UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
CURSO DE ENGENHARIA DE COMPUTAÇÃO

THOMAS VAITSES FONTANARI

Simultaneous Magnification of Subtle Motions and Color Variations in Videos using Riesz Pyramids

Work presented in partial fulfillment of the requirements for the degree of Bachelor in Computer Engineering

Advisor: Prof. Dr. Manuel Menezes de Oliveira Neto

Porto Alegre
June 2021

ABSTRACT

Videos often contain subtle motions and color variations that cannot be easily observed. Examples include, for instance, head motion and changes in skin face color due to blood flow controlled by the heart pumping rhythm. A few techniques have been developed to magnify these subtle signals. However, they are not easily applied to many applications. First of all, previous techniques were targeted specifically towards magnification of either motion or color variations. Trying to magnify both aspects applying two of these techniques in sequence does not produce good results. We present a method for magnifying subtle motions and color variations in videos simultaneously. Our approach is based on the Riesz pyramid, which was previously used only for motion magnification. Besides modifying the local phases of the coefficients of this pyramid, we show how altering its local amplitudes and its residue can be used to produce intensity (color) magnification. We demonstrate the effectiveness of our technique in multiple videos by revealing both subtle signals simultaneously. Finally, we also developed an Android application as a proof-of-concept that can be used for magnifying either motion or color changes.

Keywords: Eulerian video magnification. riesz pyramids.

LIST OF ABBREVIATIONS AND ACRONYMS

GUI	Graphical User Interface
NDK	Native Development Kit
JNI	Java Native Interface
EVM	Eulerian Video Magnification
DVMAG	Dynamic Video Magnification

LIST OF FIGURES

LIST OF TABLES

CONTENTS

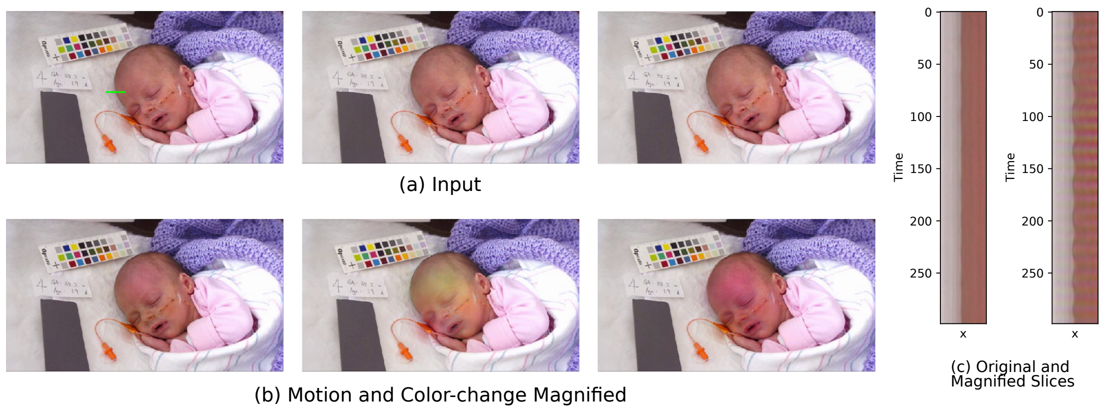
1 INTRODUCTION

The world is full of subtle motions and color variations in time that tend to be invisible to the naked eye, but nevertheless carry useful information in a wide range of areas. For instance, the cardiac cycle causes almost imperceptible skin color variations in human faces and, the pattern of our respiration causes our shoulders and chest to move slightly, and small motions of the eye may be symptomatic of a neural disease. Structures undergoing pressure or bearing heavy weights may also deform causing subtle motions and a drone will move slightly in order to maintain its stability during a stationary flight. The ability to extract and visualize subtle signals in videos, therefore, has many practical applications, ranging from the development of tools for supporting medical diagnostic to the inspection of industry equipment. Given this potential, a variety of applications have been developed (ARANGO et al., 2018; LAURIDSEN et al., 2019; WADHWA, 2016; Le Ngo et al., 2018; CHEN; PICARD, 2017; PERROT et al., 2018; DAVIS et al., 2015; BALAKRISHNAN; DURAND; GUTTAG, 2013; BELAID et al., 2010) as well as a number of algorithms for video motion and color magnification and analysis have been recently proposed. (LIU et al., 2005; WU et al., 2012; WADHWA et al., 2013; WADHWA et al., 2014b; ELGHARIB et al., 2015; WU et al., 2018b; LIU et al., 2014).

This motion microscope and the signals they reveal, however, are not so easily accessible in our everyday use. One of the reason for this is that, while these previous solutions have achieved considerable success in magnifying either subtle motions or color variations, none of them was designed for joint magnification of both motions and color variations. In order to obtain both effects on a video, one needs to apply the methods sequentially, which in general produces deformed results since the first signal which is magnified will influence the next magnification step. This is also problematic if one wishes to create a real time application. Another reason for the difficult in using these methods is that, while in general one records videos with a smartphone, there is no mobile application for video magnification in real-time. Our work therefore aims at providing a solution for these problems and we focus primarily in developing a technique for both motion and color-change magnification simultaneously. Nevertheless, we also develop an Android application which is able to magnify either color changes or motion.

Having a unified framework capable of handling both color-change and motion magnification as the one we propose here can be useful in cases where both signals are of interest, such as is the case when visualizing the blood flow caused by the pulse of a

Figure 1.1 – Magnification of subtle motion and color variations produced by our technique. (a) Three non-consecutive frames from a video of a baby. (b) Motion and color magnification applied to the head of the baby for the corresponding frames on top. (c) Comparison of the colors along a small strip of pixels (shown in green in (a) left) across the original (left) and magnified (right) video frames. The variations in width (motion) and colors of the portion of the segment corresponding to the baby skin result from magnification.


(a) Input
(b) Motion and Color-change Magnified
(c) Original and Magnified Slices
Time
x
0
50
100
150
200
250

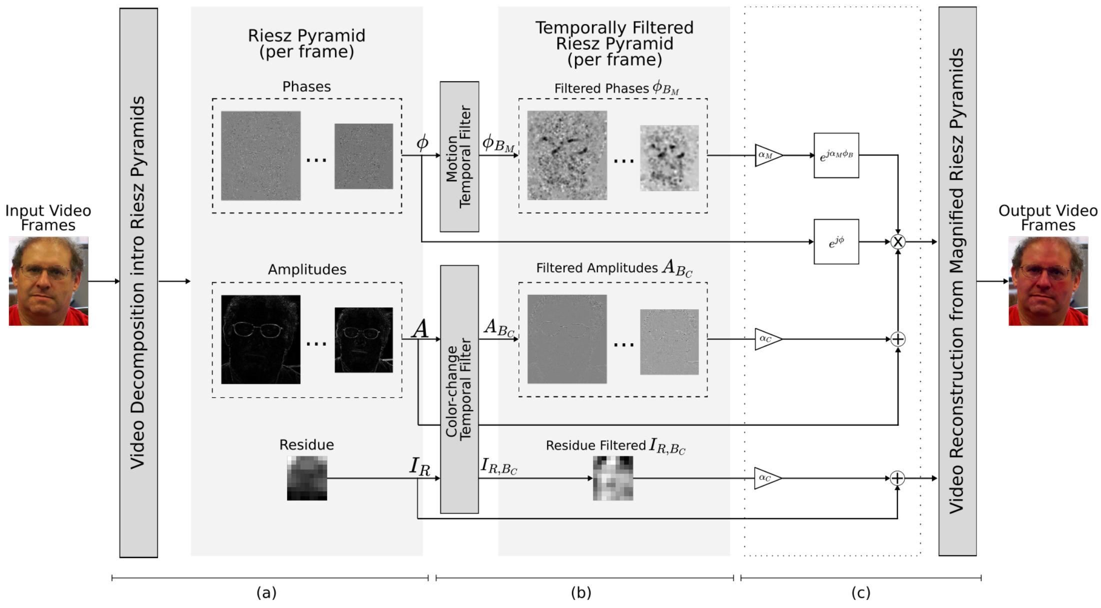
Source video is from Wu et al. (2012).

person. In this case, the motion is synchronized with the blood flow. The two signals of interest, however, might have different frequencies, not needing to be synchronous. Consider, for instance, head and chest motion due to breathing, and skin color variations due to cardiac rhythm.

Figure 1.1 illustrates the use of our technique to perform joint motion and color magnification to the frames of a video. Figure 1.1 (a) shows three (non-consecutive) frames from a video of a baby. Figure 1.1 (b) shows the results produced by our technique for motion and color magnification applied to the head of the baby for the corresponding frames shown on top. The variation in color is more easily noticeable. Figure 1.1 (c) compares the colors of a small strip of pixels along the original (left) and magnified (right) video frames. The strips of pixels are indicated by the green line segment ranging from the baby's head to the crib bedspread, shown in Figure 1.1 (a) (left). Note the variations in width (motion) and color of the portion of the segment corresponding to the baby skin in Figure 1.1 (c) (right).

Our technique uses the Riesz pyramids introduced for subtle motion magnification in (WADHWA et al., 2014b). The Riesz pyramids are based on the Riesz transform (FELSBERG; SOMMER, 2001; UNSER; SAGE; VILLE, 2009) which provides an image representation in which local phase and local amplitude are separated from each other. We then use the local phases for motion magnification, as also done by Wadhwa et al. (2014b), and use the local amplitudes, together with the low-pass residue of the

Figure 1.2 – Our simultaneous amplification of subtle motion and intensity/color variation pipeline. (a) The first stage decomposes each frame of the input video into a Riesz pyramid: *phase*, *amplitude*, and *residue*. (b) Per-pixel temporal filter applied to the phase, amplitude, and residue of the pyramid selects the frequencies bands to be magnified: ϕ_{B_M} (for motion), and A_{B_C} and I_{R,B_C} (for intensity/color). (c) The magnified video frames are recovered after scaling the selected frequency bands by factors α_M (motion) and α_C (intensity/color) and recombining them with the original Riesz pyramid elements.



Input Video Frames
Video Decomposition intro Riesz Pyramids
Riesz Pyramid (per frame)
Phases
Amplitudes
Residue
ϕ
A
I_R
Motion Temporal Filter
Color-change Temporal Filter
ϕ_{B_M}
A_{B_C}
I_{R,B_C}
Temporally Filtered Riesz Pyramid (per frame)
Filtered Phases ϕ_{B_M}
Filtered Amplitudes A_{B_C}
Residue Filtered I_{R,B_C}
α_M
α_C
α_C
$e^{j\alpha_M\phi_B}$
$e^{j\phi}$
Video Reconstruction from Magnified Riesz Pyramids
Output Video Frames
(a)
(b)
(c)

Source: The authors.

pyramid, for magnifying color changes. Figure 1.2 summarizes this process. The first stage (Figure 1.2 (a)) performs a per-frame input video decomposition into Riesz pyramids, resulting in a *phase pyramid*, an *amplitude pyramid*, and a *residue* term. The second stage (Figure 1.2 (b)) applies a per-pixel temporal filter to the phase pyramid to select the (phase) frequency band ϕ_{B_M} for which motion should be magnified. An independent per-pixel temporal filter is applied to both the amplitude pyramid and to the Riesz pyramid residual. These select the (amplitude) frequency band A_{B_C}, and (residue) frequency band I_{R,B_C} for which color variations should be magnified. The final stage (Figure 1.2 (c)) recovers the resulting magnified video frames. For this, it scales the selected phase by α_M, and amplitude and residue frequency bands by α_C, recombines them to the original phase ϕ, amplitude A, and residue I_R values, producing the magnified video frames.

Our Android application, on the other hand, implements separately motion and color-change magnification using the techniques previously described in the literature. More specifically, we implement the linear Eulerian approach (WU et al., 2012) for color-change magnification and the Riesz pyramids technique for motion magnification. The application allows the user to select the magnification factors and the temporal frequencies which should be magnified.

The **contributions** of this work include:

- A technique for simultaneous magnification of subtle motions and color variations in videos (Section 3);
- A demonstration of how the Riesz Pyramids can be used to magnify color variations (Section 3);
- A chrominance-based mask technique for modifying only regions of interest (Section 3.2);
- A Python application for using our Riesz-based motion and color-change magnification technique (Section 4);
- A proof-of-concept Android application for magnifying either motion or color changes (Section 4) .

In the next section, we given an overview about the previous work developed on Eulerian video magnification and describe how they relate to ours. We also dive deeper in the techniques which are more closely related to the method we introduce and our implementations. In Section 4, we briefly describe our implementations and Section 5 shows the results that were obtained using our magnification method. Finally, Section 6 concludes the work. We have also added multiple Appendices with more details about our Android implementations (Appendix C) and the other methods for motion magnification (Appendix B) which in the main text are only mentioned briefly.

2 RELATED WORK

The first method for motion magnification in videos was a layer-based technique that relied on computing the trajectories of feature points obtained from a reference frame (LIU et al., 2005). This algorithm is computationally intensive, requiring hours for processing even short video sequences with just a few seconds long. Because of its dependence on tracking features, the technique was later described as a *Lagrangian* approach. The technique is restricted to motion and cannot be used for magnification of intensity/color variations.

Later on, Wu et al. (2012) introduced an approach that relied only on local spatial and temporal filtering. As such, the solution was referred to as an *Eulerian* method. The technique was based on computing the Laplacian pyramid for each frame of the video and manipulating the intensities of each pixel in the pyramid. Specifically, each pixel of the pyramid was temporally filtered (across the various frames), selecting only motions whose temporal spectrum was contained in a temporal sub-band of interest. The resulting temporally-filtered signals could then be multiplied by a magnification factor and added back to the original pyramid. Collapsing this modified pyramid produced a magnified version of the frame. Furthermore, applying this process to the residue of the Laplacian pyramid would magnify the color changes in the image. This technique, however, produced low-quality and limited motion magnification, resulting from clipping artifacts which distorted the frames. The technique also significantly increased the noise levels when motion was magnified.

To improve the quality of the earlier linear Eulerian method and increase the amount of motion magnification, Wadhwa et al. introduced phase-based motion magnification techniques (WADHWA et al., 2013; WADHWA et al., 2014b). Those methods were inspired by previous works that had already demonstrated the relation between local phases of frames and movement (FREEMAN; ADELSON; HEEGER, 1991; Gautama; Van Hulle, 2002). Furthermore, the phase-based approaches still have an Eulerian character, since they rely on local spatial and temporal filtering. Their basic idea is to obtain a measure of *local phase*, which is related to the local motion of a region in the video. Manipulating these phases is then equivalent to manipulating the local motion. The results obtained using these techniques were a significant improvement over the linear Eulerian method of Wu et al. (2012), allowing for larger magnification factors and introducing much lower noise levels. Furthermore, Wadhwa et al. (2013) suggested (they have not demonstrated it) that the image representation which they used for motion magnification, the *steerable pyramid* (SIMONCELLI; FREEMAN, 1995; FREEMAN; ADELSON et al., 1991; SIMONCELLI et al., 1992), could also be used for magnifying color changes in videos. In our work, we show how the Riesz pyramids, introduced by Wadhwa et al. (2014b), where they were only used for motion magnification, can also be used to magnify intensity/color variations. Unlike Wu et al. (2012) who proposed to perform color magnification by scaling the Laplacian pyramid residue, we show that high-quality intensity/color amplification can be achieve by scaling both the amplitude coefficients of the Riesz pyramid and the pyramid's residue. As such, our work is the first to demonstrate simultaneous high-quality magnification of subtle motions using a phase-based approach and intensity/color variations in videos.

