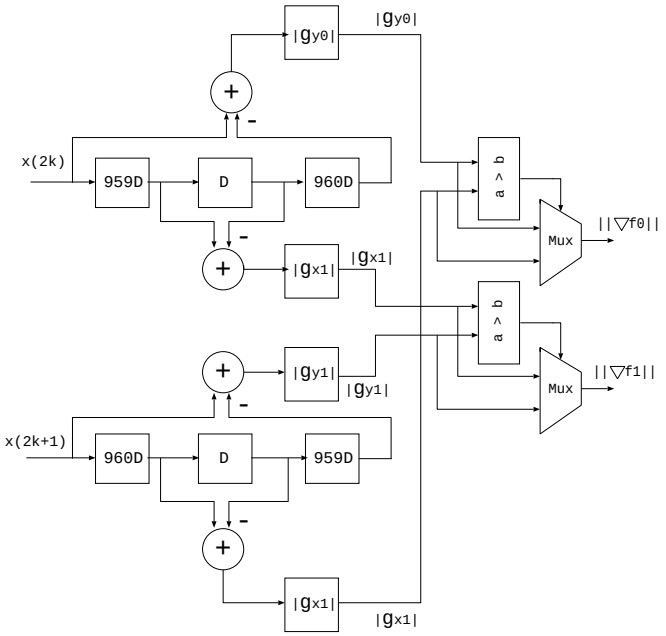

Figure 5.5: Datapath of the Sobel operator

words, the function $f(x) = x^2 - x_0 = 0$. It would use the following Equation 5.8 [6] to step towards to the correct answer, with a scaling factor β and n iterations:

$$x_n = x_{n-1} - \beta \frac{f(x_{n-1})}{f'(x_{n-1})} = x_{n-1} - \beta \frac{x_{n-1}^2 - x_0}{2x_{n-1}} \tag{5.8}$$

This would be very large in hardware and slow as it would require one division for each iteration. Testing shows that at most 15 iterations are needed to find the integer square root for a number between 0 and $0xFFFF\ FFFF$ if a scaling factor of $\beta = \frac{1}{4}$ is used. To simplify the operation the 1-norm could be used, but it has a risk of overflowing since it is always larger or equal to the 2-norm. The ∞-norm, on the other hand, is always smaller in comparison and do not risk of any overflow. It also has a very low complexity and only needs to decide which number is the largest, that is much smaller operation than 15 divisions. It is the method that will be used as the magnitude function instead of the 2-norm.

Implementing these into a datapath would result in the following Figure 5.5. The zero padding and handling of the image's edges is not covered in the datapath. To implement it pixel values from rows or columns that are outside of the image are set to 0 before the gradient calculations.

5.3 RGB to Gray

To avoid using floating-point numbers, the RGB to gray conversion was implemented in fixed-point. When converting the floating-point constants used in this model to fixed point, 27 fractional bits were needed to represent the float values fully. Note that the float values did not represent these variables fully either. For instance $0.299 * 2^{12} = 40131100.672$ but the float value can not represent 0.299 exactly but it is instead 0.298999994993209838867187500000000, which can be represented fully with 27 fractional bits. The Equation used is the same as in 2.1 but after the multiplication, the result was shifted down 27 bits and bit 26 was added to round the result. The constants that were used in the conversion can be seen in Table 5.1.

Table 5.1: Fixed point conversions of the RGB to gray constants.

Color	Constant float	Constant fixed point
Red	0.299	40131100
Green	0.587	78785808
Blue	0.114	15300821

5.4 Demosaic

The Demosaic block uses the sub-matrix structure as in Figure 5.4. The computations take one cycle and are executed based on a shift signal from the control block. The result is then passed to the output data bus. To carry out the averaging of a proximity, four different spatial filters is utilized named $f_{cross}(n)$, $f_{plus}(n)$, $f_{vertical}(n)$ and $f_{horizontal}(n)$. The base pixel is in $f_{mid}(n)$ where no interpolation is needed and the color is known. A visualization of the patterns and their Equations are given below:

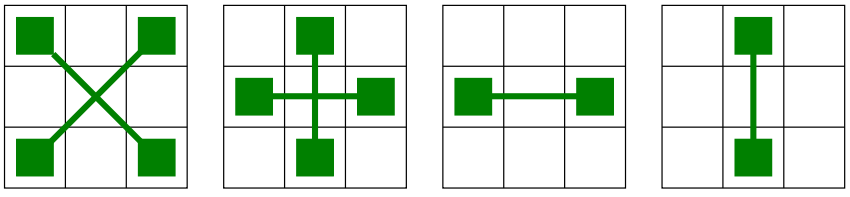

Figure 5.6: The different patterns for color calculation for performing demosaicking.

$$f_{cross}(n) = \frac{x(n) + x(n-2) + x(n-2w) + x(n-2w-2)}{4} \tag{5.9}$$

$$f_{plus}(n) = \frac{x(n-1) + x(n-w) + x(n-w-2) + x(n-2w-1)}{4} \tag{5.10}$$

$$f_{vertical}(n) = \frac{x(n-1) + x(n-2w-1)}{2} \tag{5.11}$$

$$f_{horizontal}(n) = \frac{x(n-w) + x(n-w-2)}{2} \tag{5.12}$$

$$f_{mid}(n) = x(n-w-1) \tag{5.13}$$

These filters are made with one pixel per clock cycle as input. In reality, two pixels are sampled at the same time and the Equations have been unfolded in the same way as the Sobel operator. A counter keeps track of the row and the column in the image that is operated on and that decides which spatial filters will be applied to build the two missing RGB pixels. Which filters to use follows the Table 4.1.

