FEDERAL UNIVERSITY OF TECHNOLOGY – PARANÁ
GRADUATE SCHOOL OF ELECTRICAL ENGINEERING AND
COMPUTER SCIENCE

PETER ANDREAS ENTSCHEV

EFFICIENT CONSTRUCTION OF MULTI-SCALE IMAGE PYRAMIDS FOR REAL-TIME EMBEDDED ROBOT VISION

DISSERTATION

CURITIBA

2014

PETER ANDREAS ENTSCHEV

EFFICIENT CONSTRUCTION OF MULTI-SCALE IMAGE PYRAMIDS FOR REAL-TIME EMBEDDED ROBOT VISION

Dissertation presented to the Graduate School of Electrical Engineering and Computer Science of the Federal University of Technology – Paraná as partial fulfilment for the degree of "Master of Science" in Computer Engineering.

Supervisor: Hugo Vieira Neto

CURITIBA

2014

Dados Internacionais de Catalogação na Publicação

E61 Entschev, Peter Andreas

Efficient construction of multi-scale image pyramids for real-time embedded robot vision / Peter Andreas Entschev. – 2014.

61 f. : il. ; 30 cm

Supervisor: Hugo Vieira Neto.

Dissertation (Master) – Federal University of Technology – Paraná. Graduate School of Electrical Engineering and Computer Science. Curitiba, 2014.

Bibliography: f. 60-61.

1. Image processing – Mathematical models. 2. Computer vision. 3. Robot vision. 4. Pattern recognition systems. 5. Embedded systems (Computers). 6. Simulation (Computers). 7. Electrical engineering – Dissertations. I. Vieira Neto, Hugo, superv. II. Federal University of Technology – Paraná. Graduate School of Electrical Engineering and Computer Science. III. Título.

CDD (22. ed.) 621.3

Biblioteca Central da UTFPR, Câmpus Curitiba

UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
Câmpus Curitiba

Programa de Pós-Graduação em Engenharia Elétrica e Informática Industrial

Título da Dissertação Nº. 649

"Efficient Construction of Multi-scale Image Pyramids for Real-time Embedded Robot Vision."

por

Peter Andreas Entschev

Orientador: Prof. Dr. Hugo Vieira Neto

Esta dissertação foi apresentada como requisito parcial à obtenção do grau de MESTRE EM CIÊNCIAS – Área de Concentração: Engenharia da Computação do Programa de Pós-Graduação em Engenharia Elétrica e Informática Industrial – CPGEI – da Universidade Tecnológica Federal do Paraná – UTFPR, às 14h do dia 16 de dezembro de 2013. O trabalho foi aprovado pela Banca Examinadora, composta pelos professores doutores:

Prof. Dr. Hugo Vieira Neto
(Presidente – UTFPR)

Prof. Dr. Luiz Eduardo Soares de Oliveira
(UFPR)

Prof. Dr. Alessandro Lameiras Koerich
(PUC-PR)

Prof. Dr. Gustavo Benvenutti Borba
(UTFPR)

Visto da coordenação:

Prof. Ricardo Lüders, Dr.
(Coordenador do CPGEI)

A Folha de Aprovação assinada encontra-se na Coordenação do Programa.

I dedicate this dissertation to my beloved wife, Júlia.

ACKNOWLEDGEMENTS

First of all, I thank my wife Júlia for all her support throughout this long journey, not only during study and work time, but, especially, for her support with my life as a person. Júlia taught me to think of my life as our lives, so I always have someone to share my concerns with and distribute them equally, helping me to keep going on.

I thank all my family, especially my parents, for all their support during my studies, not only at this time, but through all the years of my life.

I appreciate the help of all my friends and colleagues, for their support with their knowledge, or just for sharing some laughs at the bars and pubs that life brings us.

Finally, I would like to thank my supervisor, Dr. Hugo Vieira Neto, without whom this work would never have been possible. I am really grateful for all his support during this entire journey.

ABSTRACT

ENTSCHEV, Peter Andreas. EFFICIENT CONSTRUCTION OF MULTI-SCALE IMAGE PYRAMIDS FOR REAL-TIME EMBEDDED ROBOT VISION. 61 f. Dissertation – Graduate School of Electrical Engineering and Computer Science, Federal University of Technology – Paraná. Curitiba, 2014.

Interest point detectors, or keypoint detectors, have been of great interest for embedded robot vision for a long time, especially those which provide robustness against multiple geometrical variations and changes in scale. The detection of scale invariant features is normally done by constructing multi-scale image pyramids and performing an exhaustive search for extrema in the scale space, an approach that is present in object recognition methods such as SIFT and SURF. These methods are able to find very robust interest points with suitable properties for object recognition, but at the same time are computationally expensive. In this work we present an efficient method for the construction of SIFT-like image pyramids in embedded systems such as the BeagleBoard-xM. The method we present here aims at using computationally less expensive techniques and reusing already processed information in an efficient manner in order to reduce the overall computation time. To simplify the pyramid building process we use binomial filters instead of conventional Gaussian filters used in the original SIFT method to calculate multiple scales of an image. Binomial filters are implementable using fixed-point notation, being a big advantage for many embedded systems that do not provide native floating-point support. We also reduce the amount of convolution operations needed by resampling already processed scales of the pyramid. After presenting our efficient pyramid construction method, we show how to implement it in an efficient manner in a SIMD (Single Instruction, Multiple Data) platform – the SIMD platform we use is the ARM Neon extension available in the BeagleBoard-xM ARM Cortex-A8 processor. SIMD platforms in general are very useful for multimedia applications, where normally it is necessary to perform the same operation over several elements, such as pixels in images, enabling multiple data to be processed with a single instruction of the processor. However, the Neon extension in the Cortex-A8 processor does not support floating-point operations, so the whole method was carefully implemented to overcome this limitation. Finally, we provide some comparison results regarding the method we propose here and the original SIFT approach, including performance regarding execution time and repeatability of detected keypoints. With a straightforward implementation (without the use of the SIMD platform), we show that our method takes approximately $\frac{1}{4}$ of the time taken to build the entire original SIFT pyramid, while repeating up to 86% of the interest points found with the original method. With a complete fixed-point approach (including vectorization within the SIMD platform) we show that repeatability reaches up to 92% of the original SIFT keypoints while reducing the processing time to less than 3%.

Keywords: Multi-scale image pyramids, Interest point detection, Embedded robot vision

RESUMO

ENTSCHEV, Peter Andreas. CONSTRUÇÃO EFICIENTE DE PIRÂMIDES DE IMAGENS EM MULTIESCALA PARA VISÃO ROBÓTICA EMBARCADA EM TEMPO-REAL. 61 f. Dissertation – Graduate School of Electrical Engineering and Computer Science, Federal University of Technology – Paraná. Curitiba, 2014.

Detectores de pontos de interesse, têm sido de grande interesse para visão robótica embarcada, especialmente aqueles que possuem robustez a múltiplas variações geométricas e mudanças em escala. A detecção de características invariáveis a escala é normalmente realizada com a construção de pirâmides de imagens em multiescala e pela busca exaustiva de extremos no espaço de escala, uma abordagem presente em métodos de reconhecimento de objetos como SIFT e SURF. Esses métodos são capazes de encontrar pontos de interesse bastante robustos, com propriedades adequadas para o reconhecimento de objetos, mas são ao mesmo tempo computacionalmente custosos. Neste trabalho é apresentado um método eficiente para a construção de pirâmides de imagens em sistemas embarcados, como a plataforma BeagleBoard-xM, de forma similar ao método SIFT. O método aqui apresentado tem como objetivo utilizar técnicas computacionalmente menos custosas e a reutilização de informações previamente processadas de forma eficiente para reduzir o tempo computacional. Para simplificar o processo de construção de pirâmides, o método utiliza filtros binomiais em substituição aos filtros Gaussianos convencionais utilizados no método SIFT original para calcular múltiplas escalas de uma imagem. Filtros binomiais são implementáveis utilizando notação ponto-fixo, sendo uma grande vantagem para muitos sistemas embarcados que não possuem suporte nativo a ponto-flutuante. A quantidade de convoluções necessária é reduzida pela reamostragem de escalas já processadas da pirâmide. Após a apresentação do método para construção eficiente de pirâmides, é apresentada uma maneira de implementação eficiente do método em uma plataforma SIMD (*Single Instruction, Multiple Data*) – a plataforma SIMD usada é a extensão ARM Neon disponível no processador ARM Cortex-A8 da BeagleBoard-xM. Plataformas SIMD em geral são muito úteis para aplicações multimídia, onde normalmente é necessário realizar a mesma operação em vários elementos, como pixels em uma imagem, permitindo que múltiplos dados sejam processados com uma única instrução do processador. Entretanto, a extensão Neon no processador Cortex-A8 não suporta operações em ponto-flutuante, tendo o método sido cuidadosamente implementado de forma a superar essa limitação. Por fim, alguns resultados sobre o método aqui proposto e o método SIFT original são apresentados, incluindo seu desempenho em tempo de execução e repetibilidade de pontos de interesse detectados. Com uma implementação direta (sem o uso da plataforma SIMD), é mostrado que o método aqui apresentado necessita de aproximadamente $\frac{1}{4}$ do tempo necessário para construir a pirâmide do método SIFT original, ao mesmo tempo em que repete até 86% dos pontos de interesse. Com uma abordagem completamente implementada em ponto-fixo (incluindo a vetorização com a plataforma SIMD) a repetibilidade chega a 92% dos pontos de interesse do método SIFT original, porém, reduzindo o tempo de processamento para menos de 3%.

Palavras-chave: Pirâmides de imagens em multiescala, Detecção de pontos de interesse, Visão robótica embarcada

LIST OF FIGURES

LIST OF TABLES

LIST OF ACRONYMS

GHz	Gigahertz
MB	Megabyte
MHz	Megahertz
MIMD	Multiple Instruction, Multiple Data
MISD	Multiple Instruction, Single Data
MP	Megapixel
SIFT	Scale Invariant Feature Transform
SIMD	Single Instruction, Multiple Data
SISD	Single Instruction, Single Data
SURF	Speeded Up Robust Features

CONTENTS

1 INTRODUCTION

Efficient object recognition has always been one of the main challenges in the computer vision field of study, from both recognition performance and time consumption points of view. In general, the better performance of some particular task, more computationally complex and expensive the task is, and this is not different for computer vision applications, including object recognition.

Autonomous mobile robots will benefit from physically small, cost-effective and power efficient systems. Nowadays, several interesting embedded systems are available, such as the BeagleBoard-xM (COLEY, 2010) and the Raspberry Pi (HALFACREE; UPTON, 2012) boards. These two boards are physically small and low-power consuming, being based on ARM processor cores. These platforms are able to run the Linux operating system and a myriad of open software and code libraries, such as the Open Source Computer Vision Library, or simply OpenCV (BRADSKI; KAEHLER, 2008).

Energy efficiency is a very important characteristic for mobile robot systems, perhaps the most important, which defines robot autonomy. However, there is a price to pay with these small and low-power consuming systems, the available processing power is usually significantly smaller than physically larger systems, such as desktop PCs and servers. Normally, these embedded processors run at a much lower frequency, and the complete system includes much less memory. For this reason, computer vision techniques that are efficient from both recognition and computational points of view are needed.

The hypothesis we want to verify in this work is the possibility of calculating multi-scale image pyramids in real-time with a low-power embedded system, mainly based on binomial filters to obtain multiple scales of an image, later being used for detection of stable points of interest, relying on high-curvature corners. With the proposed method for construction of efficient multi-scale image pyramids, including a vectorized fixed-point implementation model of the pyramid, we expect to contribute towards real-time object recognition for autonomous mobile robots.

In this work we introduce an efficient method for the computation of multi-scale image pyramids, aimed at embedded robot vision systems, more specifically using the BeagleBoard-xM as implementation platform. The board has a very interesting design for robot vision applications, as it includes a low-power ARM Cortex-A8 processor, with up to 1 GHz of clock and capabilities for vectorized data processing (SIMD – Single Instruction, Multiple Data) using the ARM Neon extension. A significant amount of memory is included, a DDR class chip of 512 MB. The board includes also a dedicated camera port where a CMOS camera can be connected directly to the processor's data bus, enabling direct access without any USB or FireWire interface overhead. Fully programmable CMOS cameras are available for direct connection to the BeagleBoard-xM, supporting resolutions of up to 5 MP, which are configurable to acquire images in specific sizes to reduce the necessary image acquisition bandwidth, optimizing computational resources by avoiding unnecessary resampling processes, often needed for multi-scale image pyramid algorithms. The BeagleBoard-xM features yet another processor core in the form of a Digital Signal Processor (DSP), of the c64x+ family, developed by Texas Instruments, which can be used to accelerate some intense signal processing operations.

We base our studies mainly in the works of the Scale Invariant Feature Transform (LOWE, 2004), or SIFT, and the Speeded-Up Robust Features (BAY et al., 2008), or SURF, which are the most common methods for object recognition, due to their high robustness against many geometrical transformations, especially changes in scale. Both methods use multi-scale image pyramids in order to detect features regardless of their scale. These two methods have slightly different approaches to detect interest points, but both rely on the same basic principles, that is the image filtering to achieve multiple scales, by later performing an exhaustive search to identify extrema. With both methods not only is possible to identify these interest points, but also to estimate their scales.

The method we propose in this work uses less expensive computational methods than their counterparts in the literature in order to calculate multiple scale levels of an image pyramid. It also reuses already processed information to calculate remaining scale levels, providing high efficiency in terms of reducing the amount of calculations. To calculate multiple scale levels we replace conventional Gaussian filters, as the ones used in the original SIFT method (LOWE, 2004), by binomial filters (CROWLEY; RIFF, 2003), which have as main advantage the possibility of being computable with a fixed-point notation. Many embedded systems do not support native floating-point operations, and its emulation can be computationally expensive, for such systems the exclusive usage of fixed-point numbers can imply in a significant performance gain. The processor contained in the BeagleBoard-xM

supports native floating-point calculations, but only in the non-vectorized part of the processor (SISD – Single Instruction, Single Data architecture), while the Neon extension and the DSP only support native fixed-point notation.

The multi-scale image pyramid construction method we propose uses some parts of the information that were already processed as a method to reduce the amount of computation. Multi-scale image pyramids use multiple octaves to perform calculations more efficiently, without needing to perform image filtering in its original dimensions, reducing the amount of data to process and also avoiding the need of large filters to achieve higher scale levels. Multiple octaves are achieved by subsampling the original image multiple times, in each step reducing the image dimensions to about half its original dimensions. To achieve high scale levels while avoiding the need of relatively large filters and reducing the amount of image information to process, we subsample the image to about half its dimensions multiple times, achieving multiple octaves. In our method, some particular scale levels of each octave are used to calculate scales from other octaves, reducing the amount of filtering steps.

In modern computing, just a straightforward implementation is not enough, as many methods to speed up data processing in current processors rely on processing multiple data simultaneously. In this work, we do not aim at a straightforward implementation of the method, only implementing it in a regular basis, but also to study and modify the implementation so it can take the most advantage of the hardware available, using the architecture's best features to process information as fast as possible. To reduce the amount of clock cycles taken to process data, the ARM Neon extension can be used, processing multiple data at the same time, with a single processor instruction.

We present performance experiments and the results achieved by using our method, separating these results in two main categories. The first compares processing time, evaluating both a straightforward (SISD) floating-point implementation and a vectorized (SIMD) fixed-point implementation, comparing both to the original SIFT method. The second category tests the repeatability of detected interest points for both implementations of our method, while comparing the results to the ones obtained with the original SIFT method.

The results obtained are very compelling, drastically reducing the processing time of the original SIFT method, while repeating the majority of points of interest of the original SIFT method.

2 IMAGE PYRAMIDS

Image pyramids are vastly used in image processing, especially when it comes to interest point detection, which is a very important step for object recognition. In this chapter the aim is to introduce what image pyramids are, their uses and why they are so important.

This chapter is divided in three parts. We start by explaining the role of an image pyramid in the field of image processing, more specifically for object recognition. After that, we then follow for examples of two of the most well-known object recognition methods, namely SIFT (LOWE, 2004) and SURF (BAY et al., 2008), both of which use image pyramids to detect points of interest. In this work we will only concentrate in methods to build image pyramids, without considering feature description and feature matching methods.

2.1 ROLE IN OBJECT RECOGNITION

There are many characteristics that are desirable in a robust method for object recognition, like invariance to rotation, translation, changes in scale, among other geometrical changes that might occur in the real world. To identify features that are possible to be recognized regardless of scale, one of the most common methods is the construction of multi-scale Laplacian pyramids. With multi-scale Laplacian pyramids, not only it is possible to identify such features, but also to estimate their scales.

Multi-scale Laplacian pyramids were originally proposed as a method for image encoding for data-line transmissions, providing the user first with a rough representation that improves gradually over time when additional data is received (BURT; ADELSON, 1983). These pyramids were later improved in (CROWLEY; STERN, 1984), and had one of their first uses for detection of interest points in (CROWLEY; PARKER, 1984).

As previously explained, a key role in many methods for object recognition is the identification of stable points of interest, also called keypoints – both expressions will be used as synonyms throughout this work. To identify points of interest, it is necessary to scan the

whole image in order to find unambiguous features of an object. If a multi-scale image pyramid is available, with the aim of achieving scale invariance, the image must be scanned through all the scales of the pyramid.

An unambiguous feature is a point in the object that represents a distinguishable characteristic only in that specific position of the object, thus ideally being identifiable independently of the viewpoint of the object. Features that represent unambiguous characteristics of an object are usually blobs, locations that differ from their surrounding region in properties such as brightness, or corners, blobs with high curvature degrees.