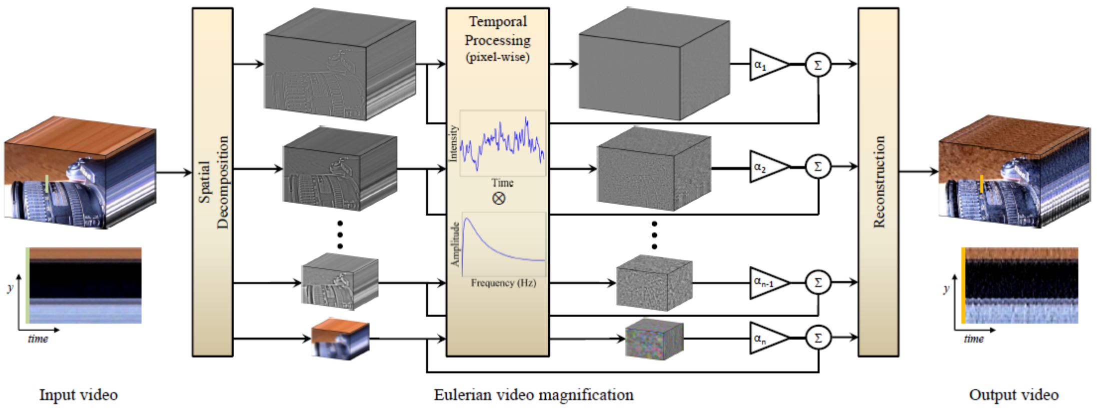
Additional techniques were designed to specifically overcome the problems that larger motions or color changes cause to the Eulerian methods, as the large variations also get modified by the algorithm (ZHANG; PINTEA; GEMERT, 2017; ELGHARIB et al., 2015; WU et al., 2018a). The large motions, however, do not fit the Eulerian frameworks, as they rely on the premise that motions are local. Moreover, large color variations are generally not the ones of interest, as they are already visible, and magnifying them leads to clipping artifacts as their magnified values might get too big when trying to amplify the subtle signals. These methods, however, handle either motion or color-change amplification, but not both simultaneously. Sharing this same limitation, some deep-learning techniques have been recently proposed (CHEN; MCDUFF, 2018; OH et al., 2018). Oh et al. (2018) presented a technique for magnifying small motions, and Chen and McDuff (2018) used of separate models for motion and color-change magnification.

In the next sections, we describe the main methods which more closely relate to our work. For completeness, we have also described other methods for magnification of subtle signals in Appendix B.

2.1 Linear Eulerian Motion and Color-change Magnification

The Linear Eulerian Video Magnification (WU et al., 2012) is an algorithm based solely on spatial and temporal filtering that is able to amplify subtle motion and color changes in a video. The key idea of their work was the realization that amplifying the intensity variations of a pixel, besides magnifying color changes, also magnified small motions involving the pixel. Moreover, temporal filtering of the intensities of a pixel

Figure 2.1 – Linear Eulerian Video Magnification



Source: (WU et al., 2012)

corresponded to selecting motions or signals containing only a desired frequency band, thus allowing frequency-based motion selection. For instance, a band-pass filter around 60 bpm could be used to magnify motion related to the heartbeats of a person.

The first step of the algorithm is to decompose each frame into sub-bands by building its Laplacian Pyramid. Then, each level of the pyramid is filtered temporally such that only pixel variations with the appropriate temporal frequency bands are retained. Each temporally filtered sub-band is multiplied by an appropriated magnification parameter α and added back to the pyramid, which is finally collapsed to generate the magnified frame. The algorithm is illustrated in Figure 2.1. As will be seen, this same framework is capable of magnifying both small color changes and motions that vary within a frequency range determined by the temporal filter.

As explained in their work, this process is able to magnify motion in videos because temporally filtering a video pixel-wise isolates the displacements, as long as the image can be locally linearly approximated. In order to see this, we consider a 1D video characterized only by translation over time $I(x, t) = f(x + \delta(t))$, where $f(x) = I(x, 0)$ is the video at $t = 0$. The goal of motion magnification is then to produce $\hat{I}(x, t) = f(x + (1 + \alpha)\delta(t))$. If $f(x + \delta(t))$ can be linearly approximated at x, then we can write its first order Taylor series at x

$$I(x, t) = f(x) + \delta(t)\frac{df(x)}{dx}. \tag{2.1}$$

Applying a band-pass temporal filter to this signal, one can remove the constant term $f(x)$. Moreover, each frequency component of the displacement signal $\delta(t)$ will also be modified according to the filter used. For instance, if an ideal band-pass filter that allows only frequencies ranging from $0.8Hz$ to $1.1Hz$ to pass is used, then only the frequency components from the displacement $\delta(t)$ that fall into this frequency range remain. One can represent this mathematically by saying that the filter $\mathcal{T}$ scales each frequency component $\delta_k(t)$ of $\delta(t)$ by a parameter γ_k. Hence, we can write

$$\mathcal{T}\{\delta(t)\} = \sum_k \mathcal{T}\{\delta_k(t)\} = \sum_k \gamma_k \delta_k(t). \tag{2.2}$$

Filtering each pixel of the video I then results in the quantity

$$B(x,t) = \frac{df(x)}{dx} \sum_k \gamma_k \delta_k(t). \tag{2.3}$$

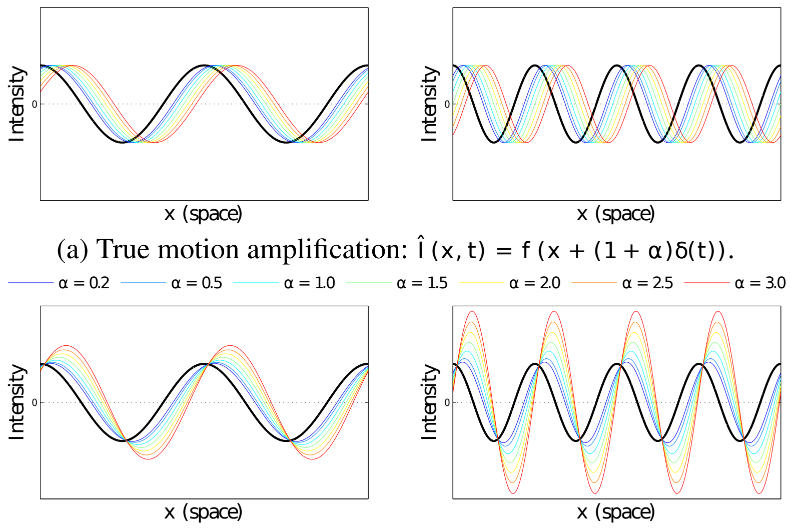
Multiplying this signal by a magnification factor α and adding it back to the original video will result in a video where the selected frequency components of the displacement were magnified, as long as the linear approximation still holds locally. That is,

$$\begin{aligned} I(x,t) + \alpha B(x,t) &= f(x) + \delta(t)\frac{df(x)}{dx} + \alpha \frac{df(x)}{dx} \sum_k \gamma_k \delta_k(t) \\ &= f(x) + \frac{df(x)}{dx} \sum_k (1 + \alpha\gamma_k)\delta_k(t) \\ &\approx f(x + \sum_k (1 + \alpha_k)\delta_k(t)) \\ &= \hat{I}(x,t), \end{aligned} \tag{2.4}$$

where $\alpha_k = \alpha\gamma_k$. In the last step, it was also assumed that the linear approximation holds for $\hat{I}(x,t)$. In that case, the motion magnified signal $\hat{I}(x,t)$ has been produced, with the additional possibility of selecting which motions should be magnified by choosing the temporal filter. If the signal with all motions magnified is targeted, then one can choose a temporal filter that only removes the constant term $f(x)$ and do not alter the other temporal frequencies.

It is possible to draw constraints on how much motion magnification can be achieved for specific spatial frequencies from the fact that the linear approximations must hold. Wu et al. (2012) consider the case of a sinusoidal wave under translation to deduce the guideline $(1+\alpha)\delta(t) < \frac{\lambda}{8}$, where λ is the wavelength of the sinusoid. Qualitatively, this means that the magnification factor needs to be smaller for higher spatial frequencies. This is illustrated in Figure 2.2. Therefore, spatially decomposing each frame before applying the temporal filter is necessary in order to allow for lower amplification in higher frequencies.

Figure 2.2 – Higher spatial frequencies are more distorted by the magnification process. The images to the left have lower spatial frequencies and the magnification process produces less distortion for the same magnification factor α then when higher spatial frequencies (right) are magnified.

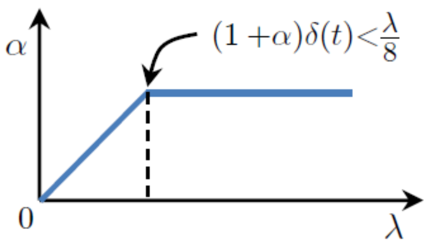


(a) True motion amplification: $\hat{I}(x,t) = f(x + (1+\alpha)\delta(t))$.

(b) Motion amplification via temporal filtering: $\tilde{I}(x,t) = I(x,t) + \alpha B(x,t)$.

Source: (WU et al., 2012)

Figure 2.3 – Amplification factor as a function of the wavelength



Source: (WU et al., 2012)

The authors use a parameter λ_c to account for this, below which the attenuation factor α decreases linearly towards 0 (Figure 2.3).

The same framework can also be used for color amplification. In this case, the finest levels of the pyramid are not amplified – that is, only the lowest spatial frequency components are modified. The result of this is analogous to using a low-pass filter in order to increase the Signal-to-Noise Ratio of the color changes. A temporal filter is used as well in order to select the desired frequencies. This process also amplifies subtle motions as artifacts. However, since, the sub-bands containing higher frequencies are not modified, motion will not be so much altered.

It is important to take the noise in the input video into account when applying this method. Because it is based on directly scaling the values of pixels, it generally increases the noise level in the video. Moreover, when color-change magnification is targeted, the subtle signals related to it are likely weaker than noise. For that reason, it is imperative to use only the higher (more coarse) levels of the Laplacian pyramid, since in the other levels the subtle signal will not have emerged from noise.

The Eulerian Video magnification technique constituted a significant step forward in this area. Its simplicity allowed real-time magnification of subtle motions. Nevertheless, the technique was able only to magnify motions by a small amount without incurring in significant artifacts and noise amplification.

2.2 Phase-based Video Motion Processing

Phase-based motion magnification techniques were later proposed by Wadhwa et al. (2013) in order to reduce the noise levels of the previous linear Eulerian approach and to increase the possible magnification factors. The main intuition behind phase-based motion magnification is the fact that shifting the phase of waves can be perceived as motion (WADHWA et al., 2013). The relation between phase shifting and motion magnification can be understood by considering the relation between a global phase, obtained through Fourier decomposition, and the displacement. For simplicity, we present the concept in 1D and later extend it to 2D. Thus, consider a single video scanline from a video $I(x, y, t)$, where x and y are respectively the pixel column and row coordinates, and t is time representing the individual video frames. We further simplify the scanline notation by dropping the y coordinate. Thus, a scanline can be described by a function f: $I(x, t) = f(x+\delta(t))$, where $\delta(t)$ is the displacement and $\delta(0) = 0$. The targeted motion magnified version can be written as $\hat{I}(x, t) = f(x+(1+\alpha)\delta(t))$. Decomposing each “frame" of this video using the complex representation of the Fourier series gives

$$f(x + \delta(t)) = \sum_{\omega=-\infty}^{\infty} A_\omega e^{j\omega(x+\delta(t))}, \tag{2.5}$$

where A_ω is the contribution of the frequency ω to the displaced image and $\omega(x + \delta(t))$ is its phase (notice that Equation 2.5 is in the spatial domain and not in the frequency domain). An arbitrary spatial frequency ω of such a 1D frame of the video can therefore be written as

$$I_\omega(x, t) = A_\omega e^{j\omega(x+\delta(t))}. \tag{2.6}$$

Removing the constant term ωx from the phase $\omega(x + \delta(t))$, which can be done by subtracting from it the phase of the frame at $t = 0$, one obtains $\omega\delta(t)$. This term can then be multiplied by a (magnification) factor α and added back to the phase shift by multiplying Equation 2.6 by the complex exponential $e^{\alpha\omega\delta(t)}$, producing

$$\hat{I}_\omega(x,t) = A_\omega e^{j\omega(x+(1+\alpha)\delta(t))}. \tag{2.7}$$

If one applies this same phase shifting process to all frequencies ω and sum over them as in Equation 2.5, one obtains the targeted motion magnified version of the video

$$\hat{I}(x,t) = f(x + (1+\alpha)\delta(t)). \tag{2.8}$$

Hence, by shifting the phases of each pixel x, one also obtains a motion magnified version of the video. However, using the global phases obtained through a Fourier decomposition would not work in general, since each wave covers the whole space. Motion, on the other hand, is generally local. Therefore, in order to magnify motions, it's necessary to obtain a *local phase*. One of the ways to obtain a measure of a local phase for motion magnification is to use complex Steerable pyramids (WADHWA et al., 2013) and the other one is to use Riesz pyramids (WADHWA et al., 2014b). In the next section, we describe the method using the Riesz pyramids, as it is the base of what we have developed in this work. More information about motion magnification using the Steerable pyramids can be found in Appendix B.1.

2.2.1 Motion Magnification Using Riesz Pyramids

Wadhwa et al. (2014b) introduced an efficient approach for phase-based motion magnification which uses the local phases obtained from a *Riesz pyramid*. The Riesz pyramid is based on the two-dimensional extension of the Hilbert transform. The transfer function $H_{\mathcal{H}}(\omega)$ of the Hilbert transform $\mathcal{H}$ is given by

$$H_{\mathcal{H}}(\omega) = -j\ \operatorname{sign}(\omega) = -j\frac{\omega}{\|\omega\|}, \tag{2.9}$$

where $\operatorname{sign}(\omega)$ is the *sign* function. The Hilbert transform phase shifts each component (i.e., each $\sin$ and $\cos$ function associated to each frequency ω) of the input signal by 90°. Thus, each $\cos$ becomes a $\sin$ and each $\sin$ becomes a $-\cos$. This allows the computation of the quadrature pair

$$f(x) + j\,\mathcal{H}\{f(x)\} = A(x)e^{j\phi(x)}, \tag{2.10}$$

where $f(x)$ is the input signal being transformed, and $A(x)$ and $\phi(x)$ are respectively the *local amplitude* and *local phase* of $f(x)$.