5.5 Remosaic

The merge generates an output image in RGB form. Hence, the remosaic block is incorporated to supply the IPP with images on the expected Bayer form. The method of doing this conversion is described in chapter 4. To implement the functionality, a MUX and a counter is used to generate the color pattern of the Bayer color model. The functionality depends on the following Figure 5.7:

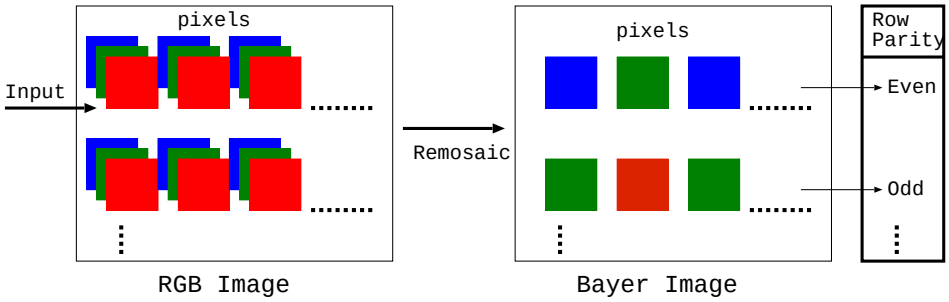

Figure 5.7: The basic color Mux of remosaic block based on row parity signal

A row parity signal is introduced to check whether the row is even or odd. With the two transfers per clock cycle formatting $[R_0G_0B_0, R_1G_1B_1]$ which selects $[B_0, G_1]$ for even rows and $[G_0, R_1]$ for odd rows.

5.6 Box Filter

By implementing the box filter as a spatial filter convolution and with the shift register structure, the following Equation can be utilized where w is the image width:

$$b(n) = \frac{1}{30^2} \sum_{r=0}^{30} \sum_{c=0}^{30} x(n - r \cdot w - c) \tag{5.14}$$

Here the resulting value is for the pixel in $(15, 15)$ seen from a 30x30 sub-matrix perspective. When unfolding the Equation 5.14, two sub-matrices will be generated and named $b_0(2k)$ and $b_1(2k+1)$. These two will overlap mostly and will have 29 columns in common. The overlapping columns can be summed up once which called $b_{both}(2k)$ in Equation 5.15 and utilized for the both sub-matrix summations in Equation 5.16 and 5.17.

$$b_{both}(2k) = \frac{1}{30^2} \sum_{r=0}^{30} \sum_{c=2}^{31} x\left(2\left(k - r\frac{w}{2}\right) - c\right) \tag{5.15}$$

$$b_0(2k) = b_{both}(2k) + \frac{1}{30^2} \sum_{r=0}^{30} x(n - r \cdot w - 32) \tag{5.16}$$

$$b_1(2k) = b_{both}(2k) + \frac{1}{30^2} \sum_{r=0}^{30} x(n - r \cdot w - 1) \tag{5.17}$$

In the non-unfolded Equation 5.14, the result was stored 15 columns behind the current bottom right pixel from $x(n)$. When unfolding this causes a problem since the result would then be calculated for $(15, 7_1)$ and $(15, 8_0)$ as seen from the perspective of Figure 5.4 scaled up to a 30x30 matrix. These two are in different output clock cycles and would make the image misaligned. To solve this the two sub-matrices need to be shifted one step to the left. This fix is included in the previous Equations 5.16 and 5.17 and it is the reason that $x(n)$ is no longer included in any of the two sub-matrices.

When the sub-matrices is shifting to the right, only the two rightmost columns are not present in the previous sub-matrices. This can also be exploited to reduce the complexity of the adder tree needed for the convolution. If only the two new columns need to be summed together during each sub-matrix calculation and then stored in a shift register, then that shift registers values can be used to calculate $b_0(2k)$ and $b_1(2k + 1)$. That way only a 30x2 matrix need to be outside of the RAM based shift registers and not a full 30x30. The first summation of a 30x1 column vector would add $\lceil log_2(30) \rceil = 5$ bits to the output and the summation of the previous columns would add 5 bits more. The number of bits the adder tree would need to handle would therefore be reduced from 32400 to 3741, where 2160 is needed for the two new columns and 1581 to sum together b_1 and b_2 fully. The number of registers needed to form the sub-matrix, excluding the indexes outside of the matrix, can be calculated using Equation 5.18 where K_h is the kernel height, K_w is the kernel width, U_{extras} is the extra columns needed because of unfolding, C_p is the color planes RGB and b_w is the bit width.

$$K_h * (K_w + U_{extras}) + C_p + b_w \tag{5.18}$$

A full 30x30 solution would require $30 * (30 + 2) * 3 * 12 = 34560$ register bits, whereas saving the summations would require $30 * 2 * 3 * 12 = 2160$ registers to

hold the 30x1 matrix and $32 * 1 * 3 * (12 + 5) = 1632$ to hold the column sums, giving a total of 3792 bit needed to calculate. The size of the RAM banks will not change since no matter if the depth changes 30 lines up or down it will still have to fit in two 36-bit wide and 1024 lines deep B-RAM blocks [16].

The normalization of the kernel, which required a division with $30 * 30 = 900$ was replaced with a multiplication of it's reciprocal. The reciprocal used 32-fractional bits and got the same result as the integer division in 99.8889% of the $2^{12+10} - 1$ possible numerators. No exact coverage can be attained since dividing 900 into primes result in $2^2 * 3^2 * 5^2$ and $\frac{1}{3}$ have an infinite number of decimals that can not be fully approximated with a finite number of bits. This reduced the LUT consumption from 179 to 4, but it now uses 10 DSP slices instead.

A datapath that for the box filter can be seen in Figure 5.8. The top lines of registers and adders are to get the two new column sums with the 30x2 kernel. The lower line of registers and adders is holding the previously passed columns and uses them to calculate Equation 5.15, 5.16 and 5.17. The multiplication and rounding in the bottom is there to perform the normalization of the result. The zero padding and handling of the image's edges is not shown. To implement it pixel values from rows that are outside of the image is set to 0 before the column summation adders and columns that are outside of the image is set to 0 before the immediate next addition.