Figure 1 shows an example of ambiguous and unambiguous locations. The region highlighted by the blue square shows an ambiguous point, a point that relies on a simple edge, thus, when performing object recognition, this point would be detectable on many different positions over the same edge of the object. On the other hand, the red square shows a point that is detectable by both SIFT and SURF as a keypoint, relying on a corner, which would only be matched against another similar corner, ideally in the same object.

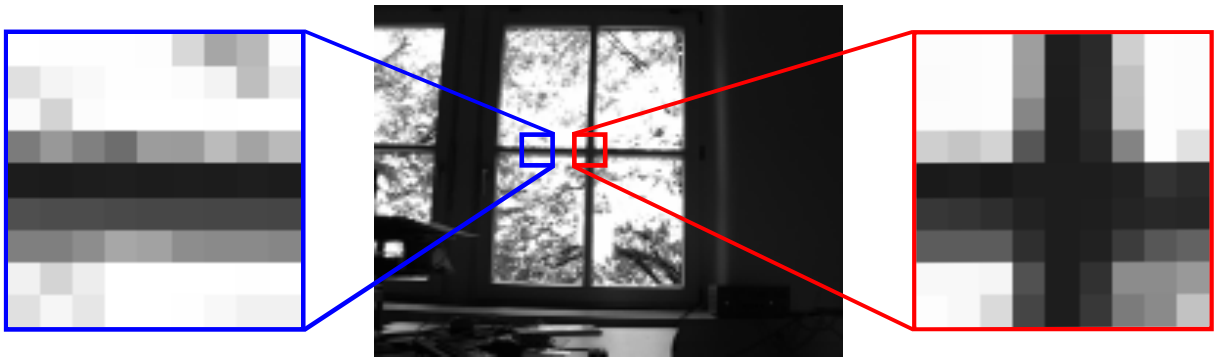

Figure 1: Example of ambiguous and unambiguous points. An ambiguous point, relying on an edge, is shown in the blue square and an unambiguous point, relying on a corner, is shown in the red square.

There are many available methods to identify points of interest in an image. In the following subsections, two of the most relevant ones will be presented.

2.2 SIFT KEYPOINT DETECTION

The Scale Invariant Feature Transform, or SIFT, was first published in (LOWE, 1999) and later extended in (LOWE, 2004). To perform object recognition, it first detects points of interest, regardless of scale. To be able to perform the detection independently of scale, a multi-scale image pyramid is constructed and afterwards scanned for possible keypoints.

The construction of the initial multi-scale pyramid is done by convolving the original

image multiple times, each time with a Gaussian kernel with σ corresponding to the desired scale. After each convolution, the resulting image is stored, keeping track of the resulting scale. As a Gaussian filter acts as a low-pass filter, attenuating high spatial frequencies (i.e., edges), the resulting image will look like a blurred or defocused version of the original image, and the higher the value of σ is, the higher the amount of blur will be.

Another approach frequently used, targeting reduction in computational cost, is applying cascaded convolution. This means that the previous level is used to compute the next one. With this approach, the kernel length to convolve each scale is kept constant, while for direct convolution the kernel length becomes larger as scale grows. The mathematical relationship between the current scale σ_i of the image and the scale σ_k of the Gaussian kernel being applied is shown in Equation 1 (CROWLEY et al., 2002).

$$\sigma = \sqrt{\sigma_i^2 + \sigma_k^2} \tag{1}$$

In Equation 1, σ represents the resulting scale – in our case, the image with initial scale σ_i, after being convolved with a Gaussian kernel with σ_k. For example, supposing we have an image with scale $\sigma = 1$, after filtered with a kernel holding scale $\sigma = \sqrt{3}$, the resulting scale is $\sigma = 2$. At the scale $\sigma = 2$ it is possible to downsample the image to half of its original dimensions with minimal loss of information (BURT; ADELSON, 1983).

To achieve higher scale levels without the need of very large and computationally expensive Gaussian filter kernels, the original images are downsampled after filtering multiple times to half their horizontal and vertical dimensions, obtaining different octaves (CROWLEY; RIFF, 2003). Each octave has half the dimensions of the previous octave and the process to compute the scale-space (in this work scale-space will be used as a synonym to multi-scale image pyramid) is then repeated. The number of octaves is determined either by an empirical number or until the image achieves such a small dimension that it is not possible to keep convolving it with Gaussian filters.

Once the scale-space is calculated, the next step is calculating the differences between scale levels, the result of this process is called difference-of-Gaussians. This is done by a simple arithmetic subtraction, pixel by pixel, of the image matrices of adjacent scales. As the filtering with Gaussian kernels acts as a low-pass filter, the process of calculating the difference between two images, or the difference-of-Gaussians, acts as a high-pass filter, leaving only information on the edges present in the image.

Figure 2 exemplifies the representation of a SIFT multi-scale image pyramid.

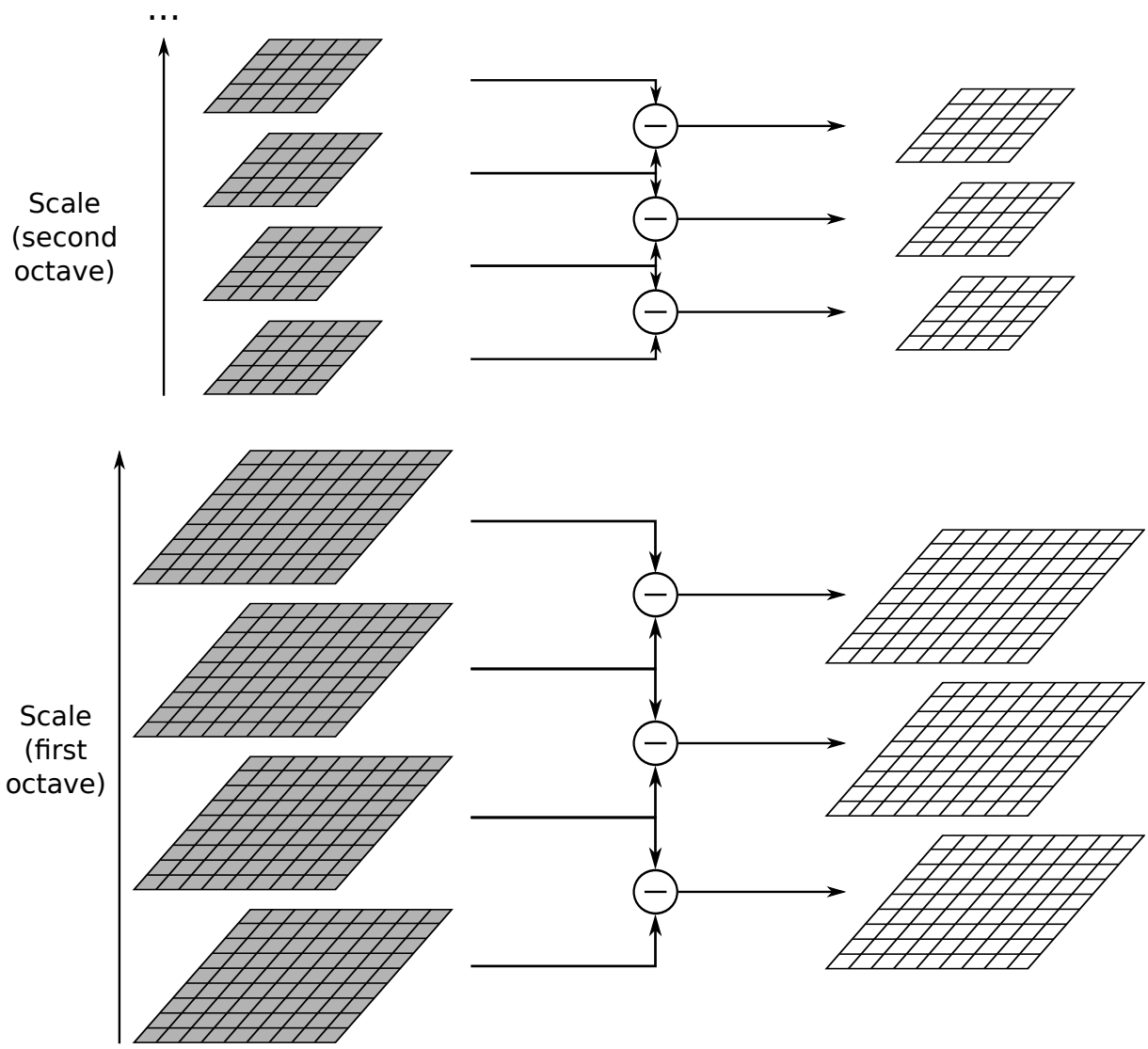

Figure 2: SIFT multi-scale image pyramid. Two octaves are shown, the top octave has approximately half the dimensions of the bottom octave. Adopting odd dimensions of $2^N + 1$ results in convenient and inexpensive subsampling. The left side represents the multi-scale pyramid, after convolutions with Gaussian filters, and the right side represents the resulting difference-of-Gaussians of the images on the left.

In Figure 3, we show an example image with its different scale levels and the corresponding differences-of-Gaussians. The first row contains the original image, without any processing. The second row contains four scale levels generated by filtering the image with proper Gaussian kernels, from left to right the scales are $\sigma = 1$, $\sigma = \sqrt{2}$, $\sigma = 2$ and $\sigma = 2\sqrt{2}$. The third row contains the three differences-of-Gaussians obtainable from the existing scales, from left to right, the first is the difference from the first and second images of the second row, followed by the difference from the second and third images and then by the third and fourth images.

After the scale-space is built and the difference-of-Gaussian images are computed,

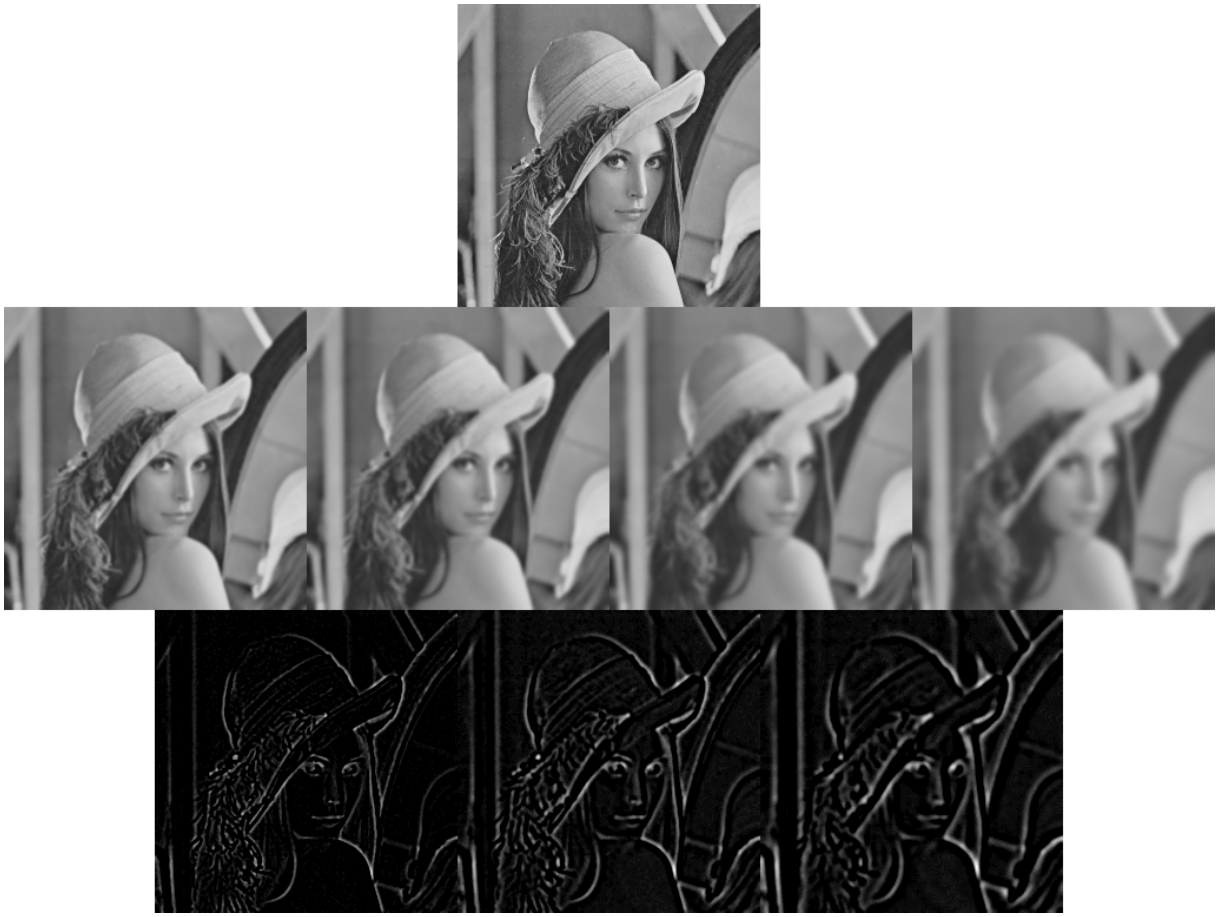

Figure 3: Multi-scale image and difference-of-Gaussian example. The first row contains the original image. The second row contains the image after Gaussian filtering with scales $\sigma = 1$, $\sigma = \sqrt{2}$, $\sigma = 2$ and $\sigma = 2\sqrt{2}$, from left to right. The third row contains the three difference-of-Gaussians of the images in the second row, obtained by subtracting the first and second images, second and third and third and fourth images, respectively from left to right – for better visualization, the contrast of the difference-of-Gaussian images was increased.

local extrema can be detected, by using three subsequent difference-of-Gaussian image matrices. This process is done by taking the central scale in an octave and comparing every pixel to its 3D-neighborhood. This neighborhood is defined by the 8-neighborhood of the pixel in the same scale and the 9-neighborhood in the previous and subsequent scales (a cube of $3 \times 3 \times 3$ pixels). If the reference pixel has an intensity value that is maximum or minimum compared to its neighborhood, it is selected as a keypoint. In the literature, this process is often called extrema detection (LOWE, 2004) or non-maximum suppression (BAY et al., 2008). A graphical representation of this process is shown in Figure 4.

Once identified as a local maximum or minimum, a descriptor can be calculated around each keypoint. The descriptor of a keypoint is also very important for the object recognition process – it needs to incorporate a series of aspects of the region, so as to ensure that it is as unique as possible and can be used to identify that particular region of the object among

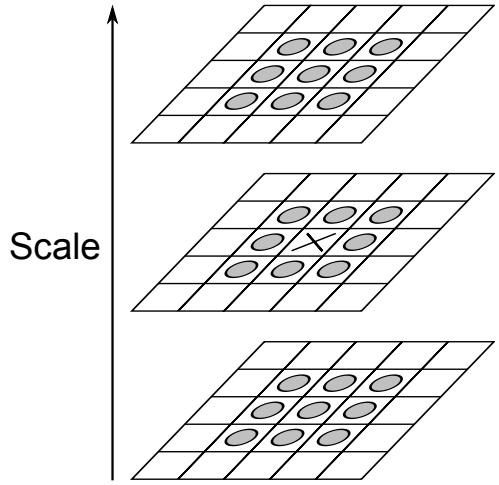

Figure 4: Neighborhood representation for SIFT scale-space extrema detection. The central pixel marked with a diagonal cross is compared to its 26 neighbors, marked with circles, if it is maximum or minimum, it is an interest point. This step is performed in the difference-of-Gaussian images.

hundreds or thousands of other descriptors. As previously mentioned, in this work we will only concentrate in the multi-scale image pyramid and the identification of points of interest.

In order to discard low-curvature points that are detected, SIFT uses a Hessian matrix H to calculate the principal curvatures of an interest point, as the one in Equation 2, where D_{xx}, D_{xy} and D_{yy} indicate derivatives of the difference-of-Gaussian function. As described in (LOWE, 2004), poorly defined peaks have large principal curvatures across edges, but small ones in the perpendicular direction.

$$H = \begin{bmatrix} D_{xx} & D_{xy} \\ D_{xy} & D_{yy} \end{bmatrix} \tag{2}$$

The ratio of the eigenvalues of the Hessian matrix should be smaller than some empirically defined number, otherwise the interest point is discarded as it has low curvature characteristics. For the ratio calculation, we need to first find the trace (Equation 3) and the determinant (Equation 4) of the Hessian matrix H.

$$Tr(H) = D_{xx} + D_{yy} \tag{3}$$

$$Det(H) = D_{xx}D_{yy} - (D_{xy})^2 \tag{4}$$

The inequality of Equation 5 must hold, otherwise, the keypoint is discarded. The r constant is empirically defined.

$$\frac{Tr(H)^2}{Det(H)} < \frac{(r+1)^2}{r} \tag{5}$$

2.3 SURF KEYPOINT DETECTION

The Speeded Up Robust Features method, or SURF (BAY et al., 2008), has a similar approach to that of SIFT, also computing a multi-scale image pyramid, but using a different technique to build the scale-space. First, an integral image is computed using Equation 6 (VIOLA; JONES, 2001).

$$I_{\Sigma}(x,y) = \sum_{i=0}^{i \leq x} \sum_{j=0}^{j \leq y} I(i,j) \tag{6}$$

As can be deduced from Equation 6, the integral image is nothing more than the addition of all intensity values in sequence, from left to right, then top to bottom, starting from the origin in the top-left pixel of the image, which is equivalent to a discretized double integral equation, and this is the origin of the name given for this technique.