As an example, we can consider the quadrature pair obtained when $f(x) = \cos(\omega x)$. Its transform is simply $\sin(\omega x)$, which results in the quadrature pair $\cos(\omega x) + j\,\sin(\omega x) = e^{j\omega x}$. In general, however, a function will not be defined by a single component. Hence, when analysing more complex signals, it is necessary to use a multi-scale approach by first decomposing the original signal into multiple sub-bands and treat them independently. The local amplitude and phase for each sub-band can then be obtained from the quadrature pair corresponding to the sub-band. The obtained local phases can be used to estimate and magnify the motion at each pixel x. However, before one can apply this to videos, it is necessary to have a generalization of the Hilbert transform to 2D.

The Riesz transform $\mathcal{R}$ generalizes the Hilbert transform to two or more dimensions, and in 2D can be defined by the pair of transfer functions

$$H_{\mathcal{R}}(\vec{\omega}) = \left(\frac{-j\omega_x}{||\vec{\omega}||}, \frac{-j\omega_y}{||\vec{\omega}||}\right), \tag{2.11}$$

where ω_x and ω_y are the frequency coordinates in the frequency domain and $\vec{\omega} = (\omega_x, \omega_y)$, and $||\cdot||$ is the L2-norm operator. As was done with the one-dimensional case, one can write this pair together with the original function. Since this results in a three component entity, we cannot write it as a complex number an instead write the vector

$$(I(x,y), R_1(x,y), R_2(x,y)), \tag{2.12}$$

where $I(x,y)$ is the two-dimensional original function (e.g., an input video frame or a sub-band of a frame), and $R_1(x,y)$ and $R_2(x,y)$ are the results of applying the first and second components of the Riesz transform to $I(x,y)$. This vector can be written in spherical coordinates as

$$I(x,y) = A(x,y)\cos(\phi(x,y)), \tag{2.13}$$
$$R_1(x,y) = A(x,y)\sin(\phi(x,y))\cos(\theta(x,y)), \tag{2.14}$$
$$R_2(x,y) = A(x,y)\sin(\phi(x,y))\sin(\theta(x,y)), \tag{2.15}$$

where $A(x, y)$, $\phi(x, y)$ and $\theta(x, y)$ are respectively the *local amplitude*, *local phase* and *local orientation* of the pixel at position (x, y). The local phase in this equation has a similar meaning to that of the phase obtained for the one-dimensional case with the Hilbert transform. Furthermore, the local phase is associated to the wave that points towards the local orientation θ, which is in fact the dominant orientation (i.e., the gradient) at pixel (x, y). The value of the local phase (ϕ) can be obtained from the following quadrature pair, which is analogous to the one produced by the Hilbert transform:

$$I + jQ = Ae^{j\phi}, \tag{2.16}$$

where

$$A = \sqrt{I^2 + R_1^2 + R_2^2}, \tag{2.17}$$

and

$$Q = \sqrt{R_1^2 + R_2^2} = A \sin\phi. \tag{2.18}$$

The reason why this local phase can be used for motion magnification is illustrated by Wadhwa et al. (2014b) using a 2D sinusoidal wave being translated horizontally. This is described by the equation $I(x, y, t) = \cos(\omega_x(x - \delta(t)) + \omega_y y)$, where $\delta(t)$ corresponds to the displacement over time. The Riesz transform of this signal is

$$\mathcal{R}\{I(x, y, t)\} = \frac{(\omega_x, \omega_y)}{\sqrt{\omega_y^2 + \omega_y^2}} \sin(\omega_x(x - \delta(t)) + \omega_y y), \tag{2.19}$$

where $\mathcal{R}$ is the Riesz transform operator as defined by its transfer function in Equation 2.11. The quadrature pair $I + jQ$ can then be computed using $Q = \sin(\omega_x(x - \delta(t)) + \omega_y y)$. The local phase at each pixel is therefore

$$\phi(x, y) = \omega_x x + \omega_y y - \omega_x \delta(t). \tag{2.20}$$

Subtracting from Equation 2.20 the phase of the first frame (where $\delta(t) = 0$), the factor $-\omega_x \delta(t)$ can be isolated and amplified by multiplication with a constant α_M to obtain the value $-\alpha_M \omega_x \delta(t)$. Finally, the phase of the quadrature pair $I + jQ$ is updated accordingly by multiplication by the complex exponential $e^{-j\alpha_M \omega_x \delta(t)}$. The real part of the resulting pair corresponds to the motion amplified video

$$\hat{I}(x, y, t) = \cos(\omega_x(x - (1 + \alpha)\delta(t)) + \omega_y y). \tag{2.21}$$

As in the one-dimensional case, the image needs to be decomposed into multiple non-oriented sub-bands before a phase analysis can be performed. This can be done using a Laplacian pyramid. The Riesz transform is then applied to each of the levels of the Laplacian pyramid, resulting in the Riesz pyramid. The local phases of each level of the Riesz pyramid can then be modified through the process just described, and the real parts of each level is treated as a level of a Laplacian pyramid, which is then collapsed in order to produce the magnified frame.

Furthermore, the local phases can also be temporally and spatially filtered before being used to modify the pyramid. Similarly to Wu et al. (2012), Wadhwa et al. (2014b) temporally filter the spatial local phases (i.e., the local phases in each video frame) in order to select only the motions whose temporal frequencies fall in the band of interest. However, as they show, filtering directly the local phases leads to discontinuity problems. For instance, if the local orientation θ is limited to the interval $[0, \pi]$ while the local phase goes from $[-\pi, \pi]$, a small change in the orientation between two adjacent pixels (spatially or temporally) can lead to a brusque phase change from ϕ to $-\phi$. Therefore, they choose to instead filter the quantities

$$\phi \cos \theta \tag{2.22}$$

and

$$\phi \sin \theta \tag{2.23}$$

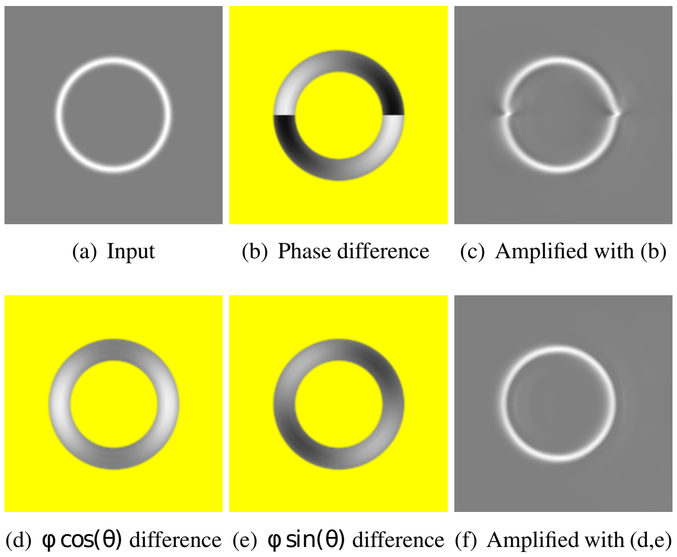
independently, which do not suffer from these discontinuities. These quantities together are called the *quaternionic phase* since they can be naturally derived from a quaternionic representation of the Riesz pyramid (see Appendix B.2). This is illustrated in Figure 2.4.

After the temporal filtering, Wadhwa et al. (2014a) have also filtered the phases spatially through the use of an amplitude weighted blur. That is, the phases are convoluted with a Gaussian kernel and weighted by the value of the amplitude of the Riesz pyramid coefficients. This spatial filtering is important in order to reduce noise from the computation of the phases and also to reduce errors from approximations used when constructing the Riesz pyramid. Finally, the filtered quantities are recombined. The equation below summarizes these steps

$$\cos\theta \frac{A\cos(\theta)\ \phi * K_p}{A * K_p} + \sin\theta \frac{A\sin(\theta)\ \phi * K_p}{A * K_p}. \tag{2.24}$$

The rest of the process proceeds as described earlier, that is, by multiplying the filtered phases (Eq. 2.24) by a magnification factor α and shifting the phases of the pixels

Figure 2.4 – The difference between filtering only the phase and filtering the quantities in Equations 2.22 and 2.23. The differences are computed between the input (a) and the same image shifted by half a pixel to the left. Middle gray corresponds to no phase difference and the regions with low amplitude are masked in yellow.



Source: (WADHWA et al., 2014b)

of the Riesz pyramid by multiplication with a complex exponential. Further details on motion magnification with the Riesz pyramids are referred to their work (WADHWA et al., 2014b) and to Appendix B.2.

3 INTENSITY AND MOTION MAGNIFICATION WITH THE RIESZ PYRAMIDS

Intensity/color variations can be magnified by modifying the residue and the local amplitudes given by the coefficients of the Riesz pyramids. The simplest way to do it is to magnify only the residue. In this case, if the Riesz pyramid is constructed based on the Laplacian pyramid, the result produced is similar to the color-change magnification in the linear Eulerian method developed by Wu et al. (2012). In order to magnify the residue, we temporally filter directly the intensities of its pixels in order to select the color variations of interest. The resulting band is then multiplied by a magnification factor and added back to the residue (see the bottom-most part of Figure 1.2 (c)). When the Riesz pyramid of the frame is collapsed at the end of the process, the result is a frame with intensity/color variations magnified.

Besides its residue, the amplitude of the Riesz pyramid coefficients also contains information on the intensity/color variations. Furthermore, since the amplitude is separated from the local phase, it is less affected by the motion of the pixel. This is in contrast to altering the intensity of the pixels or of the coefficients of a Laplacian pyramid directly. This process can be better understood by first considering its one-dimensional version, which uses the quadrature pair obtained through the Hilbert transform. Thus, consider a video given by a sinusoid undergoing a small amount of translation (phase shift, $\delta_M(t)$) and amplitude scaling ($\delta_C(t)$):

$$I(x,t) = \delta_C(t) A \cos \omega_0 (x - \delta_M(t)), \tag{3.1}$$

for which one wishes to construct its amplitude-only magnified version

$$\hat{I}(x,t) = (1+\alpha)\delta_C(t) A \cos \omega_0 (x - \delta_M(t)). \tag{3.2}$$

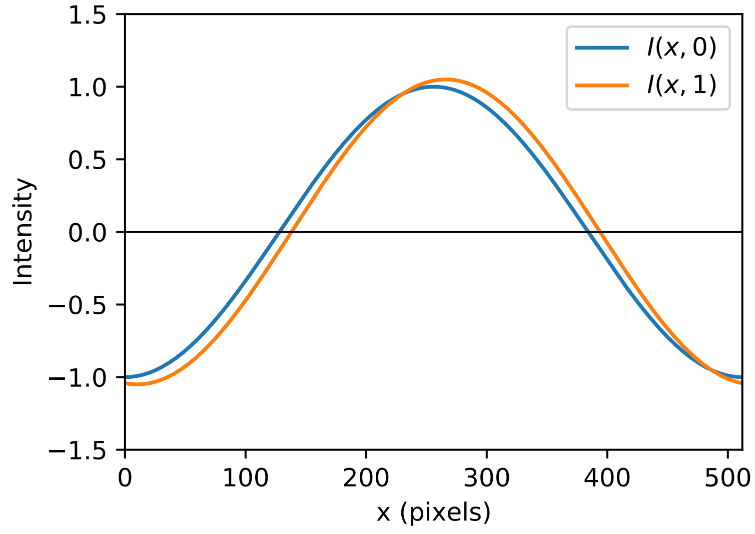
Figure 3.1 illustrates this situation, where the blue line represents the signal at time $t = 0$ and the orange line represents the signal at time $t = 1$.

The quadrature pair of $I(x,t)$ is given by

$$I_{QP}(x,t) = \delta_A(t) A e^{j\omega_0 (x - \delta_M(t))}, \tag{3.3}$$

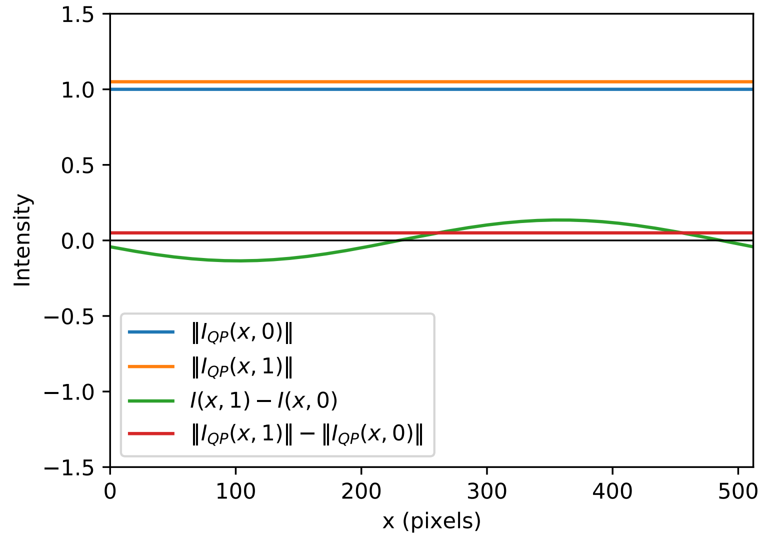
where $\delta_A(t)$ and $\delta_M(t)$ correspond respectively to the scaling and motion over time. Isolating the magnitude (Figure 3.2), one obtains $\delta_A(t)A$. This quantity can be multiplied by

Figure 3.1 – Sub-band of 1-D video at time steps $t = 0$ and $t = 1$. The first frame is both shifted to the right by 10 pixels and scaled up 0.05 times.



Source: the authors.

Figure 3.2 – The difference between the absolute values of the analytic representations of times 0 and 1 is shown by the red line. Since this is an ideal scenario, the magnitudes of the sub-bands capture perfectly only its intensity scaling, while the more straightforward linear difference approach, shown in green, is also influenced by the motion.



Source: the authors

a magnification factor α_A and the resulting value is added back to the original sub-band by adding to the sub-band the complex number number (see the mid portion of Figure 1.2 (c))

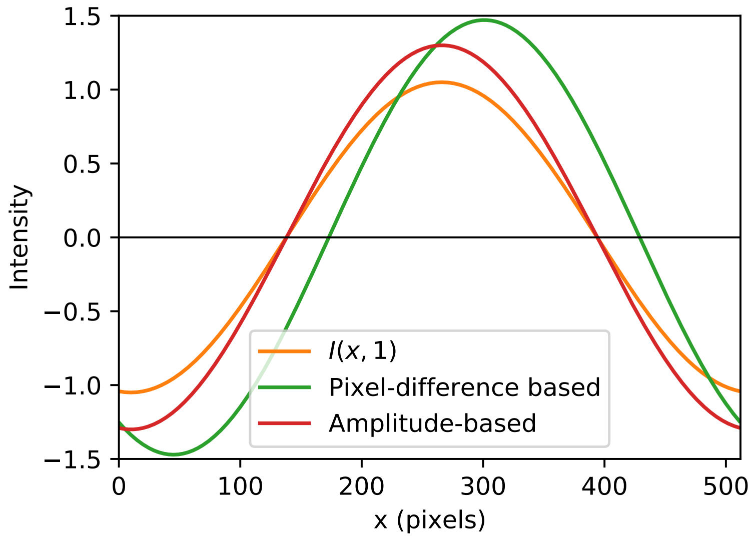
$$\alpha_A \delta(t) A e^{j\omega_0(x-\delta_M(t))}. \tag{3.4}$$

That is, the variation in magnitude is isolated and modified while the phase is kept unaltered. The real part of the resulting complex number is the intensity-magnified video:

$$\hat{I}(x,t) = \mathrm{Re}\{(1+\alpha)\delta_A(t)Ae^{j\omega_0(x-\delta_M(t))}\}, \tag{3.5}$$

while the imaginary part is the intensity-magnified version of the Hilbert transform of the input video $I(x,t)$. This is illustrated in Figure 3.3. As is done for the phases (in the case of motion amplification), one can also use a band-pass filter to select only the temporal frequencies of the amplitude signal which relate to the intensity/color variations that one wishes to magnify.