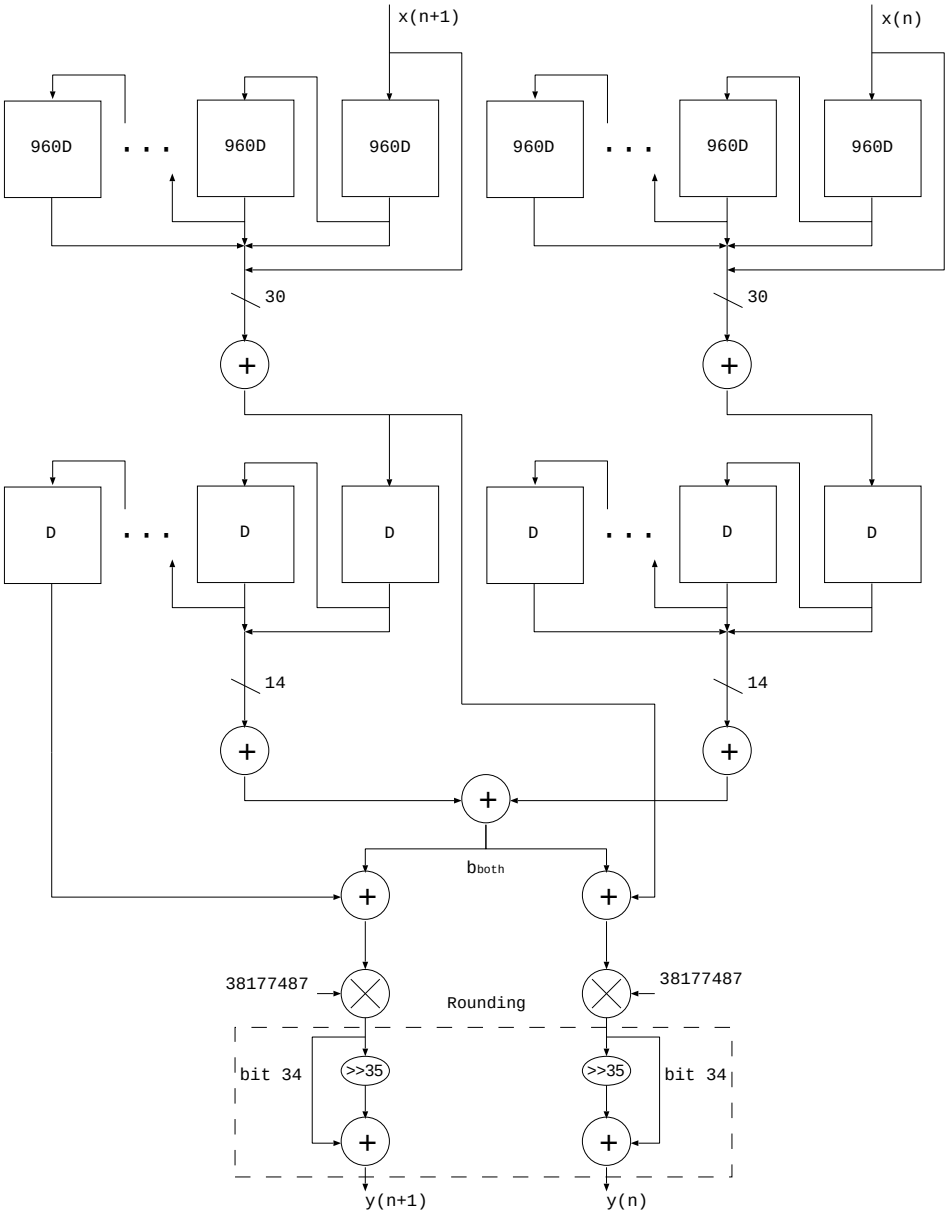

Figure 5.8: Simplified shift sum structure in the box filter. The shift reg contains an 30 rows worth of pixel data and out of it is collected a 30x2 sub-matrix.

5.7 Merge

To obtain the anonymized image, two filtered images out from the Sobel operator and box filter are overlayed inside the merger block. The overlay method that is used in this thesis is mentioned in section 2.3. The sensor generate 12 bit pixels so the *max bit value* to calculate α is in this case 4095 or $0xFFF$. The division was implemented with a multiplication of the reciprocal. For better precision 32 bits of fixed-point fractional bits are used for the alpha value and the reciprocal of 4095 is therefore 1048832. The number of fractional bits was chosen because it was the same amount of bits as in the floating-point values used in the reference algorithm. The resulting merged pixel value discards the fractional values by rounding. Because of the different kernel sizes and amount of blocks in the Sobel and box filter paths, the arrival time of inputs in the module is usually not in the same clock cycle. To handle this the path that finishes first is blocked until the other path is ready and its value will not be consumed until both are available.

Finally, a register holds the computed values until they can be consumed by the bus. A datapath without the registers can be seen in Figure 5.9, where the RGB pixels come from the box filter and the gray value from the Sobel operator.

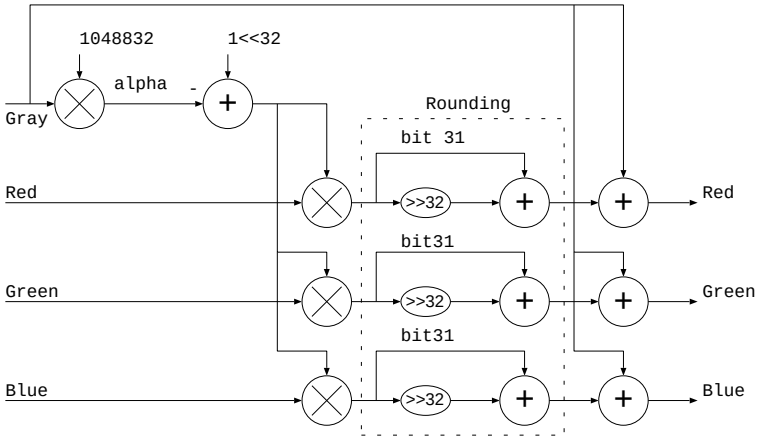

Figure 5.9: Data path for the merge block.

5.8 Hardware Utilization

Hardware utilization of the implementation on an *XC7A75T Xilinx Artix 7* FPGA can be seen in the following Table 5.2.