The advantage of using an integral image is a fast calculation of the sum of intensity values for a specific rectangular area of the interest image. This can be achieved by four element accesses, and three arithmetic operations (one addition and two subtractions). For example, to calculate the sum of pixel values in the area represented by the gray rectangle in Figure 5, one can use Equation 7.

$$I_{\Sigma}(Area) = I_{\Sigma}(P_1) + I_{\Sigma}(P_4) - I_{\Sigma}(P_2) - I_{\Sigma}(P_3) \tag{7}$$

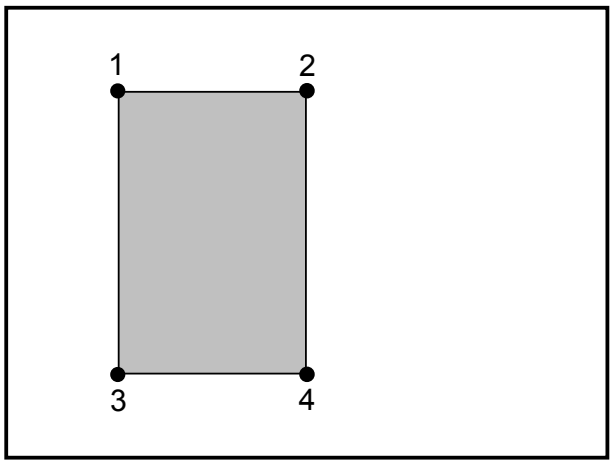

Figure 5: Integral image area, defined by the gray rectangle with vertex coordinates 1**,** 2**,** 3 **and** 4**.**

In SURF, interest points are also detected based on a Hessian matrix (Equation 2). The Hessian matrix used in SURF relies on Gaussian derivatives, which can be approximated as box filters. Examples of a Gaussian derivatives constructed as box filters are shown in Figure 6, holding a scale of approximately $\sigma = 1.2$.

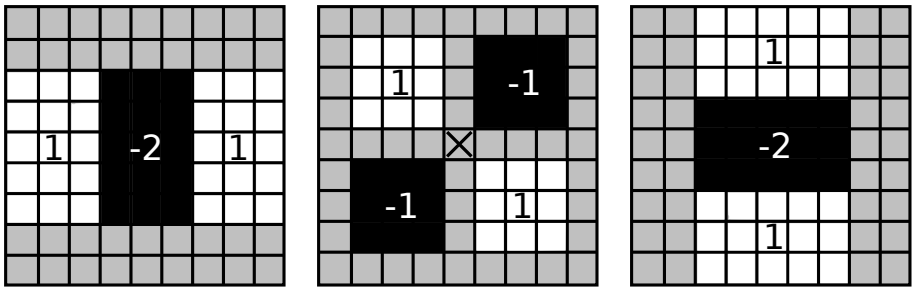

Figure 6: 9×9 Gaussian derivative matrix examples in *xx*-direction, *xy*-direction, *yy*-direction, with positive weights in white areas, negative weights in black areas and null weights in gray areas, where the central pixel is the pixel of interest. These example matrices are approximations of Gaussian derivative filters with scale $\sigma = 1.2$.

The areas in Figure 6 can be calculated by applying Equation 7 in the integral image, thus reducing the time needed to compute the addition of the intensity values for that area. As the number of operations is independent of the matrix size, the process takes constant time regardless of the box filter size and for this reason when the box filter grows in size, this reduction in computational time becomes more relevant.

Similarly to SIFT, the SURF approach also uses multiple scales to find points of interest. However, differently from the SIFT approach, the sizes of box filters are increased along with the increasing step size, without downsampling the image.

In order to achieve larger scales, the value of σ grows, and for a Gaussian derivative box filter, the box size gets larger. For the first octave, the original SURF method uses 9×9 box filters, then 15×15, 21×21 and 27×27, so it has an order of growth of 6 pixels per dimension. For the second octave, this growth doubles (resulting in 12 pixels), for the third octave this growth doubles again (resulting in 24 pixels) and it keeps growing until the last octave is achieved. The initial box size of each octave is always the second box size of the previous octave such as the first box size for the first octave is 9×9, for the second octave is 15×15, for the third octave is 27×27, and so on.

Once the scale-space computation is complete, a non-maximum suppression in a $3 \times 3 \times 3$ neighborhood is computed. For SURF, a different method (NEUBECK; GOOL, 2006) than the one used in SIFT is applied, it is supposed to be faster because it does not perform an exhaustive neighborhood search as in SIFT, but has a similar effect.

3 FAST MULTI-SCALE IMAGE PYRAMIDS

In this chapter, we present a fast multi-scale image pyramid that was first presented in (ENTSCHEV; VIEIRA NETO, 2013). This pyramid aims at reducing computation time, especially for embedded systems, by using mathematical approximations and reusing already processed information whenever possible. Although some approximations are used, it is expected that the keypoint detection performance of this method will be comparable to its original definition, proposed in (LOWE, 2004).

We divide this chapter in four main sections. The first section introduces the binomial kernels, which are used to filter the image for the purpose of obtaining multiple scales. The second section provides an overview of image resampling with a nearest neighbor approach, used in our method to obtain the first scale level of a new octave in the pyramid, followed then by a section on resampling by bilinear interpolation, used to compute the last levels of each octave. Finally, the last section explains how the whole pyramid is constructed, using all the techniques previously introduced.

The contribution proposed in this chapter lies in the usage of binomial filters instead of the conventional Gaussian filters, allied to the exclusive usage of image with $2^N + 1$ dimensions and efficient downsampling to obtain multiple octaves and upsampling to obtain the highest scale levels for each octave. As will be seen throughout this chapter, the usage of images with $2^N + 1$ dimensions is particularly important for the resampling processes as proposed.

3.1 BINOMIAL FILTERING

Originally, SIFT uses Gaussian kernels to obtain multi-scale representations in order to detect points of interest in images. Certain Gaussian kernels can be approximated in the form of binomial kernels, and this expresses a significant advantage for embedded systems.

As embedded systems provide limited computational power, the more the computational process can be reduced, the faster it can be executed, aiming near real-time

computations. This involves finding techniques that are computationally simpler, as is the case of using binomial filters, but careful implementation of these techniques has also a very important role.

The nature of binomial kernels make them very attractive for systems that do not support native floating-point operations. While floating-point operations can be emulated in software, this task might be computationally expensive. To avoid the need of floating-point operations, an approach frequently used is to find precise enough operations that can approximate the original computational problem using only fixed-point numbers, and that is the aim of the binomial kernels in this work, providing accurate enough approximations for the Gaussian kernels.

The hardware that we are using in our experiments, the BeagleBoard-xM (COLEY, 2010), has native support to floating-point operations only in its ARM core and only for SISD (Single Instruction, Single Data) operations. The SIMD (Single Instruction, Multiple Data) instructions available with the Neon SIMD extension only support fixed-point operations. The Texas Instruments c64x+ DSP core, also present in the BeagleBoard-xM media processor, has the same limitation as the ARM Neon extension, supporting only native fixed-point data processing.

With binomial kernels, it is not possible to achieve every desired scale separation, as can be done with Gaussian kernels. However, it is possible to achieve scale separations in the order of $\sigma = \sqrt{2}$, which suffices for the process of detection of keypoints invariant to scale changes (LOWE, 2004).

For this work, we are particularly interested in two binomial kernels, $\frac{1}{4} \times [1\ 2\ 1]$, and its auto-convolution, $\frac{1}{16} \times [1\ 4\ 6\ 4\ 1]$. These two kernels represent $\sigma = \frac{1}{\sqrt{2}}$ and $\sigma = 1$ (CROWLEY; RIFF, 2003), respectively. Using these kernels, recurring to the cascaded convolution property given in Equation 1, scale separations of $\sigma = \sqrt{2}$ are then achievable.

Both Gaussian and binomial kernels, when convolved with images, act as low-pass filters, thus, attenuating high-frequency information, such as edges. Visually, the effect seen is that the image has been blurred or that lenses were out of focus (see Figure 3).

Figure 7 shows an example of an image filtered with Gaussian kernels on the top row and binomial filters on the bottom row. The resulting scales are, from left to right, $\sigma = 1$, $\sigma = \sqrt{2}$ and $\sigma = 2$ for both rows. Visually, it is not possible to tell any difference between both approaches. However, some difference might be noticed when comparing values in the pixel level. The maximum difference is of two gray levels and is observed for 0.5% of the pixels in

$\sigma = 1$ and 0.02% in $\sigma = \sqrt{2}$. A difference of one gray level is observed for 14.5% of the pixels in $\sigma = 1$, 11.5% of the pixels in $\sigma = \sqrt{2}$ and 10% of the pixels in $\sigma = 2$.

Figure 7: Comparison of image filtered with Gaussian and binomial kernels. The top row contains image samples filtered with Gaussian kernels, with resulting scales of $\sigma = 1$, $\sigma = \sqrt{2}$ and $\sigma = 2$ from left to right. The bottom row contains the binomial counterparts, with the same resulting scales.

3.1.1 ACCURACY OF FILTERS

In Figure 8 a comparison of accuracy between the Laplacian and the approximation of the Laplacian with a difference of Gaussians and a binomial kernel is shown. For this comparison, we used the same values proposed in (CROWLEY; RIFF, 2003), which are $\sigma_{lap} = 1.7$, $\sigma_{dog} = \sqrt{2}$ and $\sigma_1 = \sqrt{2} * \sigma_{dog} = 2$. As described also in (CROWLEY; RIFF, 2003), the value of $\sigma = 1.7$ for the Laplacian is the number that approximates the difference-of-Gaussians for two images with scales $\sigma = 2$ and $\sigma = \sqrt{2}$, and it was found by minimizing the the sum of the squared differences. These values provides a difference in scale in the order of $\sigma = \sqrt{2}$, this separation is suffices to find interest points in most scales (LOWE, 2004). As for the binomial kernel, it is visible that it is a coarser approximation of the difference-of-Gaussians, but it still preserves the properties of scale invariance, while being computationally less expensive.

To calculate the Laplacian, the second derivative was used as given in Equation 8, while the difference of Gaussians is calculated by Equation 9.

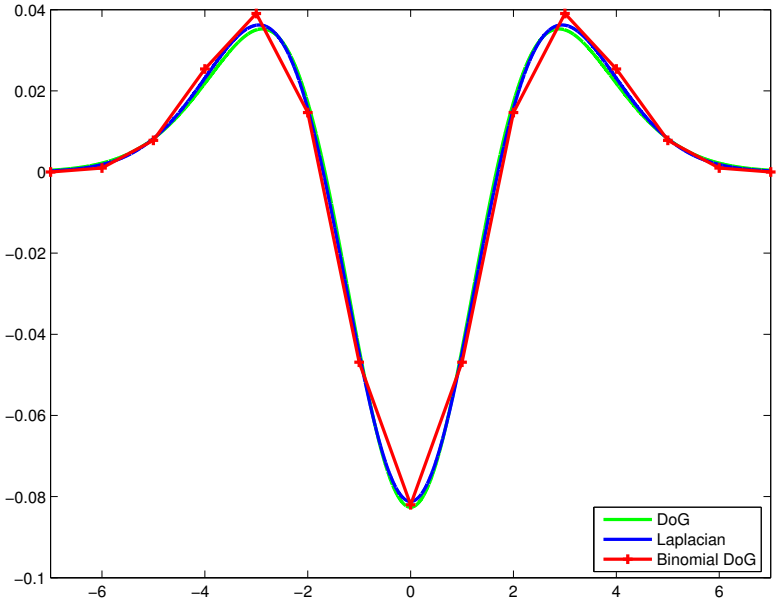

Figure 8: Accuracy comparisons between Binomial difference of Gaussians, difference of Gaussians and actual Laplacian. (CROWLEY; RIFF, 2003)

$$\nabla^2 G(r,\sigma_{lap}) = \frac{r^2-\sigma_{lap}^2}{\sigma_{lap}^5\sqrt{2\pi}} e^{-\frac{1}{2}\frac{r^2}{\sigma_{lap}^2}} \tag{8}$$

$$DoG(r,\sigma_{dog}) = \frac{1}{\sigma_1\sqrt{2\pi}} e^{-\frac{1}{2}\frac{r^2}{\sigma_1^2}} - \frac{1}{\sigma_{dog}\sqrt{2\pi}} e^{-\frac{1}{2}\frac{r^2}{\sigma_{dog}^2}} \tag{9}$$

The binomial difference of Gaussians approximation shown in Fig. 8 was obtained by convolving the kernel $\frac{1}{4}\times[1\ 4\ 6\ 4\ 1]$ twice, obtaining $\sigma_{dog}=\sqrt{2}$, and then twice more, obtaining a $\sigma_1=2$, which are equivalent to the values used for the real difference of Gaussians and finally calculating the difference of these values.

3.2 NEAREST NEIGHBOR RESAMPLING

In the field of image processing, other techniques of image resampling are usually preferred to nearest neighbor resampling. Mathematically, nearest neighbor resampling of images is one of the simpler approaches, but it leads to some effects that are often undesired.

When downsampling an image using the nearest neighbor approach, the first effect that can be observed is the loss of information, because some pixels are simply discarded. The loss of information is more significant when the image contains significant high-frequency areas, represented by edges. In the process of upsampling images with the nearest neighbor approach, there will be repeated pixels, and such copies of pixels only consume more storage

space, without actually increasing the information present in the image.

Another effect that can be observed with the nearest neighbor approach for image resampling is spatial aliasing, unless proper filtering is done previous to resampling. If the image has to be downsampled and no proper filtering is done, higher frequencies are disguised into lower frequencies, which is called aliasing (GONZALEZ; WOODS, 2006). The effect of aliasing in images can be visually perceived as jagged contours, for example.

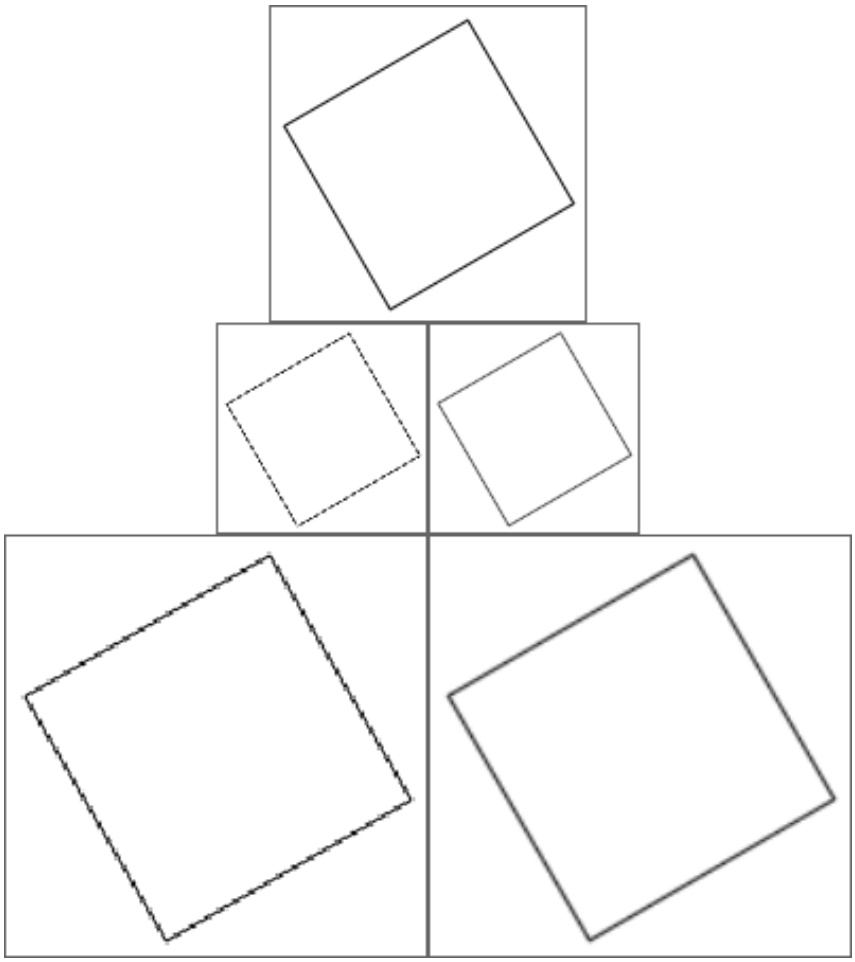

Figure 9: Nearest neighbor and bilinear resampling example. On top, there is the image of square rotated by 30 degrees anti-clockwise on a 200×200 pixels frame. The central row contains the same square with the frame downsampled to 150×150 pixels with the nearest neighbor approach on the left and bilinear interpolation on the right. The bottom row contains the frame upsampled to 250×250 pixels by the nearest neighbor approach on the left and bilinear interpolation on the right.

Figure 9 shows the image of a square rotated by 30 degrees anti-clockwise on a frame of 200×200 pixels and its downsampling and upsampling by both nearest neighbor and bilinear interpolation (see Section 3.3) approaches. In both cases, the bilinear interpolation has a superior visual appearance, being very similar to the original sample but with thicker contours when upsampled and thinner contours when downsampled. The results generated with the

nearest neighbor approach show a strong jaggedness, especially in the upsampling case, but shows also some jaggedness and discontinuity in the lines when downsampled.

When calculating a multi-scale pyramid, a nearest neighbor sampling can be performed for the particular task of generating the first scale level of an octave, provided that some previous information was already processed. For instance, an image that contains a scale of $\sigma = 2$ already contains the same amount of information as its half-resolution counterpart contains, thus, it can be resized to half its resolution with minimal loss of information.

As previously explained, when the scale of an image is increased (without resampling it, i.e., by filtering it with a Gaussian or binomial kernel), a blurring effect is noticed. This blurring effect happens because the information of a pixel contains also information of its adjacent pixels. For an image with a scale of $\sigma = 2$, it is then possible to resize the image by only taking every odd pixel, in both horizontal and vertical directions, as the odd pixels already incorporate some information of the adjacent even pixels.