Figure 3.3 – The magnification of the color changes between time steps using the difference of the magnitudes of the quadrature pair is shown in red. Since in this case motion and scaling can be separated perfectly, the process using the magnitudes is not affected by the motion of the band, while the magnification based on the the direct computation of the differences of the pixels (in green) also magnifies motion.



Source: the authors

A similar process can also be performed with two-dimensional images and the Riesz transform. In that case, one writes

$$I_{QP}(t) = I(t) + jQ(t) = A(t)e^{j\phi(t)} \tag{3.6}$$

where the quantities $I(t)$ and $Q(t)$ refer to an arbitrary coefficient of the Riesz pyramids at coordinates (x,y). The amplitude is then given by

$$A = \sqrt{I^2+Q^2} = \sqrt{I^2+R_1^2+R_2^2} \tag{3.7}$$

Temporally filtering it and multiplying by the magnification factor α, we find similarly as before $A(t) = \sum_k \alpha_k A_k(t)$, where A_k are the frequency components of $A(t)$. This can be added back to the original image by first multiplying it with $e^{\phi(t)}$, resulting in the magnified signal

$$\hat{I}_A(t) = \sum_k (1 + \alpha_k) A_k(t) e^{j\phi(t)} \tag{3.8}$$

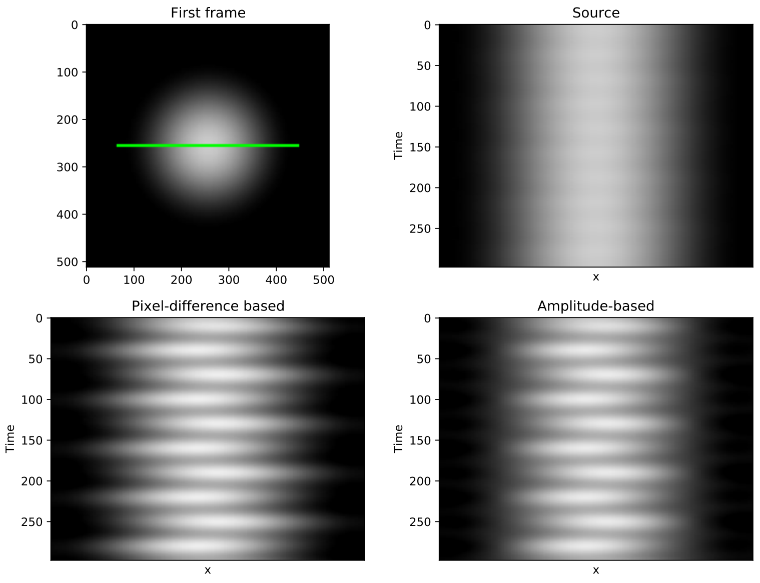
The magnified frame is then given by the real part of Equation 3.8. As mentioned previously, images in general are composed of multiple spatial frequency components. In order to perform the previously described process, then, it's necessary to first decompose the image into multiple non-oriented sub-bands, such as by using a Laplacian pyramid. The magnification can then then applied to each of the levels of the pyramid. It is important to notice, however, that color variations are usually very weak and can be indistinguishable from noise in the lower (more detailed) levels of the pyramids. For that reason, we have in general only magnified the amplitudes of the highest levels of the Riesz pyramid, besides the residue, when magnifying color changes.

The effects of magnifying the amplitudes of the Riesz Pyramid are shown in Figures 3.4 and 3.5. In this example, the video is built using only a single period of a cosine and clipped outside of the center of the image. The function moves periodically a small amount ($\pm 1px$) along the x axis and its amplitude is also periodically scaled 0.01 times. We also show for comparison the result of magnifying the pixels directly instead of the amplitudes of the quadrature pairs.

3.1 Simultaneous Motion and Intensity Magnification

Since the amplitude and the phase in $I + jQ$ are independent of each other, the amplitude of the Riesz pyramids can be modified without affecting the motion and vice-versa. We note that this would not be the case if instead of modifying the amplitudes of the pyramid we had chosen to amplify the intensity I of the pixels directly, as this would modify the phases of the quadrature pairs $I + jQ$. Furthermore, since the residue of the pyramid constitutes an additional spatial sub-band than those whose phases are modified when magnifying motion, we can also modify the residue without causing problems to the phase magnification. Although by modifying the residue some level of motion magnification is unavoidable, the residue contains only lower spatial frequencies and therefore magnifying its intensities will not produce significant motion of edges. Hence, to perform simultaneous phase-based motion magnification and intensity magnification, we magnify

Figure 3.4 – The first frame is shown in the first picture with a green strip. The evolution over time of this green strip is shown in the other images. As can be seen, the amplitude-based method also magnifies the color changes in the video. Moreover, because of the split of identity property, the amplitude-based method more closely magnifies only the intensity changes, while directly taking the differences of the intensity of the pixels also magnifies motion.



Source: the authors.

the local phases of the coefficients of the pyramid for motion magnification and use the residue and/or the amplitudes to magnify the intensity changes. Figure 1.2 illustrates the method.

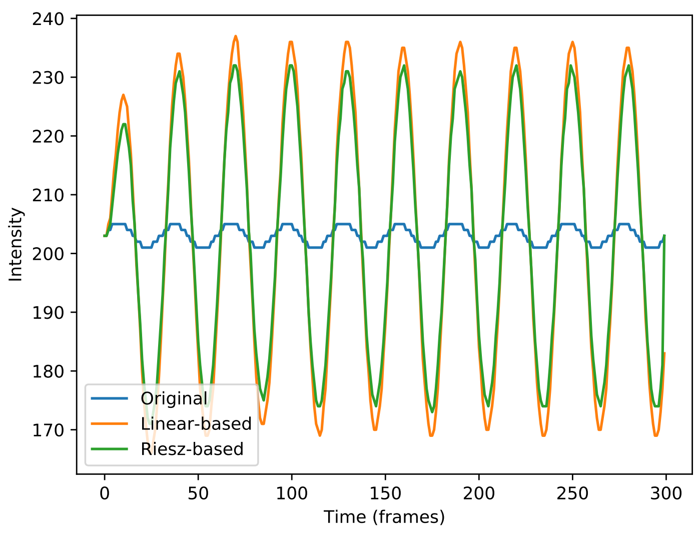
An example of simultaneous color-change and motion magnification is shown in Figure 3.6 for a synthetic video, where the result of magnifying the color changes and motion of the video using the Riesz pyramids is compared to a magnified version generated synthetically by increasing the translation and scaling in the generation process. The original video is composed of a single wave undergoing translation at 1 Hz and scaling and 2 Hz.

3.2 Selection of Regions for Magnification

The magnification methods are in principle applied to all regions in the frames of the video. This leads to the creation of artifacts in regions where it was not desired to modify anything (Figure 3.7 (a)).

Because of the Eulerian character of the magnification methods studied here, one is able to select regions that are to be magnified and not modify the other ones. Elgharib et al. (2015), for instance, have used alpha matte to select a region of interest from user

Figure 3.5 – The value in the center of the video (Figure 3.4) is shown for the source and magnified videos. Both methods magnify the intensity changes. The original video has a step-like form because it is quantized when saved as a file.



Source: the authors.

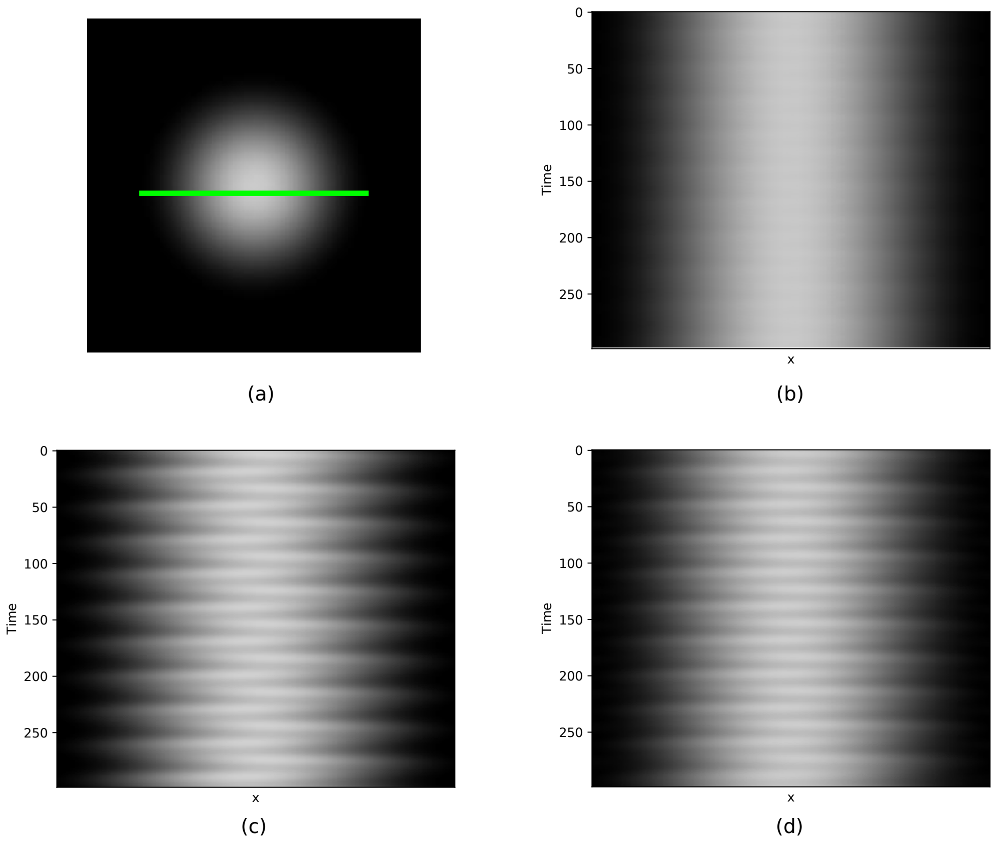
input. We instead introduce a chrominance-based masking for region selection. The user can select a pixel from a frame of the video and the algorithm then magnifies only those regions of the image which have similar chrominance values. The similarity between the chrominance channels of two pixels is measured using the Euclidean distance between them and only pixels with a distance from the select pixel smaller then a certain threshold are magnified. This is illustrated in Figure 3.8.

The mask could be used by first applying the magnification algorithms normally to produce magnified versions of each frame. Then, a masked frame would be constructed such that the pixels whose positions are inside of the region of the mask corresponded to pixels from the magnified frame, while pixels outside the mask corresponded to pixels from the original frame. This could be obtained by applying the following Equation to each frame of the video

$$\hat{I}_M(x, y) = m(x, y)\hat{I}(x, y) + (1 - m(x, y))I(x, y), \tag{3.9}$$

where $I(x, y)$ and $\hat{I}(x, y)$ are the original and magnified frames, $m(x, y) \in \{0, 1\}$ is the mask and $\hat{I}_M(x, y)$ is the masked frame. Instead we have chosen to build a mask that is applied together with the magnification factors α_M and α_C. Hence, instead of multiplying each coefficient in a certain level of the Riesz pyramid by α, we multiply it by $\alpha m(x, y)$. Therefore, it is necessary to have a mask for each level of ther pyramid, since the levels

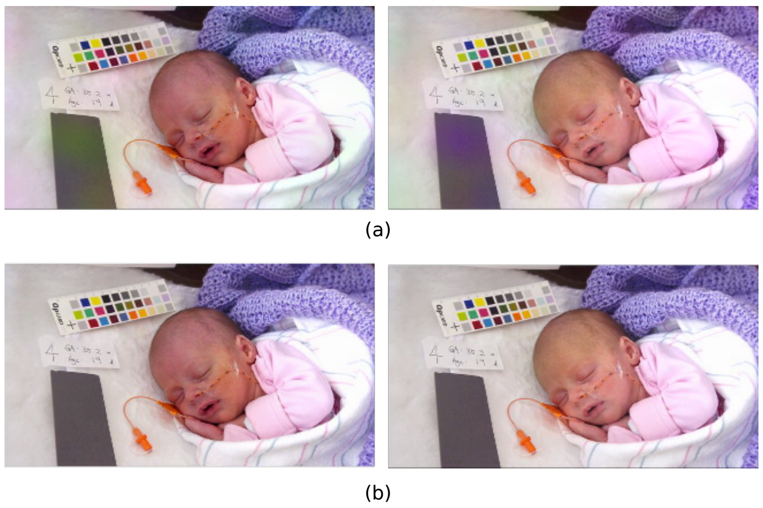
Figure 3.6 – Simultaneous color-change and motion magnification applied to a synthetic video. The first frame of the original video is shown in (a) the time slice in (b). The synthetically generated version undergoing a translation and scaling 10 times greater then the original version is shown in (c), while (d) shows the magnified version using the Riesz pyramids. The results are significantly similar, serving as evidence for the validity of the technique.



Source: the authors.

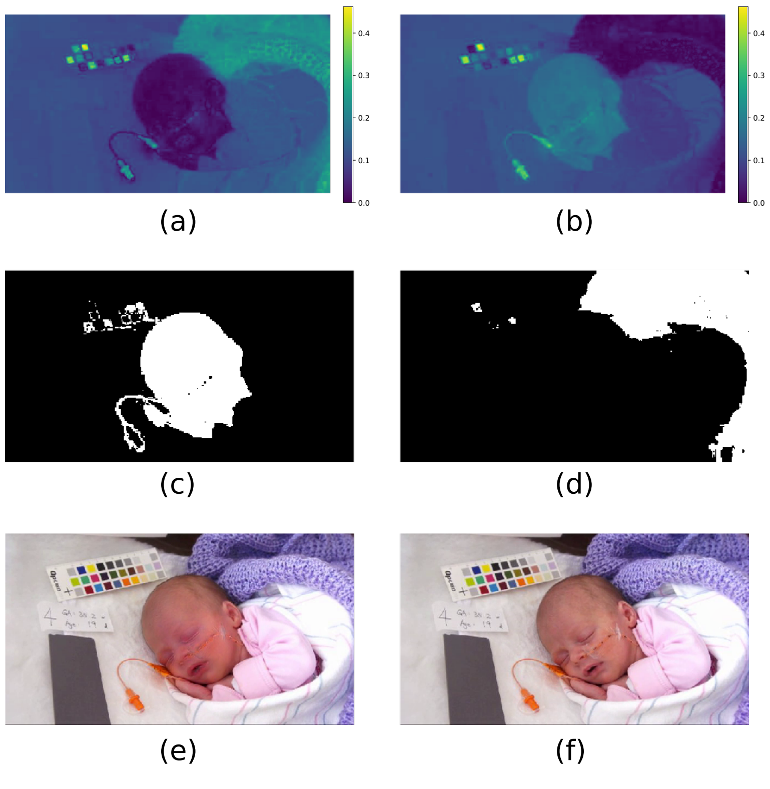
have different dimensions. We have thus constructed a Gaussian pyramid of the mask with the same number of levels as that of the Riesz pyramid. We have found that this produces slightly softer results in the borders of the mask, as illustrated in Figure 3.9.