Table 5.2: Hardware utilization of the first implementation

Block	Sub Block	LUT	FF	LUT-RAM	B-RAM	DSP
	Available	47 200	47 200	19 000	105	180
	Demosaic	581	141	1440	0	0
	FIFO	89	26	0	26	0
	RGB to Gray	74	0	0	0	8
	Sobel operator	399	119	1440	0	0
First	Box filter	6 292	3 876	0	58	10
	Merge	132	61	0	0	0
	Remosaic	1	0	0	0	0
	Top	7 568	4 300	2880	84	18

From Table 5.2 it can be observed that by far the biggest block is box filter. It uses 83% of the utilized LUTs, 90% of the registers and 69% of the RAM tiles. Only 5.6% of the LUTs is used by the controller and the datapath is therefore utilizing the most of the allocated hardware. Hence, reducing the kernel size in the box filter can go a long way in minimizing the design. However, since the kernel size was chosen to get a good anonymization at 6 meters, it is a parameter that can not be changed. A shorter row width on the other hand can reduce the memory utilization without reducing the level of anonymization. It can be reduced by for instance using a smaller resolution or by flipping the sensor 90° to get a row width of 1088 instead of 1920. The FIFO is also using plenty of RAM tiles with its 31% of the utilized and since its size is depending on the size of the box filter it would also benefit from a shorter row width. Downsampling could also be an option to reduce the width of the image. However, if it is implemented only around the box filter it would not affect the size of the FIFO.

The reason the FIFO is not utilizing a third of the B-RAM that the Box filter is using is because the number of memory locations required has to be a power of two [16] to fit in the memory modules instead of the precise value given in Equation 5.1. The mentioned Equation explains that the FIFO needed at least 25920 memory locations but the smallest power of two that would fit this amount is $2^{32} = 32768 = 32K$. The FIFO needs to be $12 * 2 + 2 = 26$ bits wide consisting of two gray pixels and two bits from the interconnection protocol. According to the FPGA's documentation [16] a 32K deep memory need as many modules as it has bits per line, this explains the 26 modules used by the FIFO. The box filter on the other hand needed 960 locations deep and 72-bits wide per shift register, which fits in two 1K deep and 36-bits wide B-RAM modules [16], with the 29 shift registers this makes the result of 58 B-RAM modules expected.

89% of the LUTs in the demosaic block and 70% in the Sobel operator was part of the controller and not in the data path. Therefore, it is not beneficial to optimize it. Changing the interconnection protocol to something simpler would not reduce the controller since the used one was chosen because of its simplicity and the controller revolves mainly on how to handle the edges in the image and would not

be affected by a protocol switch.

5.9 Human Vision Comparison

Visual comparison by juxtaposing results is one of the most common ways to compare. In this case, two outcomes from the reference algorithm and hardware implementation are placed side by side and a human vision's comparison was made. The hardware results are in the Bayer image form and the reference algorithm results in an RGB image color model. The Bayer color image is not pleasurable to human vision. To make it more visible to human eyes the anonymized image from the reference algorithm and FPGA implementation are shown in an RGB image. Figure 5.10 displays the original and the two anonymized images, from the reference model and the hardware implementation respectively. In general, both images look similar except a slight change of color intensity. It seems the reference algorithm retains more intensity.

Figure 5.10: RGB images where the left image is without the transformation, the middle one with the reference algorithm applied to it and the right image have gone through the FPGA implementation and the IPP.

5.10 Detector Results

A detector test as described in Section 3.2 was executed and the algorithm of the first implementation was compared against the reference algorithm. A second test was also performed where the first implementation had instead been implemented with the 2-norm, to better illustrate the effects that this change had on the detector performance. These results in terms of precision, recall and F1-score can be seen in Table 5.3. Only considering the value of precision or recall to measure the detector's performance can lead to a misleading conclusion. Therefore, F1-score is an important measure metric that includes the combined impact of precision and recall by calculating the harmonic mean of those.

Table 5.3: Detector performance of the reference algorithm and first implementation.

Implementation	Precision	Recall	F1
Reference	88.0882%	87.0640%	87.5731%
First 2-norm	87.7941%	86.7733%	87.2807%
First ∞-norm	80.6011%	85.7558%	83.0986%

Here, the Table shows that the reference algorithm's precision is higher than its

recall value. This indicates that the detector detects less false-positive cases compare to the false-negative cases which implies it can better detect in predicted cases even though it could not cover all the relevant cases in the reference algorithm. The hardware implementation with 2-norm follows the same result manner where the precision is higher then the recall. The F1-score of the hardware implementation with 2-norm drops with less than a percentage point with respect to the reference algorithm. This can be considered as similar performance and most likely will perform exactly like the reference algorithm after a retraining. However, the hardware implementation with ∞-norm curtails the detector's performance. By observing the precision and recall value of this, it shows that the detector had hard time to predict successfully among all predicted cases with compare to actual cases. Therefore, the F1-score dropped with 4 percentage point from the reference algorithm which from a product perspective is not desirable. The ∞-norm calculation for highlighting edges makes the borders less prominent compared to the 2-norm calculation. The detector that was used, had been trained with a non-anonymized image-set and then retrained with anonymized images from the reference algorithm which utilizes the 2-norm, this is most likely the reason for the drop in performance when using the ∞-norm, since this produces images that does not contain the same features that the detector expects. However, retraining with the new set of images from the hardware implementation will improve the detector's performance [17]. Hence, from a hardware perspective and considering retraining, the implementation of the 2-norm is not worth it as it adds complexity compared to the ∞-norm and would increase the utilization cost significantly.

Chapter 6

Algorithmic Adjustments

At this point, the implementation is functional and have reached the first goal of the thesis that is to perform anonymization on the sensor feed. However, the second goal was to try to improve the algorithm and try to reduce the hardware utilization. The two blocks where simplifications will be tested are the merge block and the box filter. The merge is one of the smaller blocks but since it is using many bits in its calculation and requires 4 multipliers to implement two strength reduction techniques will be tested. The box filter, by far is the largest block in the design and as discussed in Section 5.8, the reduction of its hardware utilization is paramount to minimizing the overall design. This chapter will first suggest three approximations and test these against the detector to see how they affect its performance. The detector will be tested by using both the 2-norm and ∞-norm in the Sobel operator. The reason behind is the ∞-norm will examine the current state of the implementation, the 2-norm will closely resemble the reference algorithm and isolate the effects of the changes. The approximations with the best detector results will be implemented and their utilization will be compared to the first implementation and any potential changes in utilization will be observed.