3.3 BILINEAR RESAMPLING

Resampling images using bilinear interpolation is more suitable than the nearest neighbor approach for most applications, such as previewing pictures in a computer screen. It is a more complex method than the nearest neighbor, but still efficient for many applications, including previewing pictures in a modern computer.

To properly resample an image with the nearest neighbor method, for every pixel in the resulting image, only one pixel value is needed to compute the result. For bilinear resampling, for every resulting pixel, four pixel values are used from the input image, each with a weight inversely proportional to the distance to the pixel being resampled.

Differently from the nearest neighbor approach, downsampling by bilinear interpolation does not simply discard some information, but aims to compress the information, so that is not completely lost. As we previously mentioned in Section 3.2, when downsampling is done with the nearest neighbor approach, the only way information is not completely lost is when the image is pre-filtered with a low-pass filter, previous to the downsampling.

When upsampling with bilinear interpolation, the result is also different from the nearest neighbor method. The nearest neighbor upsampling is processing by simply adding copies of already existing information, so the new pixels are nothing more than copies of one of its neighbors. The bilinear upsampling acts by estimating a new value for the pixel being added, taking the neighboring pixel values and calculating the new value according to the distance to

them.

3.4 PYRAMID CONSTRUCTION PROCESS

For the process of construction of fast SIFT multi-scale image pyramids, we use mainly the three approaches introduced in the previous sections, namely the binomial filtering to achieve multiple scales, nearest neighbor for downsampling and bilinear interpolation for upsampling, each being responsible for a specific task in the method. The method we describe here, targeting the construction of the SIFT multi-scale image pyramid in a fast manner, was mainly inspired by the work described in (CROWLEY; RIFF, 2003).

Unlike the original SIFT approach (LOWE, 2004), we start by proposing the usage of computationally more efficient kernels. These are the binomial kernels which were studied in detail in (CROWLEY; RIFF, 2003), where they have shown to provide a very good approximation for the Gaussian kernels, used in SIFT to filter out high-frequencies in the images. We also provide some information about the binomial filters accuracy in Subsection 3.1.1, based on the evaluation of precision of the different kernel types, also provided in (CROWLEY; RIFF, 2003).

We propose here the usage of a different approach to build image pyramids, targeting the reduction of computational operations required, thus, reducing also the computation time. In this method, we aim to approximate complex mathematical functions into simpler, but accurate enough, representations. We also reuse the already processed information whenever possible, avoiding the execution of complex tasks that may be done by some sort of simplification.

First of all, we avoid computing convolutions with bidimensional kernels by using exclusively equivalent unidimensional separable kernels, thus, passing them first horizontally, then vertically. In image processing, the usage of bidimensional kernels is avoided whenever possible because it is significantly more expensive than using the unidimensional separable equivalents, but the result is exactly the same. Although unidimensional kernels are preferred, not every kernel is separable. In other words, not every possible kernel can be expressed as a simpler unidimensional kernel and be passed twice, once in each direction. This property is called separability, and both binomial and Gaussian kernels are separable.

In terms of complexity, a bidimensional convolution grows quadratically in time, while a unidimensional convolution grows linearly in time. For example, given a 7×7 kernel, 49 multiplication operations and 48 addition operations per pixel are needed, a total of 97 operations. On the other hand, a unidimensional kernel with a length of 7 needs only

7 multiplication operations and 6 addition operations times two directions, resulting in 14 multiplication operations and 12 addition operations per pixel, or a total of 26 operations, which makes the operation significantly less expensive.

The total amount of operations per pixel for a convolution can be expressed by Equation 10 for unidimensional kernels, and by Equation 11 for bidimensional kernels, in both cases l identifies the length of the kernel. Comparing both equations, it is easily identifiable that the bidimensional kernel will grow quadratically in time according to the length of the kernel, it incorporates complexity of $O(n^2)$. The unidimensional kernel, despite of the two passages in the image, grows linearly, with complexity in the order of $O(n)$.

$$operations_1 = (l \times 2 - 1) \times 2 = 4l - 2 \tag{10}$$

$$operations_2 = l^2 \times 2 - 1 = 2l^2 - 1 \tag{11}$$

We also take advantage of cascaded convolution properties, avoiding unnecessary larger kernels, by reusing previous levels in the pyramid to achieve higher scales.

Following the approach used in (CROWLEY; RIFF, 2003), we start by first convolving the input image with a binomial kernel of scale $\sigma = 1$. As mentioned in Section 3.1, the binomial kernel with scale of $\sigma = 1$ is the five element kernel $\frac{1}{16} \times [1\ 4\ 6\ 4\ 1]$. The result will be an image of scale $\sigma = 1$. The resulting image is stored for further calculation of the difference-of-Gaussians pyramid, but it is also used immediately after to process the second scale level.

With the first level of the scale already calculated, it is again convolved with the binomial filter of scale $\sigma = 1$, resulting in a new level with scale $\sigma = \sqrt{2}$. This resulting image is again stored and used to calculate the subsequent scale level.

The third level of the octave being processed shall have a scale $\sigma = 2$. To properly obtain this scale, recurring to the Equation 1, we can calculate that now two passes of the binomial kernel $\frac{1}{16} \times [1\ 4\ 6\ 4\ 1]$ are needed. To achieve that scale, it is necessary to convolve the second level of the current octave with this kernel and convolve its result again with the same kernel, without storing the intermediate result.

At this point, an image with scale $\sigma = 2$ is already available. As mentioned in Section 3.2, as this image already incorporates a scale equivalent to its half-dimension counterpart, it is then possible to downsample the image using the nearest neighbor method to half its dimensions with almost no loss of information. The resulting image after the nearest

neighbor downsampling is the first scale level of the subsequent octave.

The original SIFT approach calculates an entire octave and then proceeds to the next one. As our proposal is to reuse available information we can not calculate a complete octave before proceeding to the subsequent octave. For this reason, instead of calculating all levels of an octave, we will start the calculation of the subsequent octave after three levels are processed in the current one.

The information we require from the following octave depends on how many scales we require per octave, and this requirement might come from our application definition or from the size of the original image, as there is a minimum size of image for an octave to be able to still detect keypoints. If, for example, only four scale levels per octave are required, we will need the second scale level of the following octave. If five scale levels are required, we will then need the second and third levels of the subsequent octave already computed to finish the current one.

Finally, to generate the last levels of each octave, a binomial interpolation upsampling is done using some specific levels of the subsequent octave. For example, the fourth level of some octave is generated from the upsampling of the second level of the subsequent octave. If five levels are desired per octave, the third level of the subsequent octave will be upsampled with the bilinear interpolation.

Another important characteristic of our method is to use only images with dimensions of $2^N + 1$ pixels. Images with such dimensions are perfectly suitable for the process described above, as we can go up or down on the pyramid (in terms of octaves, or image dimensions) without discarding borders or taking any extra computation requirements.

Images with dimensions of $2^N + 1$ are very simple to downsample with the nearest neighbor approach and upsample with the bilinear resampling method. In this way, taking advantage of all the characteristics our proposed pyramid method already holds, we can discard even pixels when downsampling with the nearest neighbor approach and, when upsampling with the binomial resampling method, we can copy every pixel of the original image into the odd coordinates of the resulting image, and only estimate the values of even pixels.

A model demonstrating the whole process is shown in Figure 10.

The complete process, for a pyramid with five scale levels per octave, can be summarized as follows:

1. Acquire input image;

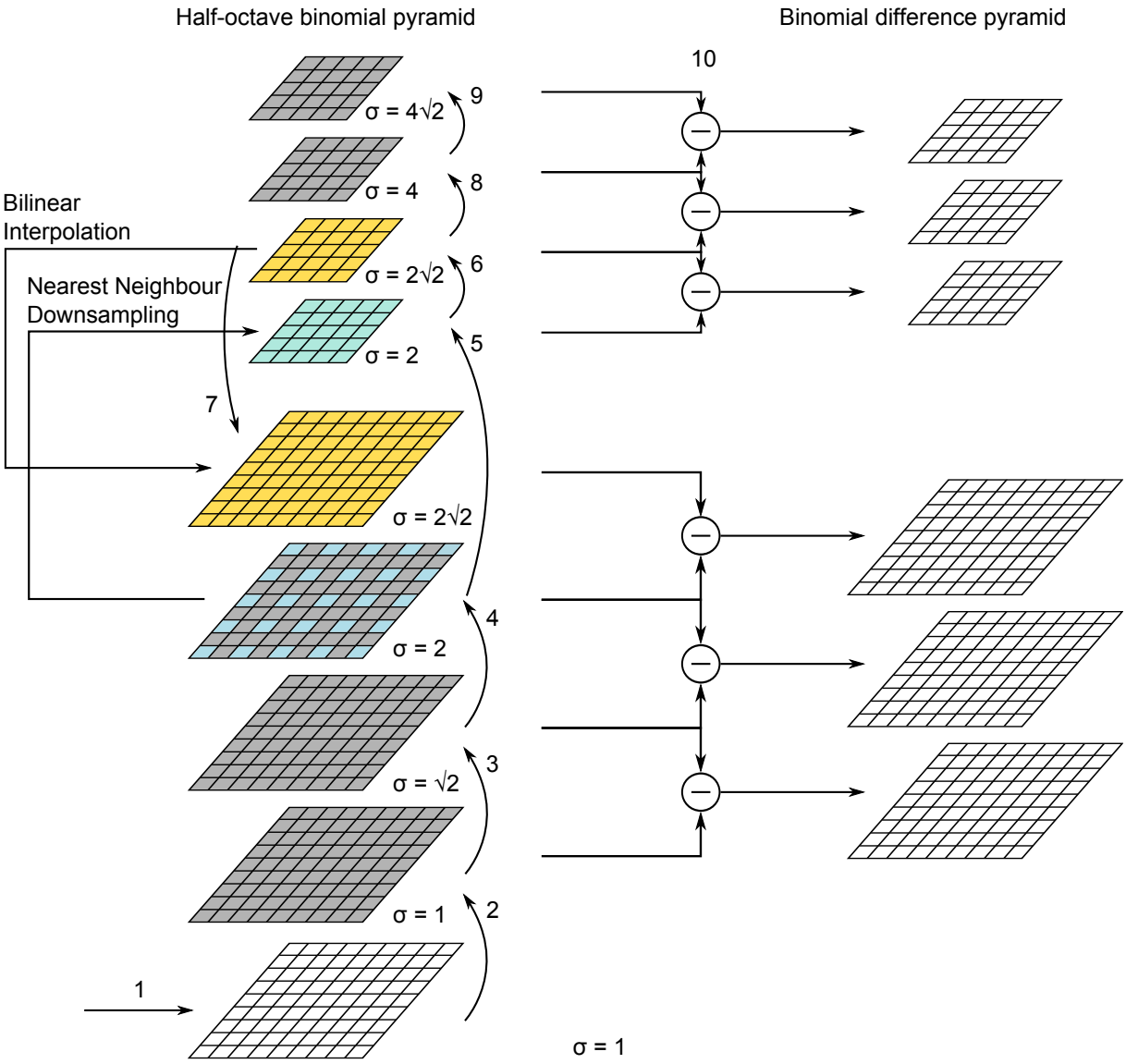

Figure 10: Model of a fast SIFT multi-scale image pyramid, this model shows the first two octaves, each with four scales. The numbers indicate the order of the steps.

2. Convolve input image with $\sigma = 1$, resulting in the first scale level of the first octave;

3. Convolve the first scale level again with $\sigma = 1$, to obtain the second scale level with scale $\sigma = \sqrt{2}$;

4. Convolve the second scale level twice with $\sigma = 1$, obtaining the third level with scale $\sigma = 2$;

5. Downsample (1:2) the third level with a nearest neighbor approach, obtaining the first level of the second octave, also with $\sigma = 2$ (relative to the input image);

6. Convolve the first scale level of the second octave with $\sigma = 1$, obtaining a scale $\sigma = 2\sqrt{2}$;

7. Upsample (2:1) the second level of the second octave with bilinear interpolation, obtaining the fourth level of the first octave, with scale $\sigma = 2\sqrt{2}$;

8. Convolve the second scale level of the second octave twice with $\sigma = 1$, obtaining the third scale level of the second octave with scale $\sigma = 4$;

9. Upsample (2:1) the third level of the second octave with bilinear interpolation, obtaining the fifth level of the first octave, with scale $\sigma = 4$;

10. Repeat steps 5 to 9, until all octaves are calculated.

4 FIXED-POINT SIMD IMPLEMENTATIONS

Flynn's taxonomy (FLYNN, 1966) classifies computer architectures based in four different branches, as follows:

- Single Instruction, Single Data stream (SISD);
- Single Instruction, Multiple Data stream (SIMD);
- Multiple Instruction, Single Data stream (MISD);
- Multiple Instruction, Multiple Data stream (MIMD);

The most common and simplest architecture, used in most personal computers, is the first one (Single Instruction, Single Data). In this architecture, each operation is executed by one instruction of the processor which processes one data element at a given time.

The second architecture, SIMD (Single Instruction, Multiple Data), is the one we are interested the most. This architecture differs from SISD regarding the fact that it processes more than one data element in a single instruction. However, the operation being executed is usually the same for all data elements. To process data in this way, a vector containing multiple data is loaded from the memory and stored in one of the processor's registers, which is then processed by a single instruction.

For image processing, the SIMD technology can be particularly useful because it is often necessary to process the same operation over a wide amount of pixels. To increase the brightness of a gray-scale image, for example, a vector containing P pixels can be loaded from memory to a register, and using one specific instruction of the processor, a certain amount, corresponding to the desired brightness increase, can be added to all the pixels contained in the vector register at the same time. This vector is then stored back to memory. If this processor is capable of processing 8 pixels (data values) at a time, we reduce the amount of operations by a factor of 8, needing only one load, one addition and one store operation, while this same process would need eight loads, eight additions and eight store operations in SISD architectures.

The first widely successful use of a SIMD architecture for desktop processors was introduced by Intel in 1997 in its Pentium family of processors, namely the MMX technology (PELEG et al., 1997). It was mainly intended for multimedia processing, like image, audio and video encoding and decoding, and still today, multimedia applications greatly benefit from SIMD instructions.

The latter two architectures, MISD and MIMD, are normally used in more complex systems, and are not normally available for personal use or embedded systems. In this work we will not cover these two architectures.

4.1 Q FORMAT

Fixed-point numbers are normally only used to represent integer numbers, but we need a way to represent fractional numbers as fixed-point real numbers. The representation of fractional numbers as fixed-point variables is not a direct operation in most processors, including ARM processors, as floating-point is normally used instead. For this purpose we use a format normally referred to as the Q format, which can be used to represent the integer part and the fractional part of the numbers according to the application. We will describe here a number in the $Q_{m.n}$ format, where m denotes the number of bits for the integer part and n denotes the number of bits for the fractional part.

When representing integer numbers in binary, every bit represents a power of 2, the least significant bit representing 2^0 up to the total number of bits N, representing a value of 2^{N-1}. With the Q format, the integer part is represented in the same manner, but we add a fractional part, in which bits are represented by negative powers of 2.

For our implementations in this work, we use 6 bits to represent the fractional part of the numbers, leaving 9 or 10 bits for the integer part, depending on the use of sign on the most significant bit. Figure 11 shows a 16-bit number with its respective powers of 2 for each bit in a $Q_{10.6}$ format.

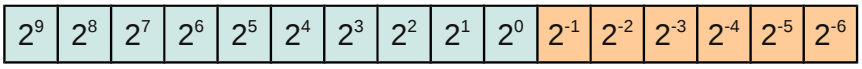

Figure 11: Q-Format representation of a 16-bit number with 10-bits for the integer part and 6-bits for the fractional part.

4.2 ARM NEON EXTENSION

With the great success of ARM processors for embedded multimedia systems, its own SIMD technology was introduced as an extension, namely Neon (REDDY, 2008). The Neon extension introduces 32 new vector registers, 64-bits wide each. These registers can also be used in dual-view mode as 16 registers of 128-bits. Each vector register can be loaded with elements of 8, 16, 32 or 64 bits, provided that all elements in a single register are of the same type (REDDY, 2008). In this work we refer a single data element in a vector register as a lane.

The instructions introduced by the Neon technology operate in all lanes of a register at once, executing the same operation for all elements in the vector register. Some operations allow widening, like a shift-left of an eight-element vector of 8-bits elements, resulting in an eight-element vector of 16-bit elements. Narrow operations are also available as, for example, shift-right operations.

In this section we introduce methods to perform the three main tasks of our efficient approach for construction of multi-scale image pyramids, using the ARM Neon extension and fixed-point notation. The three tasks are binomial filtering, nearest neighbor downsampling and bilinear interpolation upsampling. All three tasks were tested and have shown better performance than the straightforward implementations, in terms of processing time.

This section proposes the implementation of the method presented in Chapter 3 in a vectorized form using only fixed-point variables. The usage of only fixed-point variables helps by reducing the amount of data transfer from the processor to memory an enabling the usage of the ARM Neon extension. By using a vectorized implementation, multiple data is processed by a single processor instruction, thus, reducing the amount of necessary clock cycles to calculate the pyramid.

4.2.1 BINOMIAL FILTERING

In Section 3.1 we discussed how we implement binomial filters to take advantage of their properties of simpler computation and avoidance of floating-point operations. Here we explain how a binomial filtering for the Neon extension is implemented, including the differences for horizontal and vertical steps of the convolution.