Figure 3.7 – Applying magnification methods to the whole video leads to artifacts in regions where there is no signal of interest. In (a), we apply color-change magnification without using any mask, which ends up creating artifacts such as those in the black cardboard. In (b), we use our chrominance-based mask to select only the face of the baby.



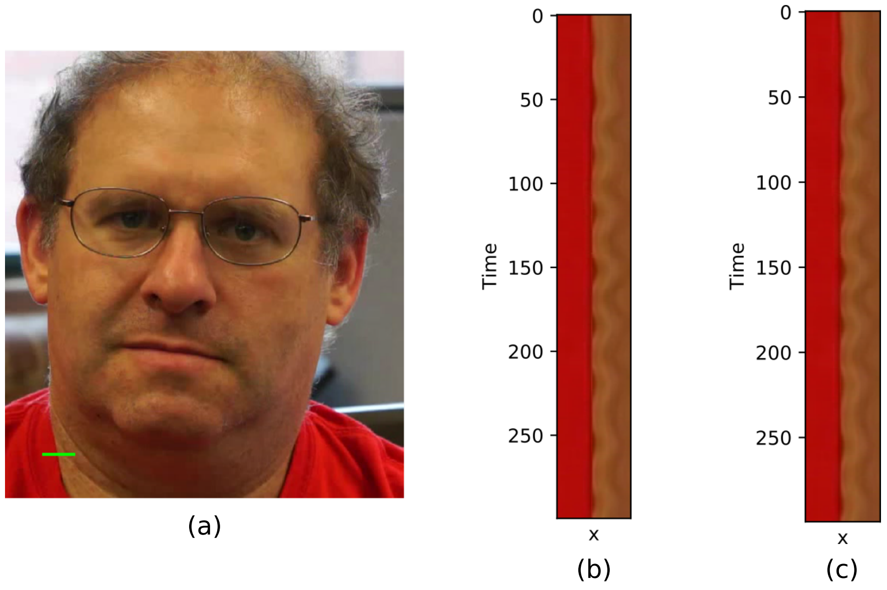
The input video of the baby is from (WU et al., 2012), the figure is ours.

Figure 3.8 – The chrominance-similarity mask allows the selection of regions of interest for magnification. In (a), the chrominance distances between a user-selected pixel in the face of the baby and the other pixels in the image are shown by the heat map, where the darker colors represent smaller distances. In (b), the same process is shown but for a pixel selected in the blanket. (c) and (d) show the masks resulting from (a) and (b), while (e) and (f) show the resulting magnification using the masks.



The input video of the baby is from (WU et al., 2012), the figure is ours.

Figure 3.9 – Comparison between two ways of applying the chrominance-based mask. In (a), the first frame of a video of a face is shown, where the motion arising from the man's heartbeats is magnified. The result of applying the mask only to the magnified frame is shown in (b) and the result of applying a mask at each layer is shown in (c). Notice that in (b) white marks are visible at the borders which are not visible in (c).



Source: the authors.

4 IMPLEMENTATIONS

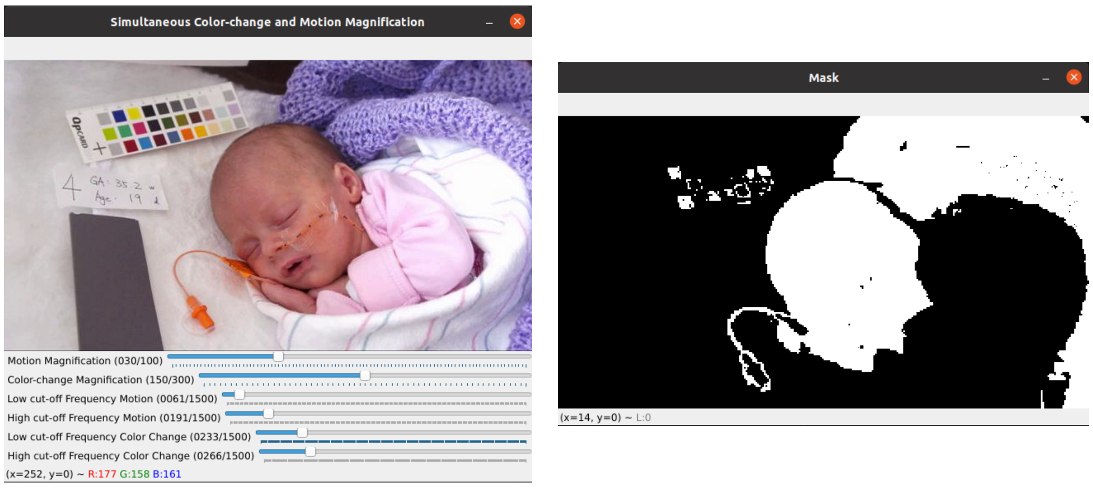
We have implemented our method for simultaneous motion and color-change magnification using Python and we have created a simple graphical user interface for using it. Moreover, we have also created an Android application that implements the Riesz motion magnification (WADHWA et al., 2014a) and the color-change magnification method of Wu et al. (2012). We have not added our implementation to the Android application, as the app was just a proof-of-concept.

In this chapter, we briefly describe our implementations, while detailed explanations for the implementations are given in Appendix C.

4.1 Python Simultaneous Color-change and Motion Magnification

We have implemented our technique using Python and the *scipy* and *OpenCV* libraries. The GUI was built using the user interface components available in OpenCV and it is shown in Figure 4.1. It is possible to select in real time the magnification factors for both color-change and motion magnification. The masks can also be constructed in real time by clicking over the regions of interest to add or remove them. The user can also

Figure 4.1 – GUI implementing the simultaneous color-change and motion magnification with the Riesz pyramids. The current mask selected by the user is also shown to the right.



Motion Magnification (030/100)
Color-change Magnification (150/300)
Low cut-off Frequency Motion (0061/1500)
High cut-off Frequency Motion (0191/1500)
Low cut-off Frequency Color Change (0233/1500)
High cut-off Frequency Color Change (0266/1500)
(x=252, y=0) ~ R:177 G:158 B:161

(x=14, y=0) ~ L:0

Source: the authors.

select different temporal bands of interest for motion and color-change magnification. It would also be possible to change the bands of interest in real-time, however this would necessitate the use of recurrent temporal filters, which introduce delay in the signal of interest because their frequency response has a non-zero phase. Therefore, we have chosen to use the *scipy* implementations of filtering and used either ideal temporal filters which were applied in the frequency domain or applied Butterworth filters forwards and then backwards. Those implementations are also efficient and take around 15s to process a video containing 500 frames of dimensions $352 \times 640 \times 3$ forwards and then backwards using a Butterworth band-pass filter of order 2.

The main class which implements the color-change and motion magnification simultaneously is partially shown in Listing 4.1. The constructor allows for a very flexible magnification process. One can choose specific magnification factors for motion (local phase shift), for the amplitudes of the Riesz pyramid and for its residue. It is also possible to give a list of values for the amplitudes specifying how much amplification is to be done in each level of the pyramid. This is important to allow magnification of the amplitudes only at the more coarse levels of the pyramids. In general, one will want to use the same values for the amplitude magnification factor and for the residue magnification factor.

It is also possible to specify different temporal filters for motion, amplitudes and the residue. A temporal filter here is simply an object containing the method *filter_array(array)*, which filters the given array. One can also attenuate the magnification factor for the chrominance channels. In general, motion magnification performs well with magnification of just the luminance channel, as is done by Wadhwa et al. (2014b).

The constructor also allows the inclusion of an initial mask and whether the ideal Riesz transform should be used (the ideal Riesz transform is applied in the spatial frequency domain, while the approximations are applied in the primal domain). Finally, the *kernel_sigma* parameter specifies the standard deviation of the Gaussian kernel used for the amplitude weighted blur step.

Listing 4.1 – Simultaneous Motion and Color-change Magnification Python Class

```python
class RieszSimultaneousMagnification:
    def __init__(self, num_levels=None, alpha_motion=50,
        alpha_amplitude=150, alpha_residue=150,
        temporal_filter_motion=None, temporal_filter_amplitude=None,
        temporal_filter_residue=None, chr_attenuation_motion=0.0,
        chr_attenuation_amplitude=1.0, chr_attenuation_residue=1.0,
        pixel_mask=None, kernel_sigma=2,
        ideal_riesz_transform=False):
        ...
```

```
4
5       def append_frame(self, frame):
6           ...
7
8       def magnify_video(self):
9           # Compute differences between adjacent quaternionic phases.
10          self.compute_differences(progress)
11
12          # Temporally filter the differences and amplitudes
13          self.apply_temporal_filter()
14
15          # Spatially filter
16          self.apply_amplitude_weighted_blur()
17
18          # Use these differences to magnify frames
19          frames_magnified = self.magnify_frames()
20          return frames_magnified
```

The *append_frame* method (Line 5) simply takes a RGB frame and stores it in the object so it can be later processed. The *magnify_video* (Line 8) magnifies the frames previously appended. It beings by computing the quaternionic phase differences between each frame and the first frame for each coefficient of the Riesz pyramid. This method also computes the amplitudes of each coefficient. Then, the given temporal filters are applied to the the phases, amplitudes and to the residue. If a temporal filter for a particular component of the pyramid is not given, the component is not magnified. For instance, if it is desired to magnify only the color variations, one can pass *None* as the temporal filter for motion. The amplitude weighted blur is also applied before the frames are magnified. The magnification step uses the filtered phases, amplitudes and residue to compute the motion and color-change magnified video.

It is not necessary to apply all these methods at once. Instead, one can compute the phases and amplitudes and apply the temporal and spatial filters as a pre-processing step. Then, each frame can be magnified at a time by calling the method *magnify_frame* instead of *magnify_frames*. The method *magnify_frames* is simply a wrapper which call *magnify_frame* for each frame in the video). This method uses the previously computed values to magnify a single frame. This way, one can change the mask or the magnification factors in real time. That is what we did for our user interface.

4.2 Android Application

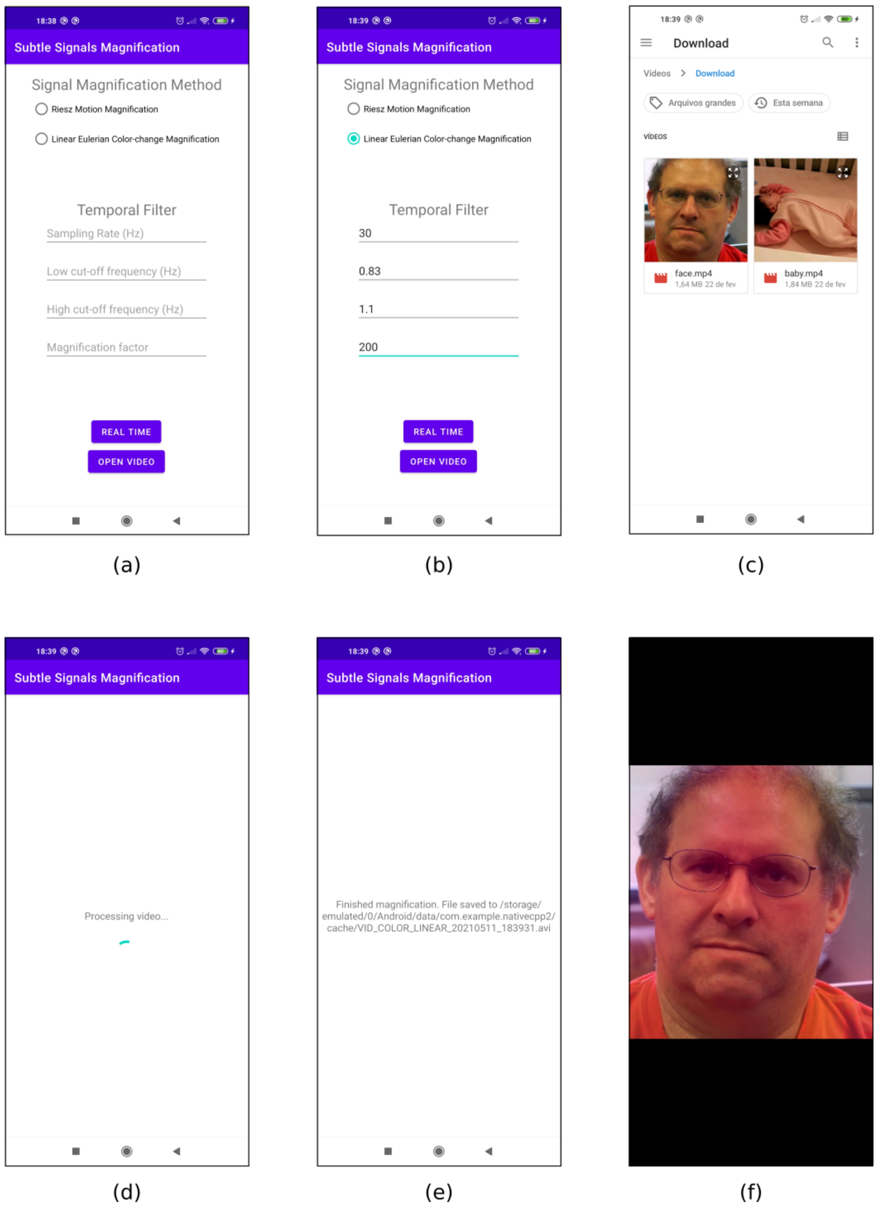
We have implemented the Riesz motion magnification and the linear Eulerian color-change magnification for Android using Java and OpenCV for Java. We have also used the Java Native Interface (JNI) for implementing time-crucial methods using C++. The C++ implementations are compiled by the Android Native Development Kit (NDK) for multiple architectures. Figure 4.2 illustrates the use of our application. The user can select which method is to be used in the magnification process. It is also necessary to specify the sampling rate of the video. This is necessary because this is not always given by the frame rate, as some videos might be recorded using a higher sampling rate but saved with a smaller frame rate so that the high frequency phenomena are visible. The user can also choose the low and high cut-off frequencies of the temporal filter used and the magnification factor. We have used a difference of low-pass first order Butterworths filters for implementing the band-pass filter. This filter does not have very sharp edges at the cut-off frequencies, but it is efficient and can be more easily designed. Finally the user selects between the *real time* and the *open video* options. In the real time case, the camera opens and shows the magnified captured frames. This interface was implemented using boilerplate Java OpenCV code. In the *open video* case, the user selects a video from the cell phone to be magnified and the magnified video is then saved to the device storage.