6.1 Merge

6.1.1 Fractional Bit Reduction

Curtailing the number of bits in a circuit is an effective way to reduce the number of logic cells needed to realize it. The number of bits used by the fixed point representation in the merge block was chosen to be able to replicate the precision of the intermittent floating-point values in the reference algorithm. Floats have very high precision and the detector might not need the full extent of it. The first implementation used 32 whole number bits with additional 12 fractional bits to the left of the decimal points, leading to a total of 44-bits. The results of a detector test with a fractional bits reduction to 10, 16 and 24-bits can be seen in Table 6.1.

Table 6.1: Detector performance of the FPGA algorithms and bit reductions in the merge. The number of bits given are the fractional bits.

Implementation	Precision	Recall	F1
Reference	88.0882%	87.0640%	87.5731%
First 2-norm	87.7941%	86.7733%	87.2807%
First ∞-norm	80.6011%	85.7558%	83.0986%
10 bits, 2-norm	80.7018%	86.9186%	83.6949%
16 bits, 2-norm	88.1832%	86.7733%	87.4725%
24 bits, 2-norm	87.7941%	86.7733%	87.2807%
10 bits, ∞-norm	72.6161%	86.3372%	78.8845%
16 bits, ∞-norm	80.7428%	85.3198%	82.9682%
24 bits, ∞-norm	80.9066%	85.6105%	83.1921%

By observing Table 6.1, it can be concluded that when using the 2-norm, reducing the number of fractional bits to 24 yields no effects at all to the performance. The precision, recall, F1-score are exactly the same as for 32-bits. With 16 fractional bits, the precision improves slightly by 0.389 percentage points. 10 fractional bits yields a better recall but worse precision which decreased the F1-score with 3.59 percentage points. It implies that the detector finds more false positives but also more of the true positives. The ∞-norm results in Table 6.1 implies that unlike to the 2-norm 24 fractional bits infers a slight recall reduction of 0.145 percentage points. 16 fractional bits did yet again increase the precision but with ∞-norm the recall reduced by 0.436 percentage points which lower the F1-score by 0.130 points. 10 fractional bits reduced the precision much more than any of the other with 7.99 points, the recall increased slightly with 0.581 points but the overall F1-score was still lowered with 0.421 percentage points. From these results it can be concluded that the decrease in the number of bits from 16 to 10 would not be worth it because of the relatively large precision loss that was observed for both the 2-norm and the ∞-norm test.

6.1.2 Division Approximation

The division with *max bit value* in the α calculation in Equation 2.15 in the first implementation was performed by multiplying the Sobel pixel with the reciprocal of the divisor. Using the reciprocal instead of a direct division operation is already a hardware simplification. However, $max\ bit\ value = 2^{bits} - 1$ which is approximately 2^{bits} is advantageous since a division by a power of two can be implemented with a simple right shift operation. Equation 6.1 can therefore be used to estimate the division. The results of a detector run using this approximation and different numbers of fractional bits can be seen in Table 6.2.

$$\alpha = \frac{Sobel\ pixel}{2^{bits} - 1} \approx Sobel\ pixel >> bits \tag{6.1}$$

Table 6.2: Detector's evaluation metric values of the First implementation and the division approximation in the merge block. The number of bits given are the fractional bits.

Implementation	Precision	Recall	F1
Reference	88.0882%	87.0640%	87.5731%
First 2-norm	87.7941%	86.7733%	87.2807%
First ∞-norm	80.6011%	85.7558%	83.0986%
No division, 32 bits, 2-norm	88.1129%	86.1919%	87.1418%
No division, 24 bits, 2-norm	87.9643%	86.0465%	86.9949%
No division, 16 bits, 2-norm	88.2789%	86.4826%	87.3715%
No division, 32 bits, ∞-norm	80.7428%	85.3198%	82.9682%
No division, 24 bits, ∞-norm	80.7428%	85.3198%	82.9682%
No division, 16 bits, ∞-norm	80.7428%	85.3198%	82.9682%

From Table 6.2, it can be observed that the approximation reduces the recall slightly for all norms and bit widths which implies more false negative cases were found. The precision, on the other hand, has increased for all norms and bit widths. The F1 score was decreased for all of the tests except for the 2-norm with 16 fractional bits test where it increased with 0.0908 percentage points. Another observation is that it does not matter for the detector how many bits are used among the three tested widths for the ∞-norm. This is expected since the twelve bit pixel from the Sobel operator are padded with zeros to add more fractional bits. It would therefore have the same performance for everything down to 12-bits, where all twelve incoming bits would just be the fractional bits.

6.2 Box Filter

6.2.1 Downsampling

As noted in section 5.8 the two things that made the box filter large are the kernel size and the image width. The kernel size is set to get a blurry image at a distance of 6 meters. It can, therefore, not directly be reduced or the blur would not be strong enough. The utilization of LUT and FF is also relatively low compared to memory-utilization. Hence, the memory usage is more important to minimize to be able to use a smaller FPGA. The image width does affect memory usage overall and its reduction would make the Sobel, demosaic and the FIFO also smaller. A simple way to diminish the width of an image is to downsample it. Downsampling is performed by discarding samples and only use a subset of pixels in the image. This will decrease the entropy of the image. An example of how to downsample the columns can be seen in the following Figure 6.1: To get back to the original resolution, upsampling needs to be performed. An example of how to reverse Figure 6.1 is to duplicate the columns in the image to the right. Since downsampling reduces entropy, upsampling can not be the exact inverse $A^{-1}_{downsample}$ and $A_{downsample}\, A_{upsample} \neq I_{eye}$.

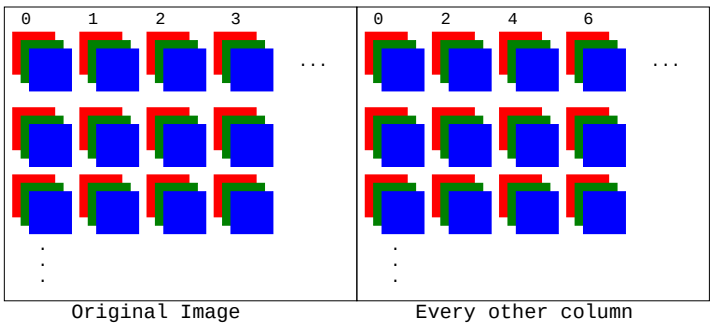

Figure 6.1: Comparison between a downsampled image and its original.