Using the Q format, with 10 integer bits and 6 fractional bits, as shown in Figure 11, we can represent both binomial kernels described in Section 3.1. The first element of the binomial kernel with $\sigma = 1$, which is $\frac{1}{16}$, can be represented as a power of 2, or 2^{-4}, and is represented

in the format $Q_{10.6}$ by setting the third least significant bit to 1.

Once the value in the Q format is properly calculated in binary, it can be multiplied by an integer-only number. The resulting value of a multiplication of two numbers, one in integer format and the other in the $Q_{m.n}$ format, will result in a number expressed in the $Q_{m.n}$ format. To represent the value back in integer-only format, the resulting value of the multiplication should be shift-right by the number of fractional bits n.

When a number in Q format is converted back to integer-only format, a loss of precision occurs (truncation), in the same way it happens when converting from a floating-point to an integer-only number. When such a conversion is necessary, most situations require rounding of the value prior to conversion. In our case, we transform the values back to integers to reduce the data overhead. The conversion from the Q format to integer-only representation can be calculated by adding 2^{n-1} to the Q-format value prior to the shift-right operation.

The first part consists of the horizontal convolution, and we define the kernel length by the variable L. The step-by-step process can be described as follows:

1. Load L 128-bit vectors, $K[1..L]$, each containing one of the kernel elements in all lanes;
2. Load L 128-bit vectors, $P[1..L]$, each containing eight pixels of the current row, sliding the window one pixel to the right according to the starting pixel of the previous vector;
3. Multiply $K[1..L]$ and $P[1..L]$, saving the results in $R[1..L]$;
4. Add all vectors $R[1..L]$ and store the result in $R[1]$;
5. Add the rounding factor vector to all lanes of $R[1]$;
6. Shift all lanes of $R[1]$ to the right by n bits;
7. Store resulting vector $R[1]$ in memory.
8. Repeat steps 2 to 7 for all rows in the image.

Figure 12 shows a graphical representation of the horizontal convolution process using the Neon extension. In the example, the length is $L = 3$, being able to represent the three-element binomial filter $\frac{1}{4} \times [1\ 2\ 1]$, with $\sigma = \frac{1}{\sqrt{2}}$. In Figure 12, we assume that the first coefficient $C1 = \frac{1}{4}$ and the second coefficient $C2 = \frac{1}{2}$, and can be represented in the $Q_{10.6}$ format as the 16-bit decimal numbers with values 16 and 32, respectively.

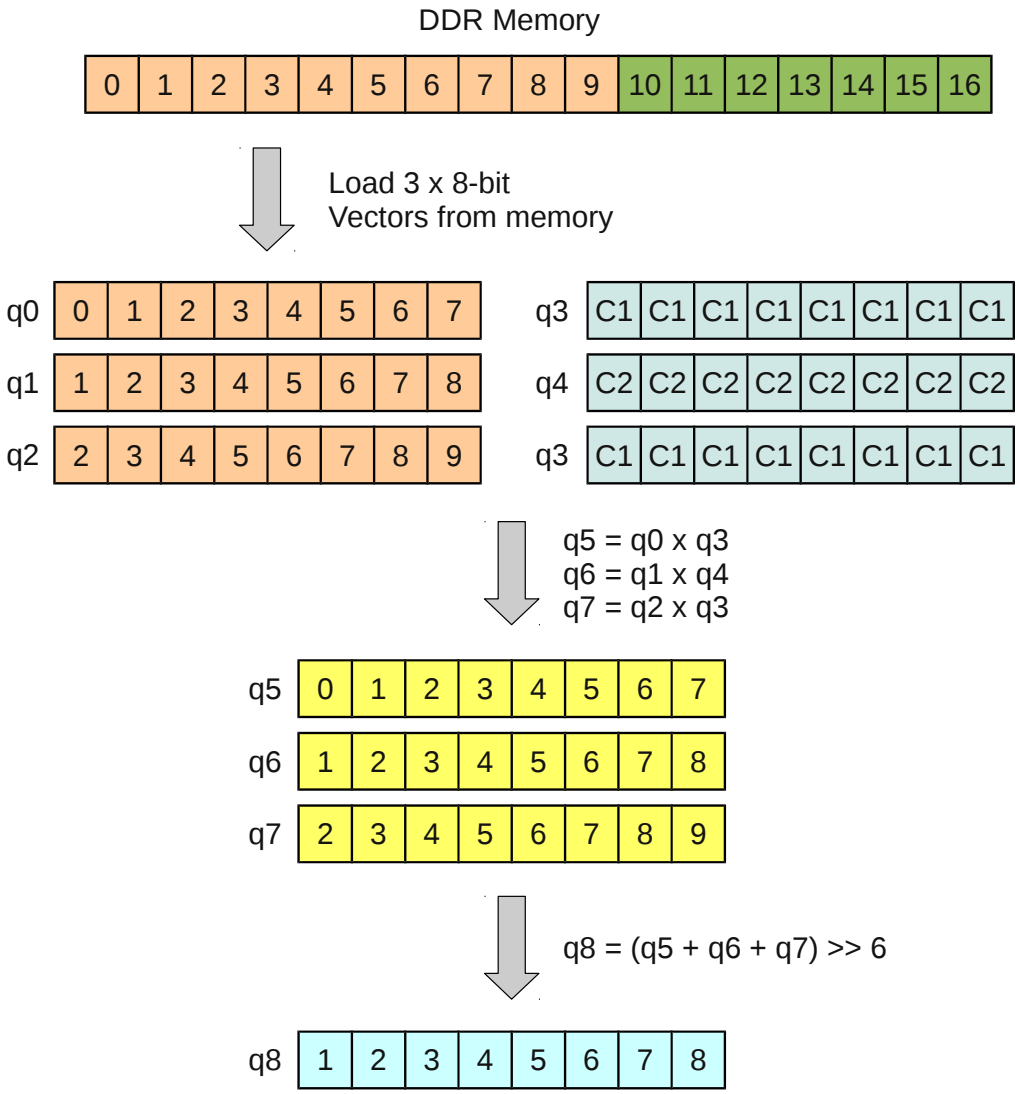

Figure 12: Horizontal convolution on the ARM Neon extension with a kernel of length $L = 3$. The numbers inside the cells represent the indices of pixels from the input image. The kernel has two coefficients, $C1$ and $C2$, $C1$ is used for both bordering elements and $C2$ is used for the central element. All vectors are 128-bits wide, with eight lanes of 16-bits each. The borders of the image (first and last pixels of each row) cannot be processed in this way, so a straightforward implementation is done to cover only these borders.

The representation of horizontal convolution in Figure 12 was meant to be simplistic for easy understanding. However, we do not use convolutions with three-element kernels, but we use the five-element binomial filter with $\sigma = 1$ instead. The representation is still valid and its implementation is done in the same way, by loading two more vectors from memory and one more coefficient value is loaded to another 128-bit register. The rest of the process is the same, with the final addition including two more 128-bit vectors.

As the borders of the input image can not be processed by the approach shown in Figure 12, a straightforward implementation is done just to take care of the border pixels, which

are the first two and last two pixels of each row in the five-element kernel case. The non-existing values are then set to the same value as the border pixel to perform the convolution.

The second part consists of the vertical convolution, which is performed over the resulting image obtained from the horizontal convolution, where L still identifies the kernel length. For this process, a different approach for loading the vectors from memory should be used. Instead of sliding the window by one pixel to the right, we slide each vector one row below from the previous pixel. The process can be summarized as follows:

1. Load L 128-bit vectors, $K[1..L]$, each containing one of the kernel elements in all lanes;
2. Load L 128-bit vectors, $P[1..L]$, each containing eight pixels of a row, sliding the window one row down according to the row of the previous vector;
3. Multiply $K[1..L]$ and $P[1..L]$, saving the results in $R[1..L]$;
4. Add all vectors $R[1..L]$ and store result in $R[1]$;
5. Add the rounding factor vector to all lanes of $R[1]$;
6. Shift all lanes of $R[1]$ to the right by n bits;
7. Store resulting vector $R[1]$ in memory.
8. Repeat steps 2 to 7 for all pixels of each row in the image.

Figure 13 shows a graphical representation of vertical convolution process with the Neon extension.

4.2.2 NEAREST NEIGHBOR RESAMPLING

As discussed in Section 3.2, we use a nearest neighbor approach only for downsampling a specific scale level in each octave. The scale level we refer in each octave is the one that incorporates twice the scale of the first scale level of the current octave, which is the third level in the octave.

To perform this process taking advantage of the Neon extension of the ARM Cortex-A8 processor included in the BeagleBoard-xM, we found out that the best way was to divide it as follows:

1. Load two vectors from memory;

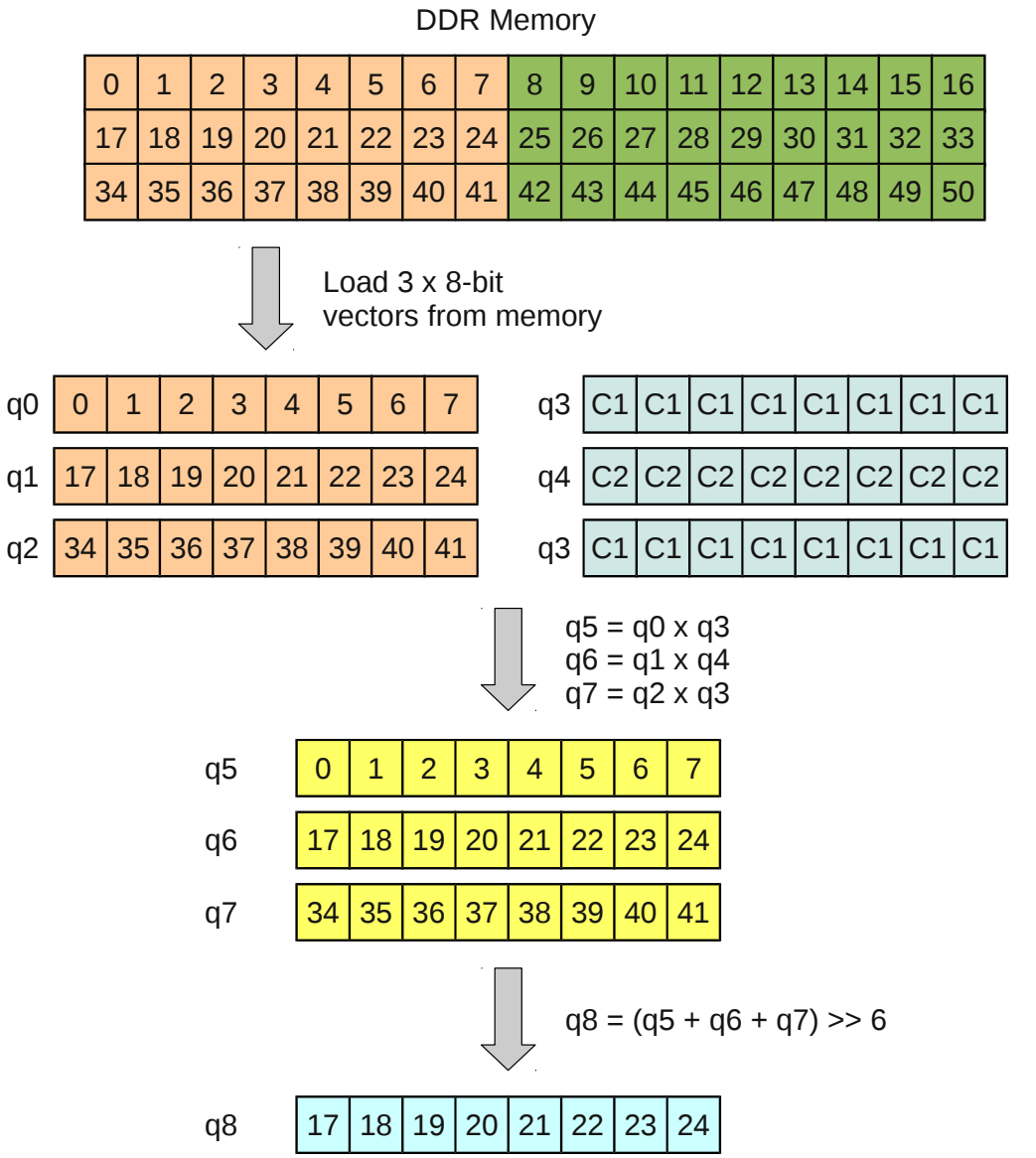

Figure 13: Vertical convolution on the ARM Neon extension with a kernel of length $L = 3$. The numbers inside the cells represent the indices of pixels from the input image. The kernel has two coefficients, $C1$ and $C2$, $C1$ is used for both bordering elements and $C2$ is used for the central element. All vectors are 128-bits wide, with eight lanes of 16-bits each. The borders of the image (pixels in the first and last rows) cannot be processed in this way, so a straightforward implementation is done to cover only these borders.

2. De-interleave pixels, using the Unzip instruction (VUZP);

3. Store the first resulting vector in memory;

4. Repeat steps 1 to 3 for the pixels in all odd rows;

In Figure 14, we provide a graphic representation of the steps described above. From the DDR memory, we load two 64-bit vectors in two registers, d1 and d0, each containing

eight pixels. As we only need the odd-pixels, we want to select only them, discarding all even-pixels. To perform this operation we use the Unzip instruction (VUZP), which deinterleaves data elements from two vectors, also resulting in two vectors, one only with the odd elements and another only with the even elements. Finally, we store in memory the register with odd-pixels, discarding the remaining pixels, and move the image pointer 16 pixels to the right.

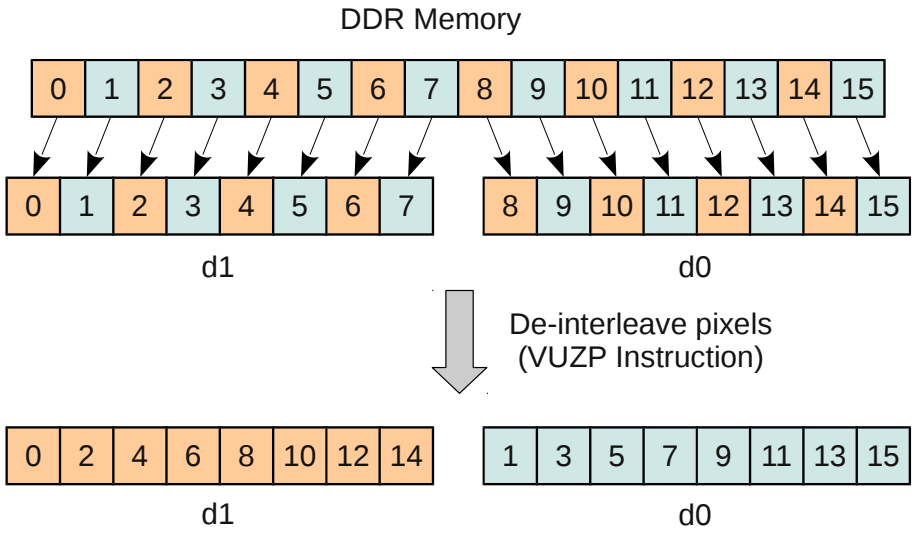

Figure 14: Nearest neighbor downsampling on the ARM Neon extension. The numbers inside the cells represent the indices of pixels from the input image. Two vectors are loaded from consecutive pixels of a row in the image, followed by a de-interleave operation of elements on both vectors with the Unzip instruction (VUZP). Only odd rows are processed, discarding even rows completely.

This process can be done by scanning the image first horizontally and then vertically, or row-by-row. In this way, we only select the odd rows, while discarding even-rows completely. When the image size does not have a number of columns that is a multiple of 16, the last few pixels are processed in a straight-forward implementation, selecting each one individually without using Neon vectors.

The process in Figure 14 can be also performed with 128-bit vectors, using registers q1 and q0, each with 16 pixels and later move the pointer 32 pixels forward instead of 16 pixels, and this is the way we actually did our implementation, but we used 64-bit vectors in Figure 14 to ease understanding and simplify representation.

4.2.3 BILINEAR RESAMPLING

We use bilinear resampling only to upsample images with dimensions of $2^N + 1$ to obtain another image with $2^{N+1} + 1$ dimensions, as explained in Section 3.3.

This is probably the most complicated process to perform using a SIMD architecture,

but because we only interpolate pixels in even rows and columns, a specific approach can be used for this particular case.

The process starts by allocating enough memory to hold an image of $2^{N+1}+1$ dimensions and copying all pixels from the original image to the odd rows and odd columns in the destination image, in other words, we copy all the pixels from the source image into the destination image, but each pair must have a pixel between them in both vertical and horizontal directions. In the horizontal direction this can be done by loading a 128-bit register with 16 pixels of the original image and another one filled with zeros. Using the Zip instruction (VZIP) (complementary to the Unzip instruction, which was used in Section 4.2.2 to perform the nearest neighbor downsampling) we interleave the pixels from both registers, resulting in two new registers, one with the first 8 pixels interleaved with 8 zeros and the second with the last 8 pixels also interleaved with 8 zeros. These two registers are then stored in the destination image. This first part of the process is shown in Figure 15.

With the destination image allocated and the original pixels already copied into the proper cells, the horizontal interpolation is first executed. The interpolation can be done by convolving only the odd rows with the kernel $\frac{1}{2} \times [1\ 2\ 1]$. This convolution can be processed exactly as the one in Figure 12 and with this kernel it has the effect of interpolating even columns by adding the halves of both horizontal adjacent elements, while leaving pixels from odd rows intact. A more efficient way to perform this particular convolution is using the vector rounding halving add instruction (VRHADD) to add the border elements together plus a rounding factor (1 in this case) and halving the addition by executing a shift-right operation of one bit, followed by an addition with the central element. By using the vector rounding halving add instruction we exempt the process of doing each operation at a time.