Both magnification algorithms implementations are based on the abstract class shown in Listing C.1. More details about the implementation are given in the Appendix C.

Listing 4.2 – Subtle Motions Abstract Class

```
public abstract class SubtleSignalMagnification {
   /**
    * Update internal states of the algorithm with given frame
       and generate the magnified version
    * of the given frame.
    * @param frame openCV Mat.
    */
   public abstract void update(Mat frame);
```

Figure 4.2 – Selecting a video for magnification with the Android application. In (a), the main interface of the app is shown. The user can then fills the magnification parameters and select the magnification method (b). By pressing *Open Video*, an interface for selecting a video opens (c). After video is chosen, it processed (d) and the result is stored in the device storage (e). The magnified video can then be seen use a video player in Android (f).



Subtle Signals Magnification

Signal Magnification Method

Riesz Motion Magnification

Linear Eulerian Color-change Magnification

Temporal Filter

Sampling Rate (Hz): 30

Low cut-off frequency (Hz): 0.83

High cut-off frequency (Hz): 1.1

Magnification factor: 200

Processing video...

Finished magnification. File saved to /storage/emulated/0/Android/data/com.example.nativecpp2/cache/VID_COLOR_LINEAR_20210511_183931.avi

Source: the authors.

```

    /**
     * Store in the given Mat the current magnified frame which
        was generated in the update method.
     * @param dst openCV Mat of same dimensions and type as the
        frames given in the update method.
     */
    public abstract void getMagnifiedFrame(Mat dst);
}
```

5 RESULTS

In this section, we show our results from applying the techniques and applications developed here to multiple different videos.

5.1 Simultaneous Color-change and Motion Magnification with Riesz Pyramids

As described in Section 4, we have implemented the described techniques in Python and used them to magnify both motion and color variations on a large number of videos. These methods are computationally efficient and allow the users to obtain simultaneous magnification of these signals interactively. The user can specify the frequency bands (in Hz) for the signals of interest, after which a Riesz pyramid decomposition is obtained and the amplitudes and phases of the pyramid coefficients are computed. Such preprocessing steps takes approximately 21 seconds for a 592×528 RGB video with 300 frames, on a notebook with a i7-10510U @1.80 GHz CPU and 16 GiB of RAM memory. After the preprocessing, the user can interactively change the magnification factors for the selected bands, as well as the frame regions for which the different magnifications should be applied to, receiving instant feedback.

In the results shown here the frames were processed in the YIQ color space, with the Riesz pyramids containing 7 layers plus the residual. We magnify motion using only the luminance (Y) channel and used both luminance and chrominance channels for magnifying color variations (in this case, additional Riesz pyramids for the I and Q channels are created and processed). It is also possible to use only the luminance channel for color magnification, but the inclusion of chrominance channels produce better results. When magnifying color changes, besides the residue, we also magnify the amplitude coefficients from levels 5 up to 7 the Riesz pyramids. We have not used levels 1 to 4 since color variations are generally very weak, being indistinguishable from noise in the lowest levels of the pyramids. Moreover, we have used a Laplacian pyramid as the basis for building the Riesz pyramid using the approximate Riesz transform from (WADHWA et al., 2013). Table 5.1 summarizes the magnification parameters used in our experiments presented here.

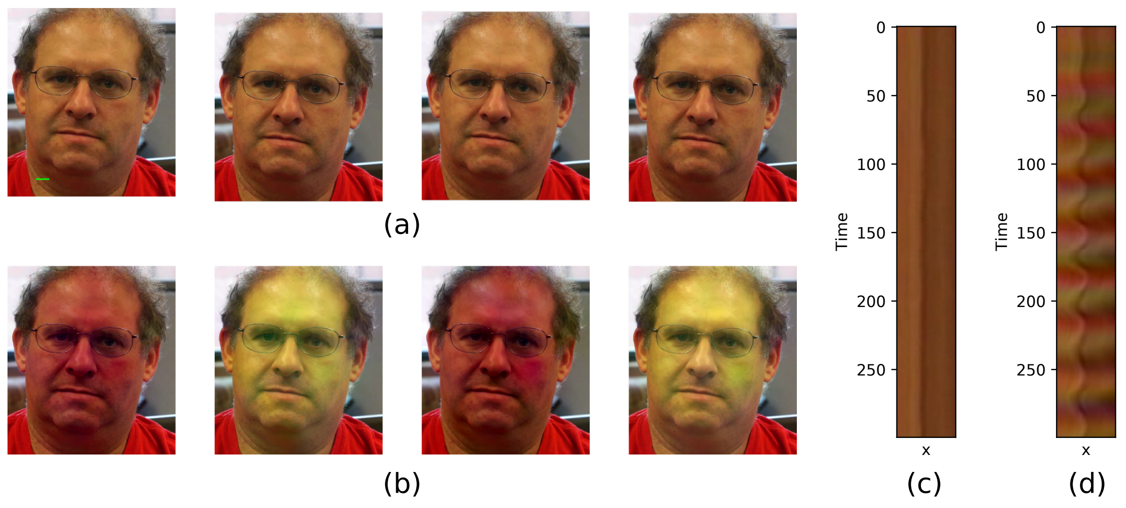
Figure 5.1 illustrates the magnification of both the color variations and motion associated to a man's blood flow.. The time slices reveal specifically how his neck moves as the blood passes through his artery. Furthermore, the heart beats also cause the head to

Table 5.1 – Summary of the magnification parameters used for each video. Kernel sigma is the Gaussian blur sigma used for the amplitude-weighted blur.

Video	α_M	α_C	Motion Temporal Bands (Hz)	Color-change Temporal Bands (Hz)	Kernel Sigma
face heartbeat	30	150	0.83 - 1.10	0.83 - 1.10	4
face2	25	122	0.83 - 1.10	0.83 - 1.10	4
face heartbeat and respiration	12	70	0.20 - 0.33	0.83 - 1.10	4
baby2	10	150	0.61 - 1.91	2.33 - 2.67	4
eye	50	70	30.0 - 40.0	0.83 - 2.00	2
violin	100	-	340 - 370	-	2
drum	5	-	74.0 - 78.0	-	2
guitar	25	-	72.0 - 92.0	-	2
baby	10	-	0.25 - 3.00	-	4
plants	6	-	0.2 - 7.75	-	4

move slightly, as illustrated in Figure 5.2

Figure 5.1 – In (a), the frames from the input video *face* are shown. Their motion and color-change magnified version showing the pulse of the man are shown in (b). (c) and (d) show the time slices over his neck in the original and magnified videos, revealing the color-changes and motion associated with the pulse in his coronary artery.

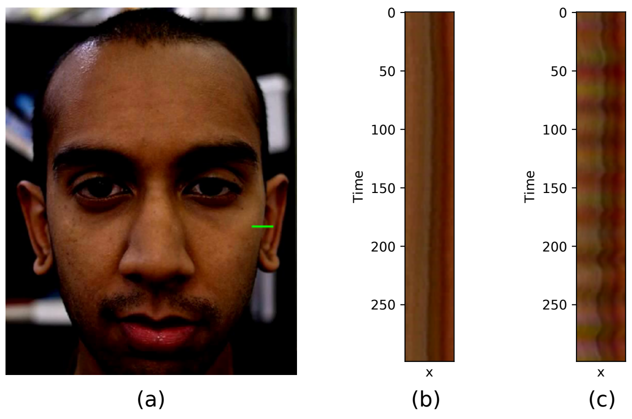


Source: The original video is from Wu et al. (2012) and the Figure is ours.

One can also choose to magnify different signals for motion and color changes by using a band-pass filter for the quaternionic phases and another one for the amplitudes and the residue. In Figure 5.3 we use a band-pass filter with lower cut-off frequencies for the quaternionic phases in order to show the man's respiration moving his head together with the color changes caused by his pulse cycle.

In Figure 5.4 we reveal the heartbeat of the baby through color-changes magnification using a temporal band-pass filter from 140 beats per minute (bpm) to 160 bpm. At

Figure 5.2 – In (a), a reference frame from the source video *face2* is shown. The time slice from the original frame is shown in (b) and the time slice of the magnified video is shown in (c). Here what is shown is the color-changes and the head movement which is associated with the pulse rate of the man.



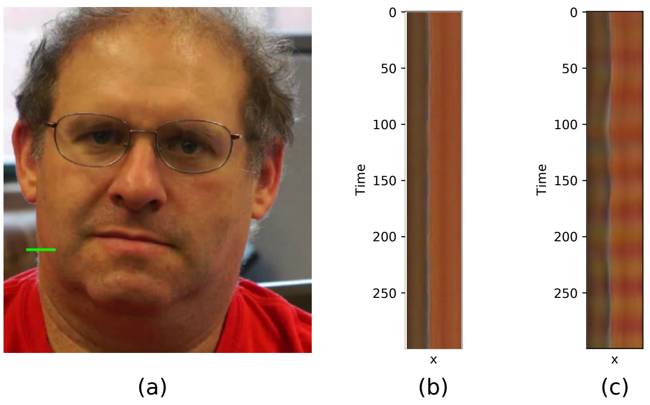
Source: The original video is from Wu et al. (2012) and the Figure is ours.

the same time we magnify the motion related to the the respiration by using a temporal filter for motion which selects the motions between $0.61\ Hz$ and $1.91\ Hz$. In this examples, we have used our chrominance-based mask to magnify only the head of the baby. This results in a video without artifacts in regions which are not of interest.

We can also select signals with different sampling rates and magnify distant frequency bands for motion and color changes. In Figure 5.5, a video captured with a sampling rate of 500 Hz is magnified. The temporal filter for the color changes selects frequencies in the range of the heart rate (between 0.83 and 2 Hz), while the temporal filter for motion selects the band from 30 to 40 Hz, corresponding to microsaccades of the eye. Since our technique extends the Riesz motion magnification (WADHWA et al., 2014a) to also support color, we can also use it for phase-based motion magnification only. In *violin* (Figure 5.6), a video with a sampling rate of $5,600$ Hz has the temporal bands from 340 and 370 magnified, revealing the motion of the bow that plays the strings.

Figure 5.7 (*drum*) illustrates the magnification of motions from 74 to 78 Hz, revealing the vibrations of the skin of a drum. Figure 5.8 (*guitar*) shows the recovery of small vibrations from a specific string of a guitar. Figure 5.9 (*baby*) presents an example of magnification of the periodic motion of a baby's chest during respiration. Finally, Figure 5.10 (*plants*) illustrates the amplified the motions from a plant leaf.

Figure 5.3 – In (a), a reference frame from the source video *face* is shown. The time slice from the original video is shown in (b) and the time slice from the magnified version is shown in (c), where motion associated to the respiration was magnified together with color changes associated with the pulse.



Source: The original video is from Wu et al. (2012) and the Figure is ours.

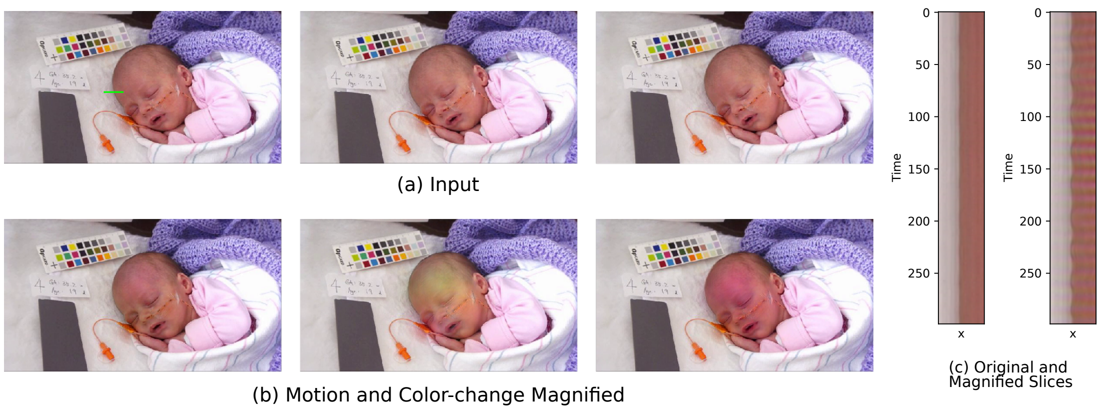
5.1.1 Amplitudes and Residue Magnification

It is possible to use only the residues of the pyramids for magnifying color changes. This is useful in cases where reducing the computational cost of the algorithm is important, since the residues are generally small, with dimensions around 6×6 pixels, depending on the original dimensions of the image and the number of levels used. The problem with this approach, however, is that spatial information will be lost, since the residue is significantly blurred. Figure 5.11 illustrates this idea, where magnifying only the residue is compared to magnifying only the amplitudes (from levels 5 to 7 of a Riesz pyramid containing 8 levels in total) and magnifying both the amplitudes and the residue. Figure 5.11 (c) shows the effect of magnifying only the residue, which produces an almost homogeneous magnification in the whole face. Including the amplitudes (Figure 5.11 (b)) then introduces information which is more localized (the region around the nose, for instance, is less magnified than the cheeks), thus producing a more complete result in Figure 5.11 (d).

5.1.2 Discussion and Limitations

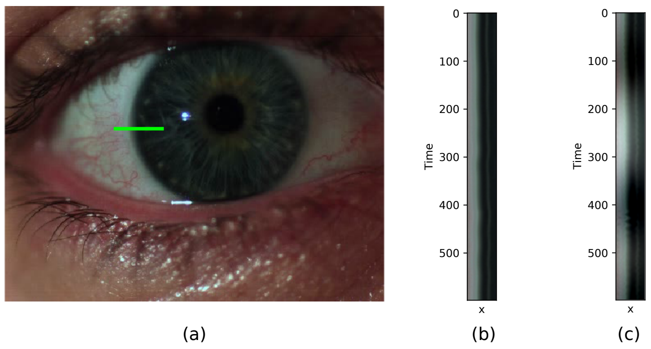
Our technique extends the Riesz motion magnification framework to also support magnification of color variations. Thus, one can obtain phase-based motion magnification and color-change magnification within the same framework. Because the motion

Figure 5.4 – In (a), the frames from the input video *baby2* are shown. Their motion and color-change magnified version is shown in (b). Figure (c) shows the time slices from his head, revealing both the motion associated with the the respiration of the baby and the color changes associated with the pulse.