However, if the kernel size of the box filter is the same size as previously used it will filter over a larger spacial area than when using the full resolution. The 30x30 kernel would cover an area equal to 30x60 in the original image. Reducing the kernel size to 30x15 by downsampling would cover the same pixel area and also cut down the number of column sums stored by half.

The detector was re-trained with images from the reference algorithm and is expecting that the edges from the Sobel operator should have the original 1920x1088 resolution. Therefore, it is inferable that downsampling only around the box filter would diminish the performance less compared with general downsampling of the design. In Table 6.3 below the difference between the following two points are compared:

- General downsampling of the design. It is applied after the demosaic and upsampled again before the remosaic. Both the 2-norm and ∞-norm are used.
- Only downsampling around the box filter with the scaled-down 30x15 kernel and only utilize the ∞-norm.

Table 6.3: Detector performance of the First algorithm and the downsampling adjustment.Here, **DS** stands for **D**own**S**ampling.

Implementation	Precision	Recall	F1
Reference	88.0882%	87.0640%	87.5731%
First 2-norm	87.7941%	86.7733%	87.2807%
First ∞-norm	80.6011%	85.7558%	83.0986%
DS all, 2-norm	92.0530%	60.6105%	73.0938%
DS all, ∞-norm	89.6629%	57.9942%	70.4325%,
DS box filter only, ∞-norm	85.5590%	80.0872%	82.7327%

Table 6.3 shows that the general downsampling decreased performance with an F1-

score reduction of 14.2 percentage points for 2-norm and 12.7 percentage points for the ∞-norm. This is significant compare to the loss of 4.55 percentage points caused by performing it only around the box filter. The drop of recall values for the general downsampling tests indicate that the lower resolution Sobel edges made it harder for the detector to find people in the image. As per expectation, downsampling only around the box filter and using a kernel with a scaled size, did not reduce performance as much as the general case. The recall only decreased by 5.67 percentage points and the precision increased by 4.96 percentage points. It, therefore, had a smaller decrease in the F1-score compared to the general downsampling.

6.3 Implementation

The following two adaptations was chosen for implementation:

- Approximating the division in the merge block and using the Sobel pixels directly as 12 fractional bits.
- Performing downsampling around the box filter with a kernel scaled down to 30x15.

They were chosen because they had the lowest F1-score reduction out of the tested cases in the Tables 6.2 and 6.3. To adapt the merge block, the datapath in Figure 5.9 only need to remove the multiplication with the reciprocal, directly assign the incoming Sobel pixel to the alpha value and reduce the number of fractional bits in the rounding to 12. The downsampling around the box filter will be realized in hardware by discarding the second pixel ($x(2k+1)$) every clock cycle. Since unfolding is no longer required Equation 5.14 can be used directly, but with $c = 15$. The column sum shift register will still be used and Figure 5.8 still applies, but to be able to upsample back to the larger resolution, $b_1(n)$ is directly connected to $b_0(n)$. Table 6.4 hold the utilization of the new adjustments and the first implementation for an easier comparison.

From Tabel 6.4 it can be observed that the merge approximation did not save any LUT, it instead increased the amount needed by 24 and also required 5 more DSP slices. Even thought the datapath was simplified, it did not lead to a smaller design. This is mainly because *Vivado* decided to map $(1-\alpha) * RGB\ pixel$ to DSP slices instead of regular LUT. The DSP slices required the extra LUTs to be controlled by the circuit. The allocation and its routing also made it harder to minimize the number of LUT in the box filter, that is why the total amount of LUT increased by 251 and not just the 24 in the merge block.

Table 6.4 also shows that downsampling around the box filter did reduce the hardware utilization. The number of LUTs within the filter is reduced by 41.0%, FF by 42.0% and RAM blocks by 50%. For the total utilization the number of LUT is reduced by 33.4%, FF by 46.2% and B-RAM by 34.5% This is expected and the memory utilization is lower due to the halving of the memory line width

Table 6.4: Hardware utilization of the new adjustments. Here, **DS** stands for **D**own**S**ampling.

Block	Sub Block	LUT-Logic	FF	LUT-RAM	B-RAM	DSP
	Available	47 200	47 200	19 000	105	180
First	Demosaic	581	141	1440	0	0
	FIFO	89	26	0	26	0
	RGB to Gray	74	0	0	0	8
	Sobel	399	119	1440	0	0
	Box filter	6 292	3 876	0	58	10
	Merge	132	61	0	0	0
	Remosaic	1	0	0	0	0
	Top	7 568	4 300	2880	84	18
Merge	Demosaic	582	142	1440	0	0
	FIFO	89	26	0	26	0
	RGB to Gray	74	0	0	0	8
	Sobel	400	118	1440	0	0
	Box filter	6 518	3 867	0	58	10
	Merge	156	61	0	0	5
	Remosaic	1	0	0	0	0
	Top	7 819	4 289	2880	84	23
DS	Demosaic	558	143	1440	0	0
	FIFO	89	26	0	26	0
	RGB to Gray	74	0	0	0	8
	Sobel	399	116	1440	0	0
	Box filter	3 711	2 314	0	29	6
	Merge	132	61	0	0	0
	Remosaic	1	0	0	0	0
	Top	4 964	2 314	2880	55	14

Figure 6.2: Resulting anonymized images from the first hardware implementation (left), the approximated merge block (middle) and with box filter downsampling (right).

from 72 to 36, the LUT and FF reduction is caused by the kernel scaling to 30x15 instead of 30x30.

This significant reduction in utilization and relatively low performance-cost makes this change strongly recommended. With it the design would fit in a smaller FPGA. The current *Xilinx Atrix 7 XC7A75T* could be replaced by a *Atrix 7 XC7A50T*, but by replacing three B-RAM shift-registers in the box filter with LUT RAM the even smaller *Atrix 7 XC7A35T* could be used instead. Visually it can be observed in Figure 6.2 that the image quality has hardly been affected by the two different approximations and it is very hard to see any differences.