At this point, the horizontal interpolation is finished, but the vertical pass is still needed. The same method used for the interpolation of horizontal pixels can be used, by performing a vertical convolution with the same kernel transposed, provided that all pixels in even rows are zeroed first. However, for optimal computational power usage, we chose to perform it with a slightly simpler approach, but providing the same result. The rows that are necessary to interpolate have no assigned information yet, because we did not set the pixels to zero to save some clock cycles from the CPU and memory bandwidth. To interpolate the pixels in even rows, we load two 128-bit vectors with 16 pixels each, one with the pixels from the row above and the other with the pixels from the row below. To calculate the resulting value, the simplest approach is to divide all lanes in both registers by two (i.e., shifting all bits one position to the right), summing the results into a new 128-bit vector that is then stored in the even row of the

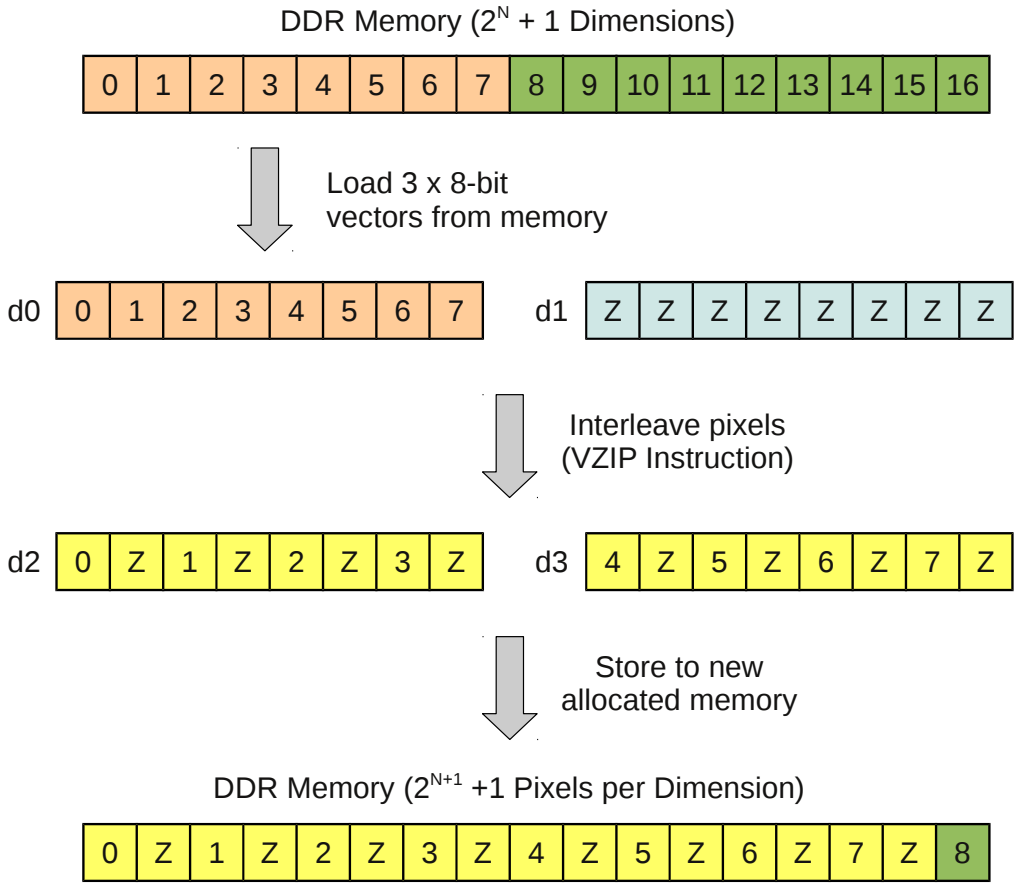

Figure 15: ARM Neon extension bilinear upsampling memory allocation and source copy. The numbers inside the cells represent the indices of pixels from the input image. Memory is allocated for a $2^{N+1}+1$ image, and one vector is loaded with eight consecutive pixels from the row of an image and another vector with eight zeros. With the interleave instruction (VZIP), both vectors are interleaved and the pixels from the source image will be distributed with a one pixel spacing, each loaded with zero (denoted by Z in the image). All rows in the original image are processed in this way and stored in the odd rows in the destination image.

destination image, or using the vector rounding halving add instruction (VRHADD) in the same way we did in the horizontal convolution, but this time without adding the central element.

The second part of the binomial resampling process is shown in Figure 16.

We can summarize the process as follows:

1. Allocate memory for an image with $2^{N+1}+1$ dimensions;
2. Load an eight-element register $R1$ with consecutive pixels of a row from memory;
3. Load an eight-element register $R2$ with zeros;
4. De-interleave $R1$ and $R2$ into $R3$ and $R4$;

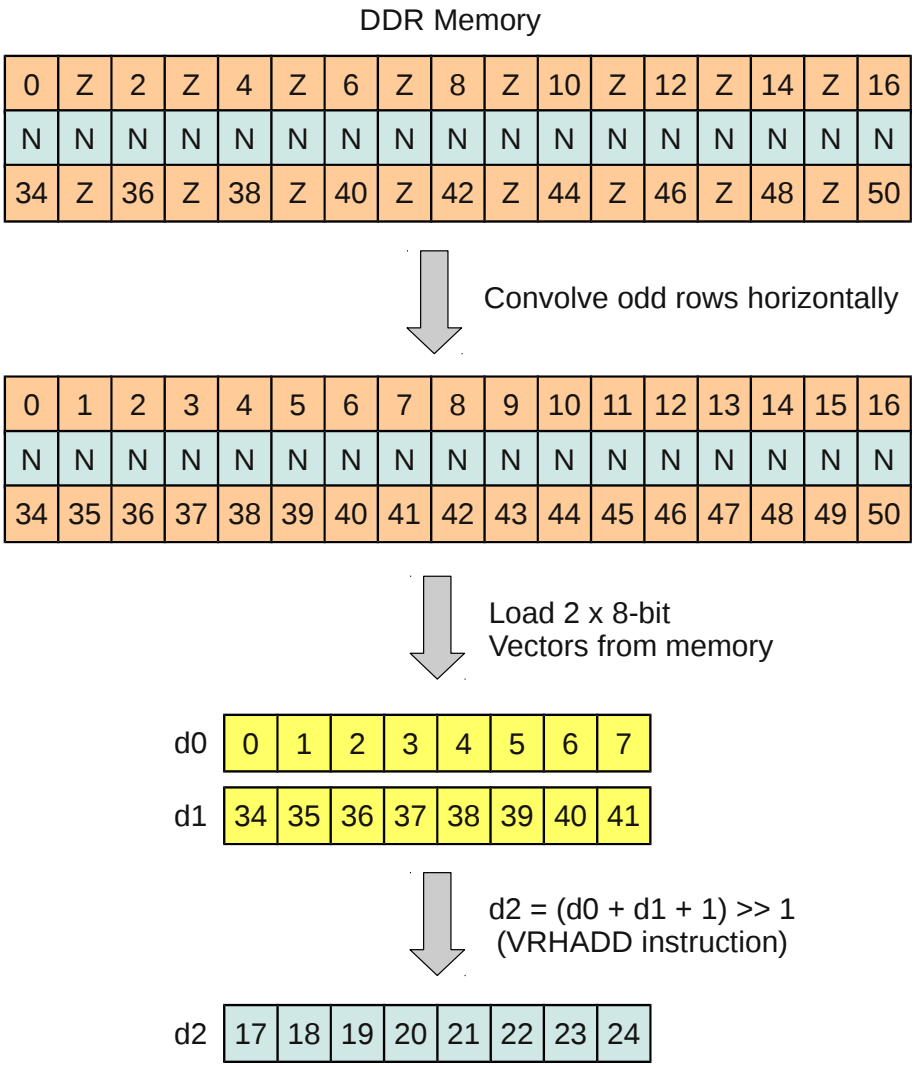

Figure 16: ARM Neon extension bilinear upsampling calculation. The numbers inside the cells represent the indices of pixels from the input image. Odd rows that are already filled are convolved horizontally in a simplified manner to change de zero values Z into the actual values (sum of halves of horizontally adjacent pixels). A second simplified operation is processed, now vertically, by summing pixels in adjacent rows and halving them to find the actual values of N, which have no assigned information yet.

5. Store $R3$ and $R4$ into odd rows in the newly allocated image;

6. If source image is properly stored into destination, follow to step 7, otherwise go back to step 2;

7. Load three vectors of eight elements, $P[1..3]$, each containing eight pixels of the current row, sliding the window one pixel to the right according to the starting pixel of the previous vector;

8. Perform a vector rounding halving add with $P[1]$ and $P[3]$, summing the result with $P[2]$, storing back to memory in the same position $P[2]$ was loaded from;

9. If all odd rows were processed, follow to step 10, otherwise go back to step 7;

10. Load two vectors of eight elements, $P[1..2]$, each containing eight pixels of the row above and below of an even row;

11. Perform a vector rounding halving add with $P[1]$ and $P[2]$, storing the result in memory to the position of the even row;

12. If all even rows were processed, finish, otherwise go back to step 10;

5 RESULTS

In this chapter we evaluate the performance of the implementation method proposed in this work. In the first section we evaluate its execution time performance in the BeagleBoard-xM, comparing a vectorized fixed-point implementation of our method to a straightforward floating-point implementation and to an implementation of the original SIFT method. The second section is responsible to evaluate the repeatability performance of the proposed detector, evaluating the repeatability of the two implementations of our method to the original SIFT method, and their repeatability against themselves while varying the rotation in the input images.

5.1 PROCESSING TIME

In this section we evaluate the processing time of our proposed method for detection of interest points, with a straightforward floating-point implementation, a fixed-point vectorized implementation on a SIMD architecture, and the original SIFT method in a straightforward floating-point implementation.

5.1.1 EXPERIMENTAL SETUP

To ensure that a fair comparison from one method to another, all tests were executed using the same software and hardware configurations, and with the same image samples. The only change is the actual code implementation of each method, and all of them are written in C++.

First of all, the BeagleBoard-xM is running at a frequency of 600 MHz for all cases. All software used is free and open source, mainly composed by the Linux Kernel version 3.3.7, G++ Compiler version 4.5.3 provided in the BeagleBoard SDK version 5.7.0 from Texas Instruments, and the OpenCV library (BRADSKI; KAEHLER, 2008) version 2.4.6.

To show that the computational complexity of the image pyramid building process is

directly related to the resolution of the input image, we use a set composed of nine images with $2^N + 1$ dimensions. The content of these images is not important, as the number of operations is strictly proportional to the number of pixels in the image, for this reason, blank images were used.

5.1.2 EXPERIMENTAL RESULTS

Following the setup described in Section 5.1.1, for each of the three methods and each image of the dataset, the pyramid is constructed 220 times. We then select the 200 innermost values (discarding the first and last 10 values) to ensure that all trials are valid, thus, discarding, for example, the possibility of getting some data during the transition from the idle state of the processor. Some of the results we present in this section are already available in (ENTSCHEV; VIEIRA NETO, 2013), but with slightly different results due to different version of the tools used.

We present the data in multiple forms, enabling the reader to visualize the contents of the data achieved in different ways. We first start presenting the results in Table 1, which shows the average execution time of the selected 200 tests for each method and input image resolution. From this information, it is possible to see that the execution time grows as a function of the image resolution, independently of the pyramid construction method used. By comparing fourth and fifth columns, dividing the average values, we see that the straightforward floating-point binomial pyramid takes about $\frac{1}{4}$ of the time necessary to build a Gaussian pyramid. When the third and fourth columns are compared, we can easily find that the processing time is reduced to around $\frac{1}{10}$ of the straightforward floating-point binomial pyramid construction time. Finally, comparing the third and fifth columns, the processing time taken is reduced to about $\frac{1}{40}$, or less than 3% of the original SIFT pyramid construction time.

Figure 17 shows the data from Table 1 as a graph of processing time as a function of the input image resolution. In this graph it is possible to see the large difference between the processing times for each method.

Lastly, Figure 18 shows the data from Table 1 in a logarithmic scale for processing time. This graph clearly shows that the complexity remains quadratic, but the amount of necessary operations is reduced from the Gaussian pyramid to the binomial pyramid and then to the vectorized fixed-point pyramid by some orders of magnitude. We can see that the processing time grows logarithmically due to the linear resolution increase, and is possible to see the processing time reduction to $\frac{1}{4}$ from the Gaussian approach compared to the straightforward floating-point binomial method, the $\frac{1}{10}$ ratio from the straightforward

Table 1: Average execution time and standard deviation for the construction of binomial and Gaussian pyramids on the BeagleBoard-xM. The construction of the straightforward floating-point binomial pyramid takes about $\frac{1}{4}$ of the time to process Gaussian pyramid, and the vectorized fixed-point binomial pyramid takes about $\frac{1}{40}$ of the time to process the Gaussian pyramid.

Input image size (pixels)	Number of octaves	Execution time (s)		
		Neon pyramid	Binomial pyramid	Gaussian pyramid
129×65	4	0.0019 ± 0.0002	0.0186 ± 0.0003	0.0771 ± 0.0112
129×129	5	0.0035 ± 0.0003	0.0366 ± 0.0030	0.1517 ± 0.0049
257×129	5	0.0062 ± 0.0002	0.0691 ± 0.0017	0.2954 ± 0.0068
257×257	6	0.0121 ± 0.0016	0.1416 ± 0.0050	0.5851 ± 0.0087
513×257	6	0.0225 ± 0.0004	0.2776 ± 0.0061	1.1601 ± 0.0139
513×513	7	0.0459 ± 0.0016	0.5561 ± 0.0100	2.3098 ± 0.0213
1025×513	7	0.0942 ± 0.0043	1.0971 ± 0.0144	4.6390 ± 0.0324
1025×1025	8	0.2126 ± 0.0049	2.2104 ± 0.0207	9.2559 ± 0.0424
2049×1025	8	0.4042 ± 0.0108	4.3565 ± 0.0315	18.6311 ± 0.0478

floating-point implementation to the vectorized fixed-point approach, and $\frac{1}{40}$ from the Gaussian approach to the Neon implementation.

An important thing to notice is that the vectorization is not the only responsible to reduce the processing time to a ratio of $\frac{1}{10}$, but also the avoidance of floating-point and consequent reduction on data size. In the ARM architecture, floating-point is implemented as 32-bit numbers and we processed most of our data as fixed-point 8-bit numbers (but also 16-bit numbers for the convolution processes). This change by itself is responsible for a reduction to $\frac{1}{4}$ in the bandwidth, and a significant change in the number of clock cycles taken for each instruction. In the vectorization process, if we were able to process floating-point numbers, we could only process two or four elements at a time, instead of eight or sixteen elements as we did, which consumes significantly more processing power.

5.2 KEYPOINT REPEATABILITY

In this section we provide results for the repeatability of keypoints of the method proposed in this work for efficient SIFT multi-scale image pyramid generation, comparing it to the results achieved with the original SIFT method.

Keypoint repeatability can be described as the amount of points of interest that are found in a reference image containing a particular object or scene that are later detected in a test image, with this test image containing the same object or scene subject to some geometrical transformation or by changing the detection method.

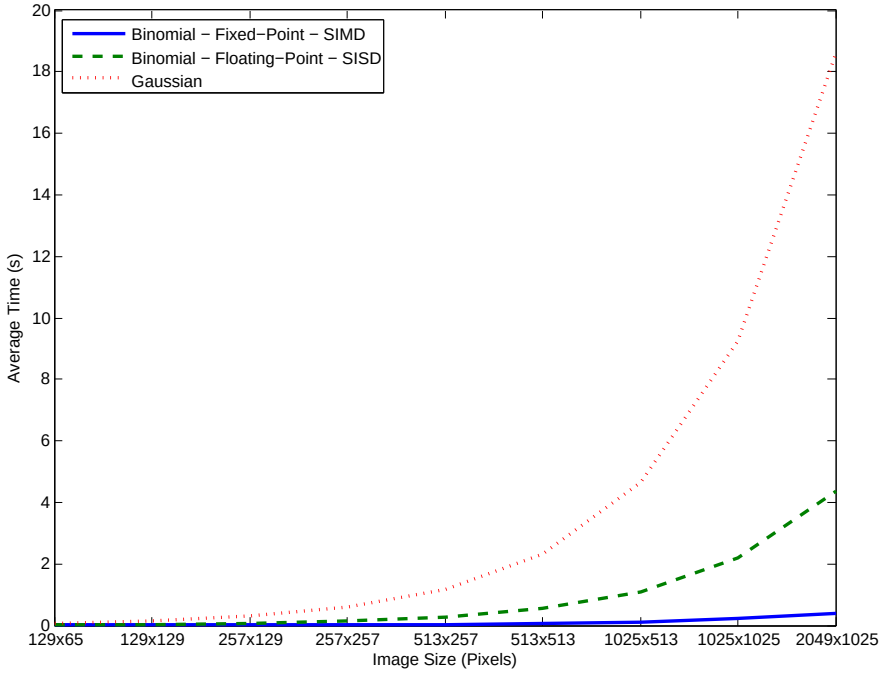

Figure 17: Processing time of the Gaussian pyramid (red dotted line), straightforward floating-point binomial pyramid (green dashed line) and vectorized fixed-point binomial pyramid (blue solid line).