(a) Input
(b) Motion and Color-change Magnified
Time
0
50
100
150
200
250
x
(c) Original and Magnified Slices

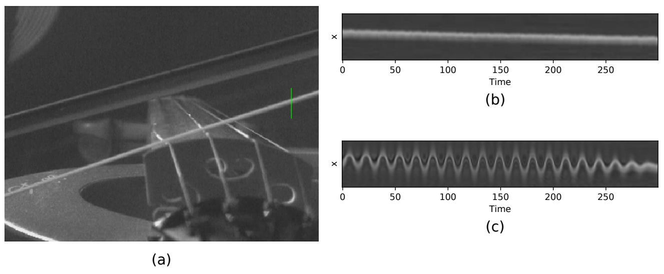
Source: The original video is from Wu et al. (2012).

Figure 5.5 – In (a), a reference frame from the source video *eye* is shown. Figures (b) and (c) show the time slices from the border of the eye in the original and magnified videos, where it is possible to see both the intensity change associated with the heart rate and the microsaccades.



Source: The original video is from Wadhwa et al. (2013).

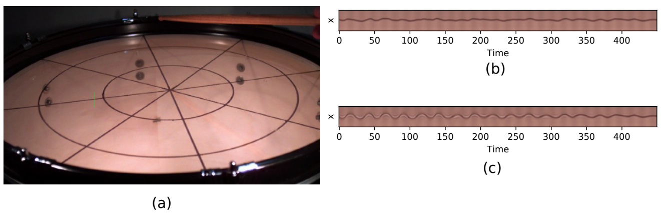
Figure 5.6 – In (a), a reference frame from the source video *violin* is shown. Figures (b) and (c) show original and magnified time slices of the bow playing the violin.



The original video is from Wadhwa et al. (2013) and the Figure is ours.

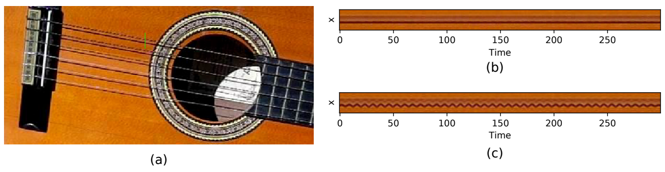
magnification component of our method is based on the Riesz motion magnification technique described by Wadhwa et al. (2014b), both have the same strengths and limitations. Specifically, the Riesz motion magnification method depends on the frames having a sin-

Figure 5.7 – In (a), a reference frame from the source video *drum* is shown. Figures (b) and (c) show original and magnified time slices of the skin of the drum.



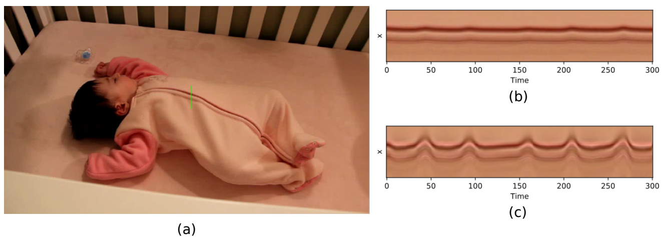
The original video is from Wadhwa et al. (2013) and the Figure is ours.

Figure 5.8 – In (a), a reference frame from the source video *drum* is shown. Figures (b) and (c) show original and magnified time slices of the vibrating string.



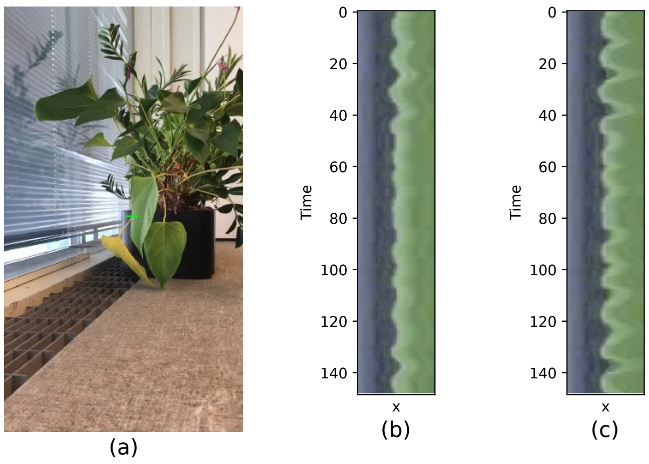
The original video is from Wadhwa et al. (2013) and the Figure is ours.

Figure 5.9 – In (a), a reference frame from the source video *baby* is shown. Figures (b) and (c) show original and magnified time slices of the chest of the baby.



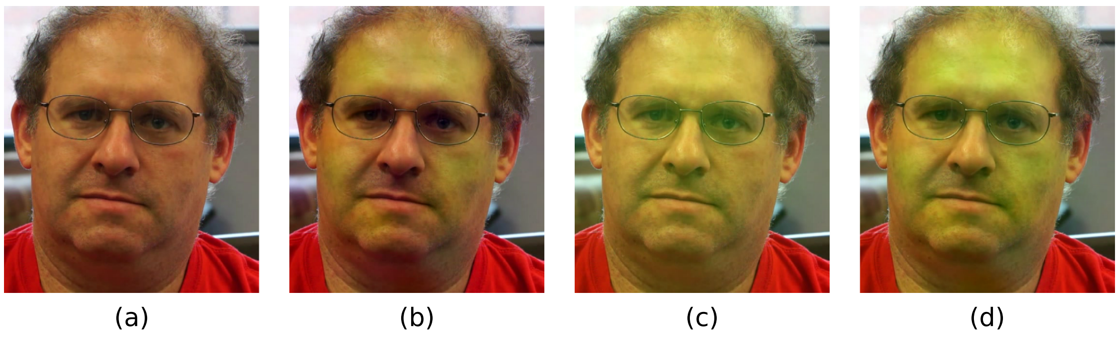
The original video is from Wadhwa et al. (2013) and the Figure is ours.

Figure 5.10 – In (a), a reference frame from the source video *plants* is shown. Figures (b) and (c) show original and magnified time slices of a leaf of the plant.



The original video is from Zhang, Pintea and Gemert (2017) and the Figure is ours.

Figure 5.11 – Comparison between magnifying only the residues versus magnifying the amplitudes as well. The frame from the original video is shown in (a). In (b), the result of magnifying only the amplitudes from levels 5 to 7 is shown. The result of magnifying only the residue is shown in (c) and the result of magnifying both is shown in (d).



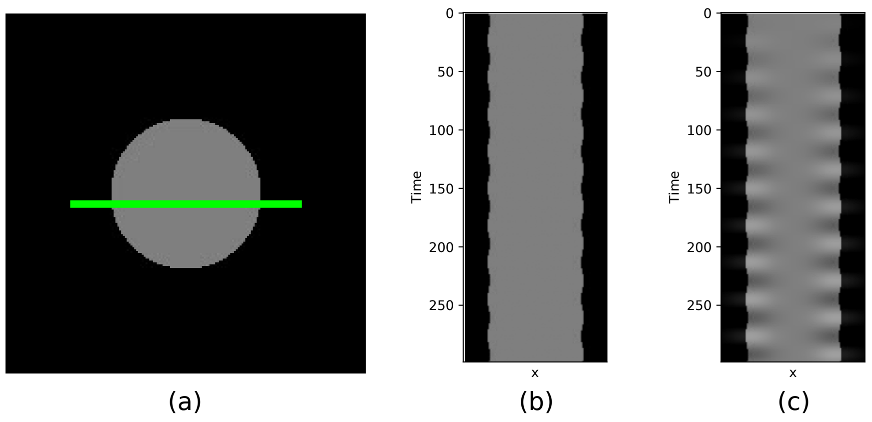
Source: the authors.

gle dominant direction and show problems when this is not the case. However, we have not found this to be a problem on real videos. Furthermore, the Riesz framework uses an approximate Riesz transform for efficiency purposes, which in principle leads to worse estimations of the local phase, causing artifacts. In practice, however, we have not observed any artifacts when compared with the use of the ideal Riesz transform.

To minimize the impact of noise, our technique for color magnification avoids using the finest levels of the spatial decomposition. Ideally, one should magnify all spatial bands for visualizing color changes. However, besides noise issues, this actually also magnifies subtle motions (WU et al., 2012). On the other hand, since we are using the amplitudes of Riesz pyramids and they separate phase changes from amplitude changes – which in an ideal scenario with a single sub-band means that intensity variations and motion are completely separated – our color magnification strategy should not affect motion so much. Nevertheless, since motion and color variations are ultimately variations in the intensity of the pixels, they cannot be completely separated in the magnification process, which means that motion can be magnified when only color changes were to be modified. This is more prone to happen near edges, where the pixel-intensity variations related to motion are more pronounced. Figure 5.12 illustrates a synthetic video of a homogeneous disk undergoing translation and the result of magnifying color changes in it.

Finally, we did not address the problem caused by the presence of larger motions in videos. However, since our method is similar to the previous Eulerian techniques, one can apply any of the approaches for video magnification in the presence of larger motions such as the ones by Elgharib et al. (2015) and Zhang, Pintea and Gemert (2017).

Figure 5.12 – Color-change magnfication applied to a homogeneous disk undergoing translation. The first frame of the video is shown in (a). In (b) and (c) the time slices of the original and magnified videos are shown. Notice that the color-change magnification process magnifies color variations that were actually caused by motion.



Source: the authors.

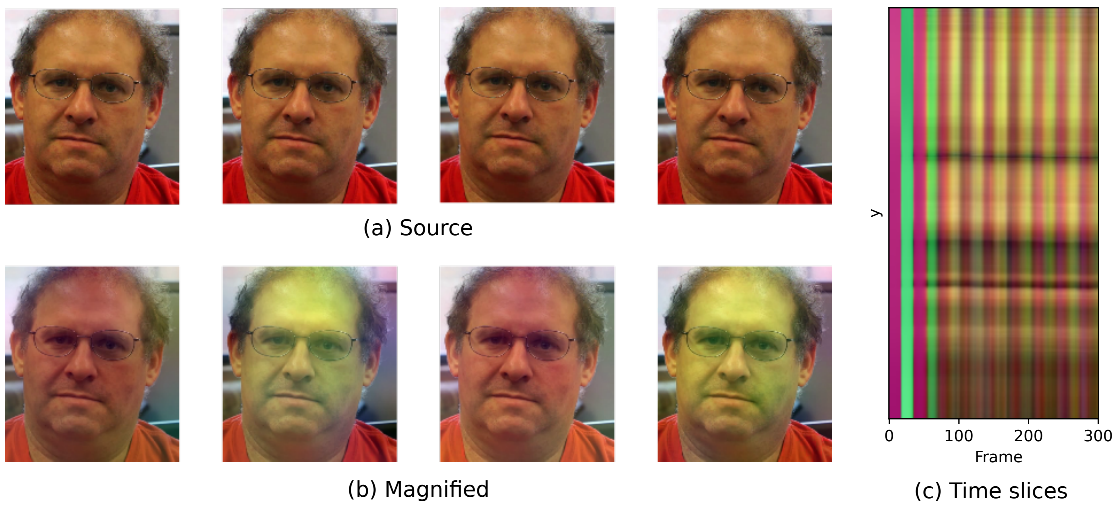
5.2 Android Implementation Results

We have implemented the phase-based Riesz motion magnification (WADHWA et al., 2014b) and the color-change magnification (WU et al., 2012) methods in an Android mobile phone, as described in Section 4.2. Both of the implementations work in the YIQ color space. While the color-change implementation magnified all three channels, the motion magnification is performed only to the luminance channel. Our implementations were tested on a mobile with a MTK Helio G85 processor with up to 2GHz. The implementations of the Riesz motion magnification and the color-change magnification methods take respectively 22.8 seconds and 10.5 seconds to process a video containing 300 RGB frames, each with dimensions 528×592, on the previously mentioned mobile phone.

We illustrate the results obtained using the Android application with two examples, one for color-change magnification and the other one for motion magnification. Figure 5.13 shows the results of magnifying the color changes caused by the heart beats of the man. As can be seen, the temporal filter implemented has ringing artifacts during its beginning. This is a result of the fact that its initial conditions are zero. As such, the video can be thought of as containing a step function at its beginning. Using copies of the initial frame for the initial conditions also cause these ringing artifacts. They could be fixed, however, by choosing appropriate initial conditions.

Figure 5.14 illustrated the magnification of the periodic motion of the chest of the

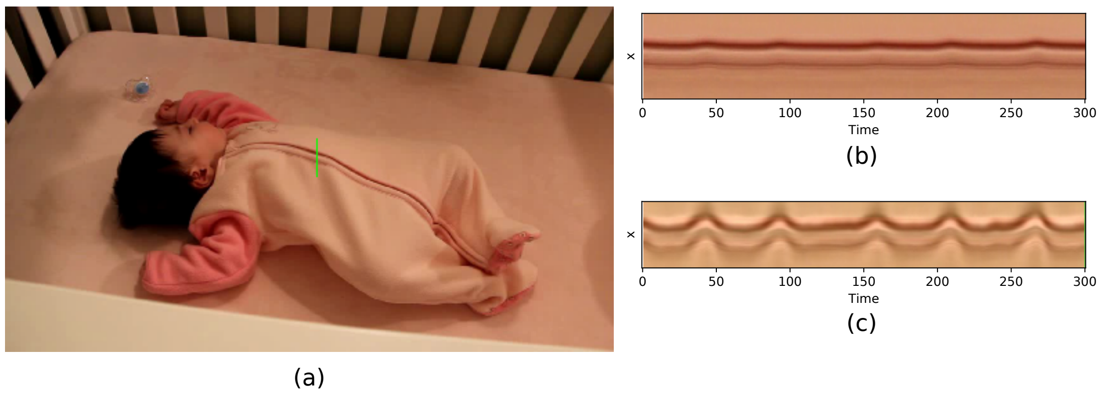
Figure 5.13 – Color-change magnification of the *face* video in the Android application. In (a), the frames from the input video *face* are shown. Their motion and color-change magnified version showing the pulse of the man are shown in (b). (c) shows the evolution of the video over time from a slice in the middle of the face of the man, going from the top to the bottom of the image.



The original video is from Wu et al. (2012) and the Figure is ours.

baby caused by its respiration.

Figure 5.14 – Motion magnification of the *baby* video in the Android application. In (a), the frames from the input video *face* are shown. Their motion and color-change magnified version showing the pulse of the man are shown in (b). (c) shows the evolution of the video over time from a slice in the middle of the face of the man, going from the top to the bottom of the image.



The original video is from Wu et al. (2012) and the Figure is ours.

5.2.1 Discussion and Limitations

As is the case with the Android implementation, our implementations inherit the limitations from the original works on phase-based motion magnification and color-change magnification. In particular, however, our Android implementations also suffer from the problem of ringing in the beginning of the magnified videos. As mentioned previously, this is the result of poorly defined initial values for the temporal filters. Our Python implementation does not suffer from this, as we have used available temporal filter implementations.

Furthermore, the intended real-time implementation only performs in real time for the color-changes magnification algorithm. In that case, it can execute at a rate of approximately 30 frames per second for frames with dimensions 528×592. However, capturing videos in general and processing them in real time implies that the actual frame rate will vary during execution, though not significantly. Since we have used a soft band-pass temporal filter, small changes in the frame rate should not be of great concern. The Riesz motion magnification implementation, on the other hand, can process frames with dimensions 528×592 only at a rate close to 13 frames per second.