Chapter 7

Conclusion

7.1 Implementations and Improvements

The chosen algorithm is already well suited for FPGA implementation. Yet, for it to function on a sensor stream it needs to be adapted to take Bayer images as input and output. This can be done as described in chapter 4 by performing a bilinear interpolation on the incoming Bayer image and then remosaic the image before sending it out of the FPGA again. Remosaicking can be performed by taking out the colors in the Bayer pattern from the outgoing RGB image.

7.1.1 Spatial Filters and FIFO

As discussed in chapter 5, the demosaic block, Sobel operator and box filter can all be implemented like spatial filters. These use sub-matrices and can be implemented with a shift-register structure, where the amount of rows in the kernel dictate how many rows in the image need to fit in the shift-register. The parts of the rows that are not needed in the sub-matrices will be placed in distributed LUT-RAM but to make the box filter fit, it needs to place those indexes in in dual-port RAM banks. Unfolding does not change the number of bits that need to be stored to perform the filtering. It will instead double the width of each register but halve the shift-register depth, leading to a net-zero change in the number of bits. It was shown in table 5.2 that the box filter was the largest hardware block and can preferably to be minimized in regards of memory and logic utilization. To perform this, either the kernel size or the number of columns in the image requires to be diminished. In this thesis row width was cut down by performing downsampling around the box filter, where every other column was discarded. It also forced a kernel reduction from $30x30$ to $30x15$, since the kernel should cover the same area in the image after downsampling as before it. Table 6.3 shows that this change diminished the detector's performance slightly, but it is inferable that retraining will make the detector regain its performance. In contrast, the utilization reduction was very high, as seen in table 6.4. The utilization in the FPGA fell between 33.4% and 46.2% for LUT, FF and B-RAM. Therefore downsampling can bring great benefit to the design.

Table 5.3, its following discussion and the section 5.2 illustrates that the 2-norm calculation in the Sobel operator can preferably be replaced with an ∞-norm. The main reason is to avoid implementing the square root in the 2-norm. The square root would as discussed in section 5.2 have to be implemented numerically with Newton's method and would need at least 15 very large division operations. This change will reduce detector performance with multiple percentage points. However, it is probable that these losses would be regained after retraining the detector. Hence, with retraining and the hardware complexity reduction, the ∞-norm is highly recommended.

In accordance with chapter 5, a FIFO can preferably be added between the RGB to gray converter and the Sobel operator. It will be used to decouple the Sobel and box filter paths and allow them to perform calculations at their own rate. The FIFO needs to contain sufficient memory so that the Sobel path can store at least as many pixels as the box filter needs to consume to produce its first pixel. The depth therefore depends on the number of kernel rows in the Sobel operator and the box filter.

7.1.2 RGB to Gray and Merge

Floating-point arithmetic is very complex and hard to implement. Section 4.2 therefore, advise that all floating-point calculations should be replaced with fixed-point arithmetic. Section 5.3 states that the RGB to gray converter require 27 decimal point bits to fully replicate the floating-point operations. Table 5.2 shows that it does not utilize much hardware and can be allowed to keep its decimal bits without reducing them.

The merger, on the other hand, could be simplified by approximating the $maxbitvlaue = 4095 = 2^{12} - 1 \approx 2^{12}$ and use the Sobel pixels directly as a 12-fractional bits alpha channel. The approximation did cause a performance loss displayed in Table 6.2. Table 6.4 shows that the hardware utilization increased instead of reduced. The approximation is therefore discouraged and using the reciprocal of 4095 is a better optimization.

7.2 Future Work

The person detector is trained to detect people in images similar to the ones it has been trained on. If an alteration of the algorithm changes how the resulting image looks it will make it harder for the detector to find people. If the detector was retrained with a dataset constructed with the new algorithm, more alterations would be viable and possible to perform.

7.2.1 Sobel Operator

The Sobel operator is a block that can affect the look of the image a lot with just small changes. For instance replacing the gradients with the larger kernel in Equation 2.8 and 2.9 the edges are highlighted more strongly and smoothly.

Figure 7.1: Edge detection with a 3x3 kernel.

With the current detector, this modification got a precision of 0.944862, recall of 0.547965 and F1-score of 0.693652. Performing a retraining of the detector to see if the more strongly highlighted edges would make it easier to get a good detector would be interesting. It is, however, firmly out of scope for this thesis.

One problem that exists with the current implementation of the Sobel operator is that an edge will disappear between two color fields that have the same or very close grayscale values. For instance by using Equation 2.1 it can be proved that an edge between the blue field with intensity 200 and the red field with intensity $200\frac{0.114}{0,209} \approx 109$ will have the same gray value and no edge will be found. This can also be seen in the following Figure 7.2:

Figure 7.2: Visualization of how grayscale conversion can hide edges. The left fields are of color and the right field is both colors grayscale value.

That it is a problem for edge detection can be seen in the following Figure 7.3, where the similar gray values do not highlight the edges between the body and the pillow:

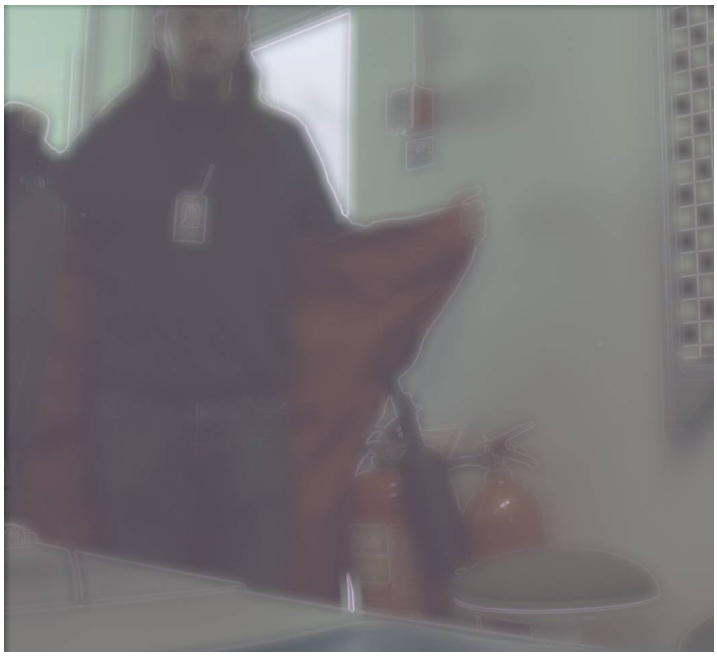

Figure 7.3: Anonymized image where same gray value caused an edge to disappear between the body and the pillow.