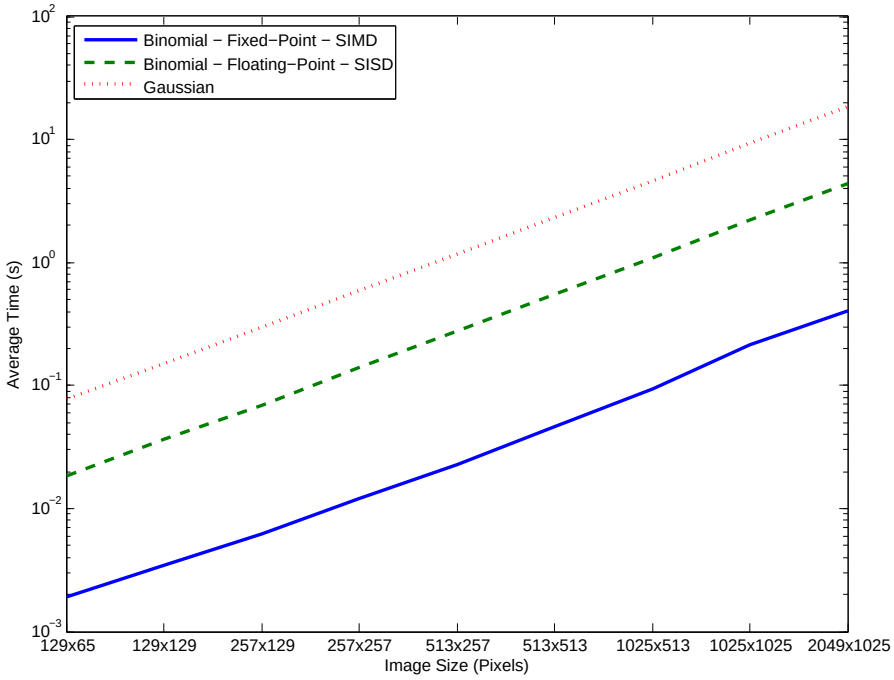

Figure 18: Processing time (logarithmic scale representation) of the Gaussian pyramid (red dotted line), straightforward floating-point binomial pyramid (green dashed line) and vectorized fixed-point binomial pyramid (blue solid line).

There are normally two types of tests done regarding keypoint repeatability when a novel method for interest point detection is formulated. The first one compares the robustness of the novel method against an existing and well-accepted method for such a task. The second one involves testing the keypoint detector against itself.

In the first type of test, testing the novel keypoint detector against another existing method, the reference and test images are normally the same, without any natural or synthetic transformations. When the comparison of keypoint detectors is done using the same images we ensure that the conditions given for both detectors are the same, without incorporating any variable that might be significantly harmful for only one of the detectors, allowing a fair comparison.

For the second type of test, the test image, or set of images, must have some transformation according to the reference image, like changes in orientation, scale, viewpoint, affine transformations or noise. In the case of geometric transformations, it is important that all the keypoints can be translated into the new geometric form, in such a way that it is possible to say that a particular keypoint in K_r in the reference image corresponds to some keypoint K_t of the object or scene in the test image.

In this section, we will first evaluate the repeatability of the keypoint detector described in Section 3.4 against the original SIFT method, as described in (LOWE, 2004). The second part will consist of the assessment of the detector as described in Section 3.4 against itself, when images are subject to rotation transformations.

5.2.1 EXPERIMENTAL SETUP

The experimental setup for keypoint repeatability data extraction is strongly based on the one presented in (ENTSCHEV; VIEIRA NETO, 2014), as the data that will be evaluated is very similar to the one in that work.

The construction of the straightforward floating-poing binomial pyramid is done following the description presented in Chapter 3, and the vectorized fixed-point binomial pyramid is based on the contents of Chapter 4. The construction of the Gaussian pyramid is done as explained in (LOWE, 2004), using the implementation from the OpenCV library (BRADSKI; KAEHLER, 2008) with some modifications to build the pyramids with the same scales as the ones achieved with the binomial pyramid. The software used and its version is the same as described in Section 5.1.1.

The number of octaves o in each pyramid is proportional to the smallest dimension of

the image. It can be calculated using Equation 12, where d is the dimension of the smallest side of the image and $\lfloor . \rfloor$ is the floor function.

$$o = \lfloor \log_2 d \rfloor - 2 \tag{12}$$

To consider a keypoint K_t from a test image as a repetition of some keypoint K_r from the reference image, the position of K_r must be estimated in the test image and the Euclidean distance of K_t to the estimated K_r position must be no larger (in pixels) than the scale s_r of the reference keypoint K_r. The scale s_t of the test keypoint K_t must also meet the condition from Equation 13.

$$\sqrt{2}s_r - s_r \leq s_t \leq \sqrt{2}s_r + sr \tag{13}$$

The nature of the scale space might cause keypoints to shift slightly, for this reason we can not consider the exact estimated position of a keypoint, making it necessary to consider a small surrounding area to validate the keypoint. For different octaves, the keypoints must be estimated according to the input (original) image, and this might vary from one pyramid construction method to another. To identify the keypoint location more precisely we follow the method proposed in (BROWN; LOWE, 2002), which is also used in the original SIFT approach, fitting a quadratic 3D function to the scale space to interpolate the location of the keypoint.

When the images are subject to rotation we discard keypoints lying outside a circle with radius equal to the smaller dimension of the image, because in our synthetic method for rotation the area outside that circle might be occluded, thus, the keypoint will not be detected because of there is no image data where it is located.

The dataset we use to perform the repeatability tests is composed of 12 images, taken from the Affine Covariant Regions Dataset and the original dataset of SIFT images, provided by the Visual Geometry Group of the University of Oxford (Visual Geometry Group, 2004) and by the SIFT demo program from David Lowe (LOWE, 2005), respectively. The images were not used in their original formats, but were cropped to meet the $2^N + 1$ dimension requirements of the proposed method, for both vertical and horizontal directions.

5.2.2 REPEATABILITY COMPARED TO SIFT

In this first experiment we test our method against the original SIFT approach. Both straightforward floating-point and vectorized fixed-point implementations are tested and

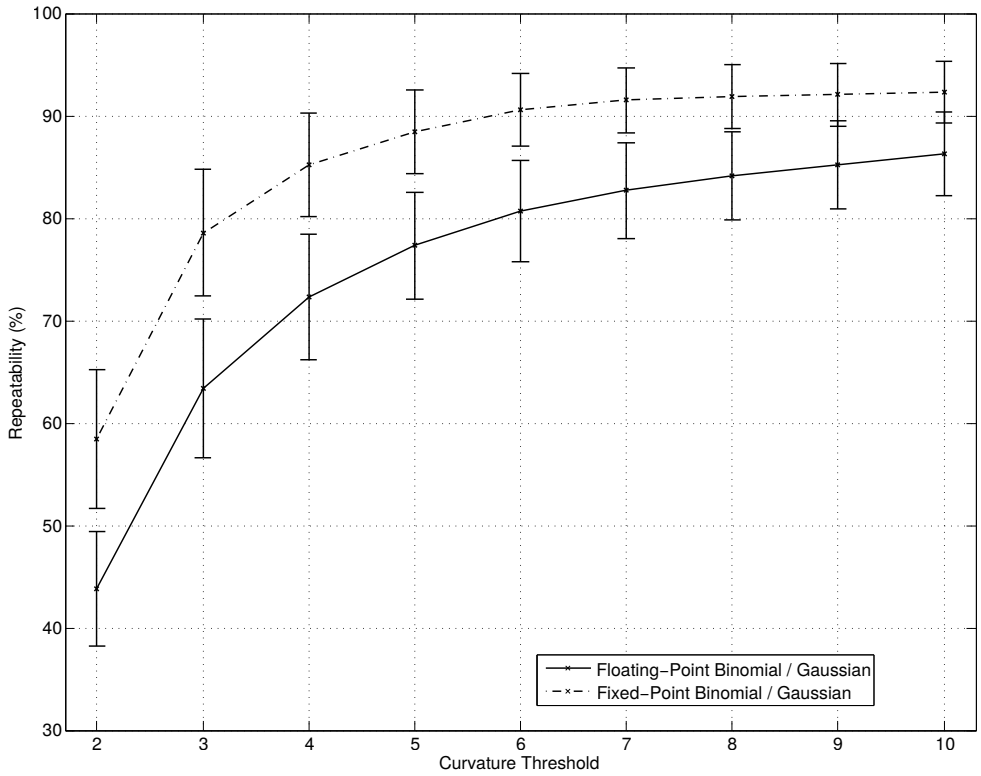

Figure 19: Average amount of keypoints found using a Gaussian pyramid repeated in the straightforward floating-point binomial pyramid (solid line) and in the vectorized fixed-point binomial pyramid (dashed line), both as functions of the curvature threshold used in the binomial pyramid. Vertical bars represent the standard deviations. As curvature threshold is increased, the total amount of found keypoints is increased together with the repeatability ratio.

compared to the SIFT detector.

The solid line in Figure 19 shows the average amount of keypoints found using a Gaussian pyramid repeated in the binomial pyramid built with a straightforward floating-point implementation. The dashed line shows the same average amount of keypoints, but with the binomial pyramid constructed with a vectorized fixed-point implementation. The graphs show that the floating-point pyramid achieves in average around 86% of repeatability with a curvature threshold of 10 and around 78% with a curvature threshold of 5. The fixed-point pyramid in average achieves up to 92% of repeatability at a curvature threshold of 10 and around 88% for a curvature threshold of 5.

Even though the fixed-point binomial pyramid has a higher repeatability rate than the floating-point binomial pyramid, both compared to the Gaussian pyramid, there is a downturn in this type of implementation. As can be interpreted from the dashed line in Figure 20, the total amount of keypoints found is much higher than the amount found with the floating-point implementation of the binomial pyramid, the solid line. Experimentally, we found that this is the result of multiple rounding of values during the pyramid computation. Such rounding

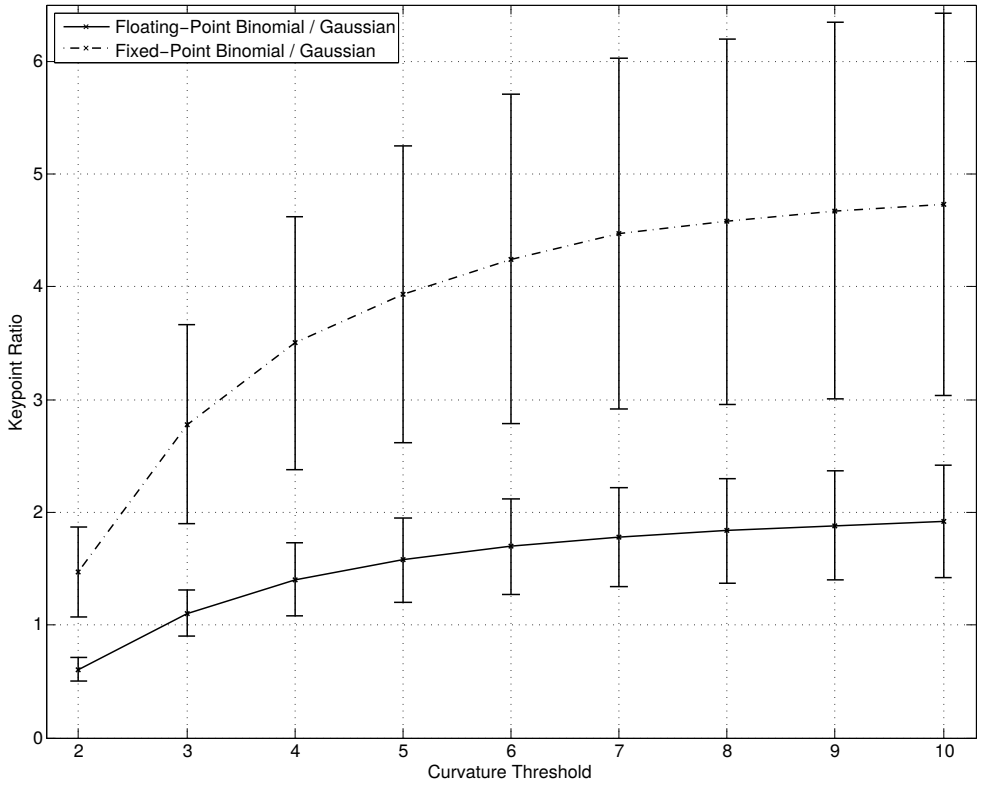

Figure 20: Ratio of keypoints found in a straightforward floating-point binomial pyramid (solid line) and in a vectorized fixed-point binomial pyramid (dashed line), both compared to a Gaussian pyramid as a function of the curvature threshold used in the binomial pyramid. Vertical bars represent the standard deviations. As curvature threshold is raised, the total amount of found keypoints is increased, consequently increasing the ratio.

operations are not performed in the floating-point version, as we always store the values with the fractional parts, and use the numbers with fractional parts to compute the next steps of the pyramid.

Figure 24 shows a sample image containing the detected keypoints with a Gaussian pyramid with a curvature threshold of 10 on the left, and the floating-point binomial detected keypoints with a curvature threshold of 5 on the right.

5.2.3 REPEATABILITY REGARDING ROTATIONS

In this experiment we test the repeatability of keypoints for both methods when images are subject to synthetic rotations. We first identify the keypoints with the straightforward floating-point binomial pyramid for all images in the dataset without rotation, using a curvature threshold of 5, then synthetically rotate each image from 5 to 355 degrees, in steps of 5 degrees, and compare the keypoints found in each resulting image to the keypoints found in the original image. We repeat the process for the vectorized fixed-point binomial pyramid. We also compare

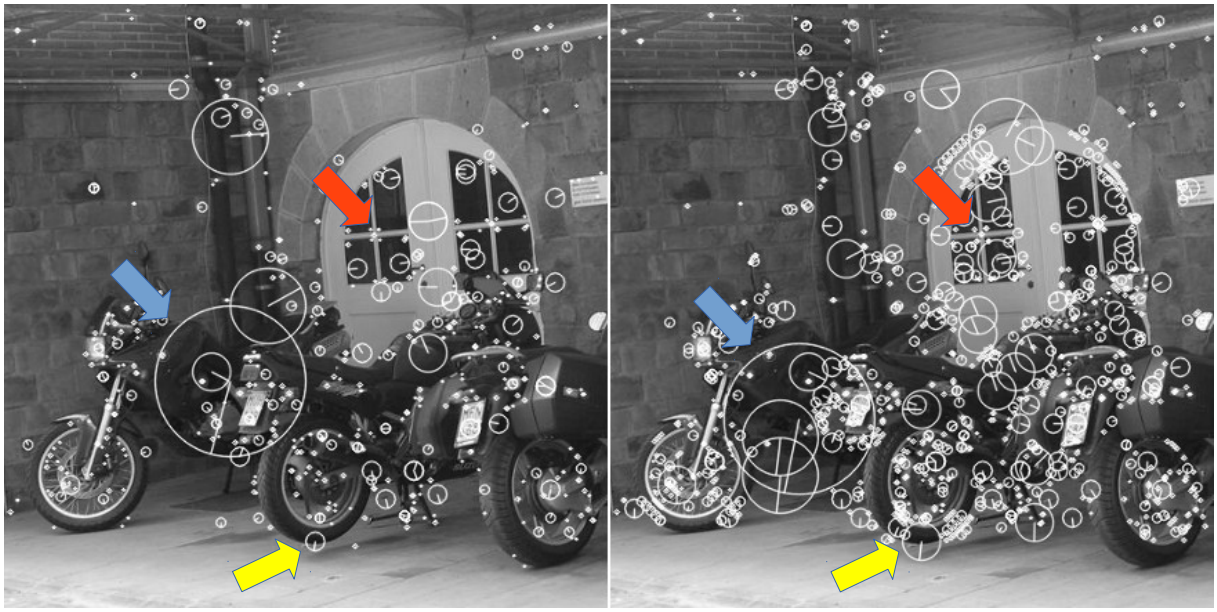

Figure 21: Detected keypoints on a sample image with a Gaussian pyramid and curvature threshold of 10 (left) and with a floating-point binomial pyramid and curvature threshold of 5 (right). The blue arrow points a keypoint that exists on both cases but shifted in space, the yellow arrow shows a keypoint that is detected with a different scale and the red arrow indicates a repeated set of keypoints.

to the original SIFT pyramid, but this with a curvature threshold of 10, the threshold suggested in (LOWE, 2004).

Figure 22 shows the average amount of keypoints that are repeated with the straightforward floating-point binomial pyramid implementation as the gray solid line. The dashed black line shows the average amount of keypoints that are repeated with the original SIFT pyramid. In this graph we can see that the repeatability over rotation of our method is slightly superior than the original SIFT, except for multiples of 90 degrees. The repeatability of the proposed method achieves up to 90% on multiples of 90 degrees and is minimum on odd multiples of 45 degrees, yielding about 85%.

As in Figure 22, the dashed black line of Figure 23 shows the average amount of keypoints that are repeated with the original SIFT pyramid (exactly the same data as in the previous graph). The gray solid line in this graph shows the repeatability of keypoints when images are subject to rotation for the vectorized fixed-point binomial pyramid. Interestingly, this implementation has a higher repeatability ratio than the straightforward floating-point implementation. The average of repeatability is almost stable at about 93%, but achieving a slightly higher ratio on 180 degrees rotation of about 96%.

Figure 24 shows a sample image containing the detected keypoints with a floating-point binomial pyramid with 0 degrees rotation on the left, 45 degrees rotation on the center

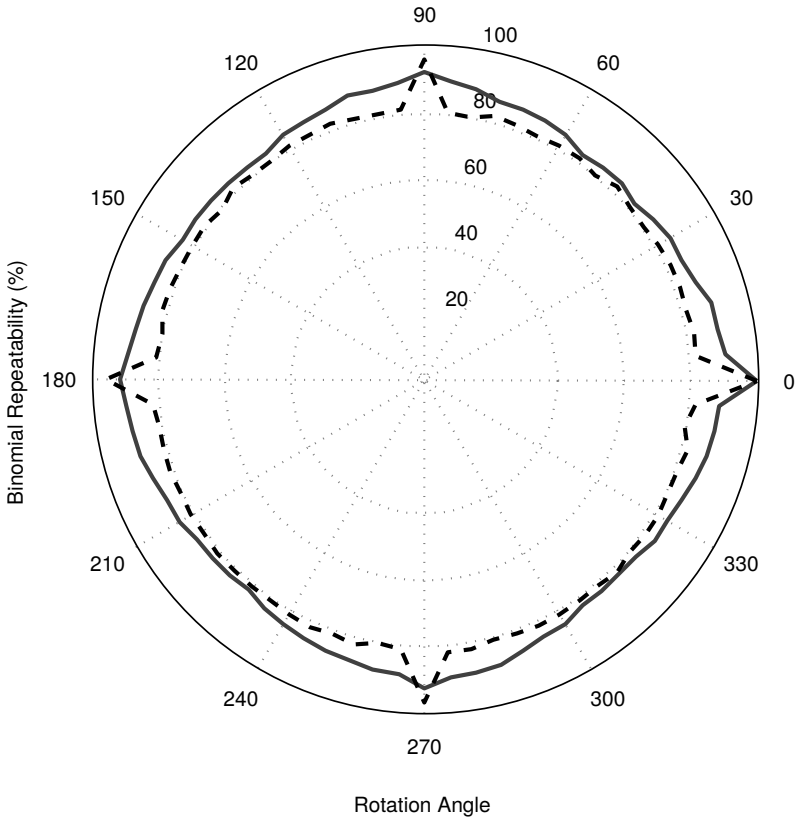

Figure 22: Rotation repeatability of straightforward floating-point binomial pyramid (gray solid line) and Gaussian pyramid (black dashed line), the external solid line indicates 100%.

and 90 degrees rotation on the right.

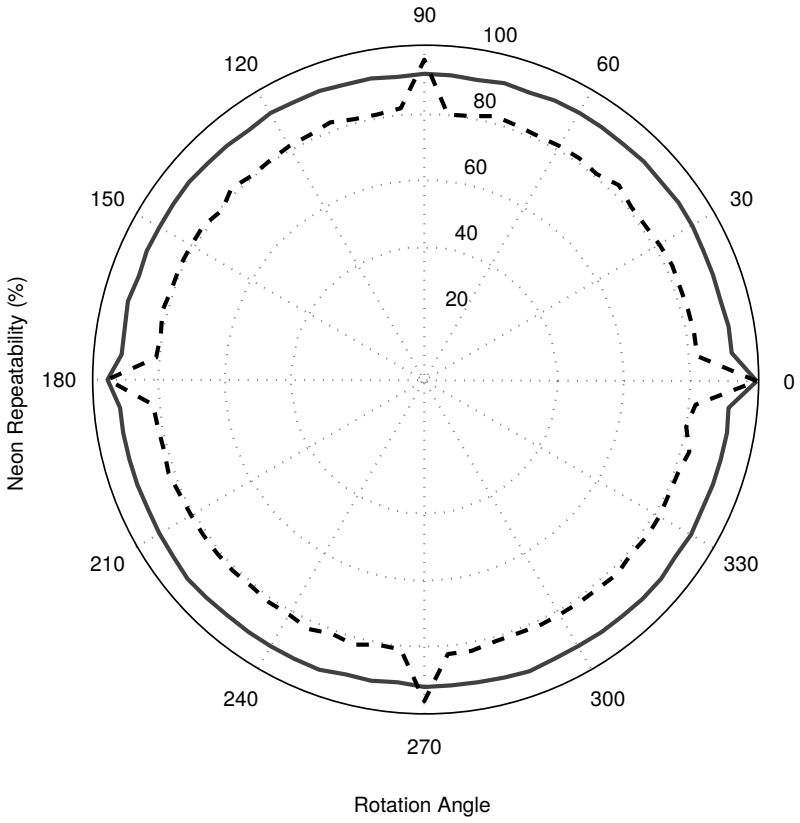

Figure 23: Rotation repeatability of vectorized fixed-point binomial pyramid (gray solid line) and Gaussian pyramid (black dashed line), the external solid line indicates 100%.

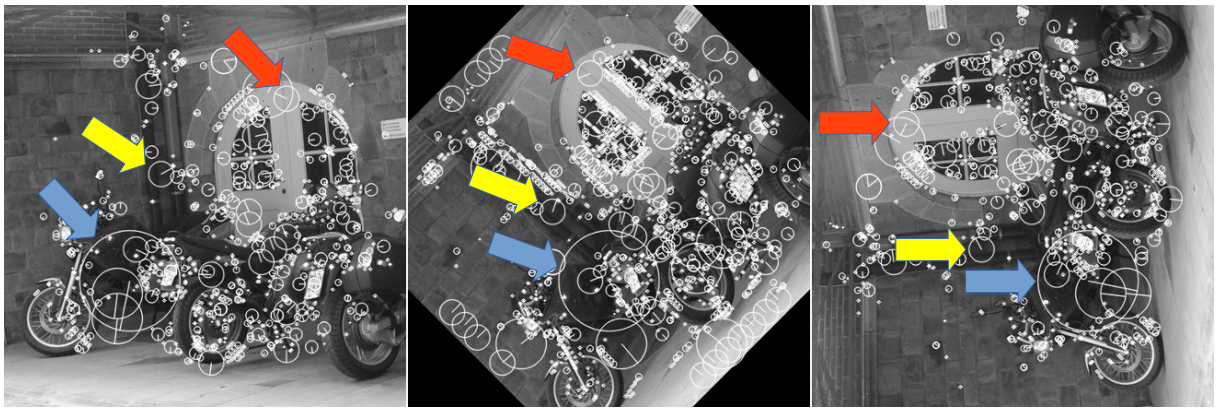

Figure 24: Detected keypoints on a sample image with a floating-point binomial pyramid and 0 degrees rotation (left), 45 degrees rotation (center) and 90 degrees rotation (right). Arrows with the same colors indicate the same keypoints detected after rotating the image.

6 CONCLUSION

In this work we introduced an efficient method to perform interest point detection on gray-scale images. The method aims at embedded implementation to be used in mobile robot vision and is able to build an entire multi-scale image pyramid in real-time. We used the BeagleBoard-xM to implement the method and provide a proof of concept of our method.

To allow the reader a complete understanding of the study, we introduced multi-scale image pyramids, while explaining their importance for object recognition to allow scale invariant interest point detection, which is often a desirable feature in an object recognition method, as objects might appear in a scene with different sizes compared to the reference object. We based our study in two methods, SIFT and SURF, that are normally used as reference in this field for the good performance both methods achieve not only in interest point detection, but also for the complete object recognition process.

We introduce a method for the efficient calculation of multi-scale image pyramids, partially based on the SIFT method. The method we propose uses binomial filters instead of the conventional Gaussian filters used in SIFT, thus significantly reducing computation time and allowing the calculation of multiple scale levels without floating-point operations. We downsample the third scale level of an octave to achieve the first scale level of a new octave of the pyramid. Instead of calculating every scale level directly by convolution with binomial or Gaussian filters as is done in SIFT, we upsample some specific levels of the next octave to achieve higher scale levels of each octave in a computationally more efficient way. A very important characteristic of our method is the use of images with $2^N + 1$ dimensions, as they are perfectly suitable to allow both upsampling and downsampling without discarding borders in any case, when being resampled to another $2^N + 1$ dimension.

As in modern computing it is not enough to only implement algorithms in a straightforward way, we explained completely how to implement our method in a vectorized manner, using a SIMD (Single Instruction, Multiple Data) architecture, which is included in our chosen embedded platform, the BeagleBoard-xM, as an extension to the ARM Cortex-A8

core, called Neon. SIMD architectures differ from traditional SISD (Single Instruction, Single Data) in their capability to process multiple data with one single instruction, but normally performing the same operation for all data elements. We give a detailed explanation on how to implement efficiently the three main processes we use to build our multi-scale pyramid, which are the convolution, in both vertical and horizontal directions, as they need significantly different implementation methods for efficient use of resources, followed by nearest neighbor downsampling and bilinear upsampling. As the Neon extension does not support floating-point operations natively, all our implementation is completely fixed-point based.

We then proceed to the results our method achieves, separating them in processing time and interest point detection performance evaluations. Without any special kind of implementation, our method takes only $\frac{1}{4}$ of the time taken to build a Gaussian pyramid in the SIFT, which is already a very good reduction in processing time. The vectorized fixed-point implementation of our method has an even more significant reduction on computation time, taking $\frac{1}{10}$ of the straightforward floating-point implementation of our method, or $\frac{1}{40}$ of the original SIFT method. The processing time reduction achieved is not only because of the SIMD implementation by itself, but also because of the data size – the ARM Cortex-A8 processor represent floating-points as 32-bit numbers and we normally used only 8-bit fixed-point numbers or, at most, 16-bits for the convolution process. With this implementation running at 600 MHz in the BeagleBoard-xM, we can achieve real-time performance with image resolutions of 513x257 pixels at 44 frames per second, and around 21 frames per second with 513x513 pixel images. The BeagleBoard-xM processor still supports 800 MHz and 1 GHz clock frequency, which are easily interchangeable in the Linux operating system by properly configuring the kernel, allowing even higher frame rates.

We compared both the straightforward floating-point and vectorized fixed-point implementations to the original SIFT implementation to evaluate the repeatability of the detector, and found that we achieve up to 86% in the first method and up to 92% in the vectorized fixed-point implementation, a result that shows that most expected keypoints are found. However, we find more keypoints than the Gaussian approach, up to approximately 1.8 times in average for the floating-point implementation and 4.7 times for the fixed-point implementation, which means that they have stronger edge responses than the original SIFT method. This effect can be compensated by lowering the curvature threshold when detecting keypoints.

The rotation repeatability outperforms the original SIFT approach when rotating images for angles which are not multiples of 90 degrees. The original SIFT method has a

repeatability of about 95% for multiples of 90 degrees, otherwise it remains stable at 80%. The floating-point implementation of the proposed method has a maximum repeatability rate at multiples of 90 degrees, approximately 90%, and achieves its minimum for odd multiples of 45 degrees, around 85%. The repeatability on rotation variations for the fixed-point implementation is almost stable at all rotation angles, with a repeatability of 93%, achieving about 96% only for 180 degrees of rotation.

The main disadvantage of the present method lies in the detection of significantly more points of interest, most of them relying on low-curvature corners that are not sufficiently stable. A higher amount of keypoints implies more descriptors being calculated, increasing the overall computation time for a given image.

During our work, we performed some tests with the DSP core available in the BeagleBoard-xM, but implementation complexity restrained further code development for this architecture in the time frame we had available. According to the literature (KARADAYI et al., 2003), the bandwidth currently available for memory to DSP transfers is the main bottleneck in performance. In our tests, the memory bandwidth seemed to be a limiting factor for image processing, but further studies are required to verify that this is the only bottleneck or others exist.

6.1 FUTURE WORK

Experimentally, we identified that the process of image upsampling to the last levels of each octave in the pyramid was the most inaccurate part in the whole process, as it slightly shifts pixels, enabling the identification of lower curvature corners as extrema, thus leading to detection of some undesired edges as interest points. This particular step needs careful evaluation in order to identify interest points more coherently, discarding keypoints that do not hold high enough curvatures. To reduce edge responses in our method, it might also be interesting to evaluate a complementary approach to the method used in SIFT to discard edge responses, which tests for the ratio between the largest magnitude eigenvalue and the smaller one, removing keypoints with the smallest ratios.

More tests regarding the robustness of interest point detection are desirable, to allow the evaluation of performance of the method against noise, affine transformations, scale changes, among other geometric transformations. Even though we evaluated the performance of our detection method against the original SIFT method and itself on rotation variations, these are not enough to evaluate all its properties. Tests including the recognition rate of the detected

keypoints with the original SIFT feature description calculation and matching steps, or some other feature description and matching technique, are also important to be further investigated.

Proper studies of DSP implementations are also required for complete evaluation of the proposed method in the BeagleBoard-xM. If the memory bandwidth can be used efficiently, an implementation of our method in the DSP might have even better results, perhaps using a hybrid implementation, processing some parts of the pyramid in the ARM core and the remaining parts in the DSP core.

6.2 CONFERENCE PAPERS

The following papers were published in conferences during the development of this work.

ENTSCHEV, P. A.; VIEIRA NETO, H. Efficient construction of SIFT multi-scale image pyramids for embedded robot vision. In: **Proceedings of the 14th TAROS: Towards Autonomous Robotic Systems**. Oxford, UK: 2013.

ENTSCHEV, P. A.; VIEIRA NETO, H. Towards Embedded Robot Vision for Multi-Scale Object Recognition: Repeatability of Interest Points Detected in Half-octave Binomial Pyramids. In **Proceedings of the 4th International Conference on Pervasive and Embedded Computing and Communication Systems**. Lisbon, Portugal: 2014.

REFERENCES

BAY, H. et al. Speeded-up robust features (SURF). **Computer Vision and Image Understanding**, Elsevier, v. 110, n. 3, p. 346–359, 2008.

BRADSKI, G.; KAEHLER, A. **Learning OpenCV: Computer Vision with the OpenCV library**. [S.l.]: O'Reilly Media, 2008.

BROWN, M.; LOWE, D. G. Invariant features from interest point groups. In: **Proceedings of the 13th British Machine Vision Conference**. [S.l.: s.n.], 2002.

BURT, P.; ADELSON, E. The Laplacian pyramid as a compact image code. **IEEE Transactions on Communications**, IEEE, v. 31, n. 4, p. 532–540, 1983.

COLEY, G. **Beagleboard-xM system reference manual, revision A2**. [S.l.], 2010. Disponível em: <http://beagle.s3.amazonaws.com/design/xM-A/BB_xM_SRM_A2_01.pdf>.

CROWLEY, J. L.; PARKER, A. C. A representation for shape based on peaks and ridges in the difference of low-pass transform. **IEEE Transactions on Pattern Analysis and Machine Intelligence**, IEEE, n. 2, p. 156–170, 1984.

CROWLEY, J. L.; RIFF, O. Fast computation of scale normalised gaussian receptive fields. In: SPRINGER. **Scale Space Methods in Computer Vision**. [S.l.], 2003. p. 584–598.

CROWLEY, J. L.; RIFF, O.; PIATER, J. Fast computation of characteristic scale using a half octave pyramid. In: **Proceedings of the International Workshop on Cognitive Vision**. [S.l.: s.n.], 2002.

CROWLEY, J. L.; STERN, R. M. Fast computation of the difference of low-pass transform. **IEEE Transactions on Pattern Analysis and Machine Intelligence**, IEEE, v. 6, n. 2, p. 212–222, 1984.

ENTSCHEV, P. A.; VIEIRA NETO, H. Efficient construction of SIFT multi-scale image pyramids for embedded robot vision. In: **Proceedings of the 14th TAROS: Towards Autonomous Robotic Systems**. Oxford, UK: [s.n.], 2013.

ENTSCHEV, P. A.; VIEIRA NETO, H. Towards embedded robot vision for multi-scale object recognition: Repeatability of interest points detected in half-octave binomial pyramids. In: **Proceedings of the 4th International Conference on Pervasive and Embedded Computing and Communication Systems**. Lisbon, Portugal: [s.n.], 2014.

FLYNN, M. J. Very high-speed computing systems. **Proceedings of the IEEE**, IEEE, v. 54, n. 12, p. 1901–1909, 1966.

GONZALEZ, R. C.; WOODS, R. E. **Digital Image Processing (3rd Edition)**. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 2006. ISBN 013168728X.

HALFACREE, G.; UPTON, E. **Raspberry Pi User Guide**. [S.l.]: Wiley, 2012.

KARADAYI, K. et al. Strategies for mapping algorithms to mediaprocessors for high performance. **IEEE Micro**, IEEE, v. 23, n. 4, p. 58–70, 2003.

LOWE, D. G. Object recognition from local scale-invariant features. In: IEEE. **Proceedings of the 7th IEEE International Conference on Computer Vision**. [S.l.], 1999. v. 2, p. 1150–1157.

LOWE, D. G. Distinctive image features from scale-invariant keypoints. **International Journal of Computer Vision**, Springer, v. 60, n. 2, p. 91–110, 2004.

LOWE, D. G. Demo software: SIFT keypoint detector. **University of British Columbia**, 2005. Disponível em: <http://www. cs.ubc.ca/ lowe/keypoints/>.

NEUBECK, A.; GOOL, L. V. Efficient non-maximum suppression. In: IEEE. **Proceedings of the 18th IEEE International Conference on Pattern Recognition**. [S.l.], 2006. v. 3, p. 850–855.

PELEG, A.; WILKIE, S.; WEISER, U. Intel MMX for multimedia PCs. **Communications of the ACM**, ACM, v. 40, n. 1, p. 24–38, 1997.

REDDY, V. G. Neon technology introduction. **ARM Corporation**, 2008. Disponível em: <http://www.arm.com/files/pdf/AT_-_NEON_for_Multimedia_Applications.pdf>.

VIOLA, P.; JONES, M. Rapid object detection using a boosted cascade of simple features. In: IEEE. **Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition**. [S.l.], 2001. v. 1, p. I–511.

Visual Geometry Group. Affine covariant regions dataset. **University of Oxford**, 2004. Disponível em: <http://www.robots.ox. ac.uk/~vgg/data/data-aff.html>.