A possible solution would be to run a Sobel operator for each color channel and take the maximum value of all three. An example image can be seen in Figure 7.4:

The edges are even more pronounced than with the $3x3$ kernel discussed above. It is, therefore, likely that the detector with its current training would not function well with this modification. The detector would have to be retrained for it to be properly evaluated. With the very pronounced lines, there is also a reduction in anonymization. It adds more lines in the face and makes it easier to recognize a person compared to the relatively more blurry faces with grayscale Sobel.

7.2.2 Demosaic

The bilinear interpolation method has a smoothing effect on edges [11]. In the paper *Demosaicking methods for Bayer color arrays* [11] it is suggested that Freeman's method introduces the least amount of errors when there is a speckle behavior in the image. It performs a bilinear interpolation first and then applies a median filter to the difference between the color channels to remove fringes and make edges clear again [11]. A median filter ranks the brightness in a neighborhood around a pixel and takes the median as its new value [7]. This method will no longer fulfill Equation 4.10. If it would matter or not needs to be evaluated but it would be a larger departure from the replication of the IPP.

One interesting thing to test would be to not perform the up-sampling from the

Figure 7.4: Image that is using one Sobel operator per color channel and uses the maximum to calculate the alpha channel.

interpolation methods at all. Rather use four Bayer pixels from the patern in Equation 4.5 to build an RGB pixel and assume that the difference in location is negligible. The calculation would follow the following Equation:

$$\begin{bmatrix} B_{0,0} & G_{0,1} \\ G_{1,0} & R_{1,1} \end{bmatrix} \rightarrow R = R_{1,1},\ G = \frac{G_{0,1} + G_{1,0}}{2},\ B = B_{0,0} \tag{7.1}$$

To remosaic, simply add back the pixels in its pattern and use the green value twice. It will not perform the color distortion that is caused by the current method of simply taking out the Bayer pattern from the RGB image. The edges that are detected can therefore stay white and make it easier for the detector to find them. There would no longer be a risk that they would disappear because of the phenomenon discussed in section 4.1.2.

It will add more distortion than the interpolation methods but it will also not perform any up-sampling. For minimal hardware implementation, this could be more efficient. Especially if the detector can be retrained to expect the edges from this smaller resolution.

Bibliography

[1] R. C. Gonzales and R. E. Woods, *Digital image processing, second edition*. Prentice Hall, 2002.

[2] *Color conversions*, accessed on the webpage `https://docs.opencv.org`, version 4.20, OpenCV team, Mar. 2020.

[3] J. Månsson and P. Nordbeck, *Endimensionell analys*. Lund Sweden: Studnetlitteratur AB, 2011.

[4] P. R. Mercer, *More Calculus of a Single Variable*. New York, NY, United States: Springer, 2014.

[5] *Sobel derivatives*, accessed on the webpage `https://docs.opencv.org`, version 4.20, OpenCV team, Mar. 2020.

[6] T. Sauer, *Numerical Analysis*, second. Harlow, Great Britain: Pearson, 2014.

[7] J. C. Russ, *Image processing handbook: fourth edition*. CRC Press, 2002.

[8] L. Liu, W. Ouyang, X. Wang, P. Fieguth, J. Chen, X. Liu, and M. Pietikäinen, "Deep learning for generic object detection: A survey," *International Journal of Computer Vision*, vol. 128, pp. 261–318, Feb. 2020.

[9] T. Hamed, R. Dara, and S. C. Kremer, "Chapter 6 - intrusion detection in contemporary environments," in *Computer and Information Security Handbook (Third Edition)*, J. R. Vacca, Ed., Third Edition, Boston: Morgan Kaufmann, 2017, pp. 109–130, ISBN: 978-0-12-803843-7. DOI: `https://doi.org/10.1016/B978-0-12-803843-7.00006-5`. [Online]. Available: `http://www.sciencedirect.com/science/article/pii/B9780128038437000065`.

[10] L. Leonard, "Chapter one - web-based behavioral modeling for continuous user authentication (cua)," in *Advances in Computers*, A. M. Memon, Ed., vol. 105, Elsevier, 2017, pp. 1–44. DOI: `https://doi.`

org/10.1016/bs.adcom.2016.12.001. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0065245816300742.

[11] R. Ramanath, W. E. Snyder, G. L. Bilbro, and W. A. Sander, "Demosaicking methods for bayer color arrays," *Journal of Electronic Imaging*, vol. 11, no. 3, pp. 306–315, Jul. 2002.

[12] "Ieee standard for floating-point arithmetic," *IEEE Std 754-2008*, pp. 1–70, 2008.

[13] P. R. Babu, *Digital signal processing*, 7th ed. Tambaram West, Chennai, India: Scitech Publications (India) PVT.LTD, 2017.

[14] "Ieee standard computer dictionary: A compilation of ieee standard computer glossaries," *IEEE Std 610*, pp. 1–217, 1991.

[15] K. K. Parhi, *VLSI Digital Signal Processing Systems: Design and Implementation*. John Wiley & Sons, 1999.

[16] X. Inc., *7 series fpgas data sheet: Overview*, Last Access: 2020-06-24, https://www.xilinx.com/support/documentation/data_sheets/ds180_7Series_Overview.pdf, Feb. 2018.

[17] B. Moons, D. Bankman, and M. Verhelst, "Embedded deep neural networks," in *Embedded Deep Learning: Algorithms, Architectures and Circuits for Always-on Neural Network Processing*. Cham: Springer International Publishing, 2019, pp. 1–31, ISBN: 978-3-319-99223-5. DOI: 10.1007/978-3-319-99223-5_1. [Online]. Available: https://doi.org/10.1007/978-3-319-99223-5_1.