6 CONCLUSION AND FURTHER WORK

Videos posses a range of signals which occur in such a small level that cannot be easily seen. In order to recover these signals, multiple techniques have been developed over the years, as we have described. In this work, we have given a step forward in the research of Eulerian video magnification. First, by making a simple Android application showing that it is possible to have these algorithms running in mobile phones. More importantly, we have improved upon the other techniques in order to provide simultaneous phase-based motion and color-change magnification by extending the Riesz pyramids approach which was developed for motion magnification only. This way, it becomes possible to see both signals without having to recur to the application of different algorithms sequentially, which does not always produce good results. Moreover, this also allows the implementation of real time applications for magnification of both signals. We have also created a simple GUI which allows the user to play around with the magnification tool and to mask regions of the video based on the chrominance similarity of the pixels.

Possible directions for future exploration include improving our technique riesz-based motion and color-change magnification technique by adding to it the methods developed for dealing with larger motions (ZHANG; PINTEA; GEMERT, 2017; ELGHARIB et al., 2015; WU et al., 2018a). Furthermore, the orientation of the Riesz coefficients could be used as a parameter for creating directional selective masks. In this case, it would be possible to only magnify motions oriented along a group of intended directions.

Our Android implementations also could be improved in future works. First of all, different temporal filter types could be introduced to allow stronger cut-off frequencies and more appropriate initial conditions could be used in order to avoid the ringing artifacts in the beginning of the magnified videos. A dynamic adaptation of these temporal filters would also be useful for dealing with the varying frame rates in the real time scenarios. Finally, further improvements on the phase-based motion magnification method could be introduced in order to reduce its processing time per frame.

APPENDIX A — TEMPORAL FILTERS

In this Appendix, we review some concepts about temporal filters which we have used for our implementations. This does not aim at being a thorough review of the area of temporal filters, but only those aspects of it which we had to use directly.

A.1 Bilinear Transform

The bilinear is a popular method for converting an analog filter into a digital filter by mapping its s-plane to the z-plane. This mapping can be written as

$$s = \frac{z-1}{z+1} \tag{A.1}$$

and its inverse is given by

$$z = \frac{1+s}{1-s}. \tag{A.2}$$

This is a non-linear mapping from the s-plane to the z-plane which avoids the aliasing problem of impulse invariance. Some important properties of this transformation are listed.

1. The imaginary region of the s-plane is mapped to the region inside the unitary circle in the z-plane, with the imaginary axis being mapped to the unity circle.
2. $s = 0$ is mapped to $z = 1$.
3. $s = j\infty$ and $s = -j\infty$ are mapped to $z = -1$.
4. An analog frequency ω is mapped to $\tan(\theta/2)$.

Property 1 can be seen by writing $z = re^{j\Omega}$ and $s = \sigma + j\omega$ in Equation (A.2) then noticing that the radius r of z is given by

$$|z| = r = \sqrt{\frac{(1+\sigma)^2+\omega^2}{(1-\sigma)^2+\omega^2}}. \tag{A.3}$$

Hence, if $\sigma > 0$ the numeration in Equation A.3 is greater then the denominator and $r > 1$, the reverse is also true. Properties 2 and 3 follow from Equation A.2. Property 4 can also be seen by writing $z = re^{j\Omega}$ and $s = \sigma + j\omega$ and getting the argument of z from

Equation A.2. This gives

$$\theta = \arg\{z\} = \arctan\frac{\omega}{1+\sigma} - \arctan\frac{-\omega}{1-\sigma}. \tag{A.4}$$

The mapping from a certain frequency ω to the digital frequency θ can then be seen by setting $\sigma = 0$ and noticing that $\arctan(-x) = -\arctan(x)$, which gives $\theta = 2\arctan(\omega)$ or $\omega = \tan\frac{\theta}{2}$. An important consequence of this is that if it is desired to design a digital filter with cutoff frequency θ_d, with $\theta_d \in [-\pi, \pi]$, then our original analog filter must have a cutoff frequency given by $\omega_d = \tan\frac{\theta_d}{2}$. Hence, if the goal is to design a digital filter with that will cut off the frequencies above the continuous frequency ω_c for a signal sampled at a sampling rate r, then $\theta_d = \omega_c/r$ and the actual cutoff frequency of the designed analog filter must be $\omega_d = \tan\frac{\omega_c}{2r}$

A.2 Butterworth Filters

The butterworth filters are a widely used class of filters designed to be maximally flat in the pass-band. That is, they are design such that their squared magnitude has very little variation in the pass band. Moreover, the squared magnitude is approximately unity in the pass-band and is small as possible in the stop-bands.

The squared magnitude of the frequency response of a low-pass butterworth filter with cutoff frequency $\omega_c = 1$ is given by

$$|H(j\omega)|^2 = \frac{1}{1+\epsilon^2\omega^{2n}}, \tag{A.5}$$

where n is the order of the filter. If we set $\epsilon = 1$, the cut-off frequency is the frequency at which the energy has decreased by half. The transfer function for this energy response can be found by noticing that $H(s)H(-s)|_{s=j\omega} = |H(j\omega)|^2$ and substituting $\omega = -js$. This gives

$$s = e^{j\pi(2k+1)/(2n)}, \tag{A.6}$$

with $k \in [0, 2n-1]$. In order to have a stable filter, we must choose the poles for $H(s)$ such that they are in the left-hand side of the s-plane, that is, they all have negative real parts. The poles for the filter are therefore

Hence the poles of $H(s)$ can be written as

$$p_k = -\sin\left(\pi\frac{2k-1}{2n}\right) + j\cos\left(\pi\frac{2k-1}{2n}\right), \tag{A.7}$$

now with $k \in [0, n-1]$. The transfer function of a low-pass Butterworth filter of order n with $\omega_c = 1$ can therefore be written as

$$G_n(s) = \prod_{k=0}^{n} \frac{1}{s - p_k}. \tag{A.8}$$

For orders 1 and 2, for instance, this gives $G_1(s) = \frac{1}{s+1}$ and $G_2(s) = \frac{1}{s^2+\sqrt{2}s+1}$.

Low-pass filters with other cut-off frequencies, band-pass filters and high-pass filters can be obtained from the prototype with $\omega_c = 1$ through an appropriate frequency transform. Specifically, low-pass and high-pass Butterworth filters with cut-off frequency ω_c can be computed through the transformations $s \to \frac{s}{\omega_c}$ and $s \to \frac{\omega_c}{s}$, respectively.

A band-pass filter can also be obtained from the low-pass template. In this case, the band-pass is said to be centered at a frequency ω_0, around which $H(\omega) \approx 1$ and is 0 at $\omega = 0$ and $\omega = \infty$. If the bandwidth of the filter is B, then a valid transformation for these requirements is given by

$$s \to \frac{s^2 + \omega_0^2}{Bs}. \tag{A.9}$$

A digital version of these filters can be obtained using the bilinear transform (Appendix A.1):

$$s \to \frac{z-1}{z+1}. \tag{A.10}$$

The non-linearity of the bilinear transform implies that the important frequencies in the design of the filter must be pre-warped before the application of the transform. Specifically, if the goal is to produce a low-pass filter with cut-off frequency ω_c, then the analog filter should be designed to have a cut-off frequency of $\omega_d = \tan\frac{\omega_c}{2r}$, where r is the sampling rate of the signal.

Hence, applying the low-pass to low-pass and the continuous to digital filter transforms consecutevely, we obtain the following transform for converting the low-pass prototype

$$s \to \frac{1}{\alpha}\frac{z-1}{z+1}, \tag{A.11}$$

where $\alpha = \tan\frac{\omega_c}{2r}$.

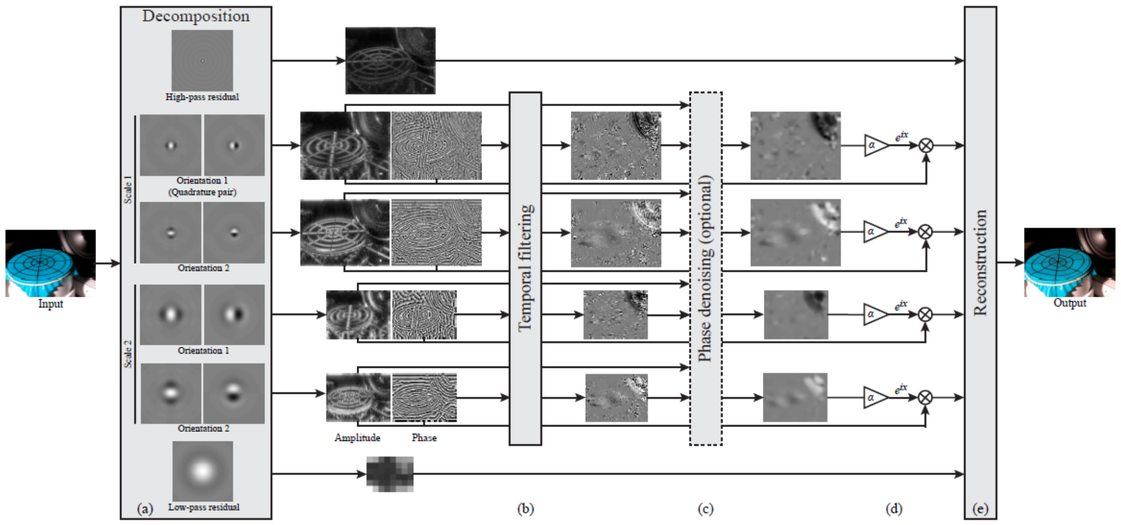
APPENDIX B — OTHER EULERIAN VIDEO MAGNIFICATION METHODS

B.1 Motion Magnification with the Steerable Pyramids

In 2013, Wadhwa et al. (2013) introduced a phase-based Eulerian video motion magnification. The main idea of their work was to decompose each frame into oriented sub-bands containing both local amplitude and local phase information using complex Steerable Pyramids (FREEMAN; ADELSON et al., 1991; PORTILLA; SIMONCELLI, 2000). Inspired by works such as done by Gautama and Van Hulle (2002), which used phase information to estimate the motion field in a video, the authors showed how amplifying only the local phases of a complex steerable pyramid would also magnify the local motions.

Instead of using a Laplacian Pyramid, therefore, the algorithm begins by building the complex steerable pyramid for each frame, which results in a decomposition into oriented sub-bands, each of which containing separate information for local phase and local amplitude. These phases can then be temporally filtered in order to select motions in the chosen frequency band, analogously to what was done in the previous linear Eulerian technique. Finally, adding the modified phases back to the pyramid and collapsing it resulted in the magnified frames. This process is illustrated in Figure B.1.

Figure B.1 – Phase-based Video Motion processing with Steerable Pyramids



Source: (WADHWA et al., 2013)

The relation between the local phase of a steerable pyramid and the displacement can be better understood by first considering the relation between a global phase, obtained through the Fourier decomposition of the image, and the displacement. A 1-dimensional video characterized only displacement is described by $I(x,t) = f(x+\delta(t))$ and the targeted motion magnified version is $f(x+(1+\alpha)\delta(t))$. If we decompose a frame of this video using a complex Fourier Series, an arbitrary band ω of the displaced image could be written as

$$S_\omega(x+\delta(t)) = A_\omega e^{i\omega(x+\delta(t))}.$$

The phase of this component is given by $\omega(x+\delta(t))$. Temporally filtering the phase at each x such as to remove the DC component results in $B_\omega(x,t) = \omega\delta(t)$. Multiplying this signal by a magnification parameter α, it can be added back to the phase in the form of a complex exponential in order to give the band of the motion magnified signal

$$\hat{S}_\omega(x,t) = S_\omega\ e^{i\alpha B_\omega(x,t)} = A_\omega e^{i\omega(x+(1+\alpha)\delta(t))}.$$

This equation implies that by modifying the phase of a pixel x in a given band we obtain a new value for the pixel which is equivalent to that if the band had been translated. Hence, if all the pixels participating in a certain motion have their phases modified accordingly, the motion will also be translated. Instead of using a temporal filter that only removes the DC signal, using a band-pass temporal filter is also possible. In this case, we can modify each band independently in order to select only the motions that fall within the specified frequency range. The mathematical development for the band-pass filter is analogous to that in 2.1 and is omitted here.

However, using Fourier basis for the decomposition would not work, because their spatial support occupies the whole image. Hence, different features would be contained in the same sub-band and hence if one of them moves, but the others don't, we would have displacement in only one region of the band and the phase change over time would not correspond simply to any the motion. That is, the phase has to rearrange itself in order to represent both a region that has moved and a region that has remained static. This is the reason for using the complex Steerable Pyramids, since they have a finite impulse response and therefore we could expect its filters to capture approximately only one image feature. Moreover, the fact that the sub-bands are oriented is also necessary. If, instead, we had non-oriented sub-bands (a Laplacian Pyramid, for instance), and used only the positive frequencies in order to generate a complex signal with a phase in the spatial domain, then there is no underlining idea of a wave and no meaning for that phase. In the Fourier decomposition, where there is a single point in frequency for each band – that is, each band is a single wave–, the underlining idea of a wave is made perfect, while in the Steerable Pyramids it is made approximate.

In order to derive bounds for the amount of magnification possible with the Steerable Pyramids, Wadhwa et al. (2013) model the Steerable filters analytically as a Gabor filter and consider a Dirac function under translation. The frequency response of a Gabor filter is a Gaussian centered at some frequency f_c and is given by $e^{-2\pi(f_x-f_c)^2\sigma^2}$. Its impulse response is therefore (up to a constant)

$$S_{f_c}(x,0) = e^{-x^2/(2\sigma^2)}e^{2\pi i f_0 x}. \tag{B.1}$$

If the Dirac translates, it results in

$$S_{f_c}(x,0) = \exp\left(-(x-\delta(t))^2/(2\sigma^2)\right)\exp\left(2\pi i f_0(x-\delta(t))\right). \tag{B.2}$$

Filtering the phase thus results in $B_{f_c}(x,t) = 2\pi f_c \delta(t)$. If we magnify it and add it back to the band through a complex exponential, we will translate the underlining complex wave but *not* the Gaussian envelope multiplying it. Hence, the wave will be attenuated and distorted. The authors choose therefore to bound the magnification such that the additional displacement ($\alpha\delta(t)$ pixels further than the non-magnified motion) must be below one standard deviation of the Gaussian envelope. Moreover, in the Steerable Pyramid there is approximately one period of the sinusoid under the Gaussian envelope, which means $\frac{1}{f_c} \approx 4\sigma$ and $\frac{1}{f_c} = \lambda$. Therefore,

$$\alpha\delta(t) < \frac{\lambda}{4},$$

which is 2 times greater then the possible magnification factor using the linear Eulerian method of Wu et al. (2012).

Widening the Gaussian can increase the allowable magnification. In order to do this, Wadhwa et al. (2013) also develop sub-octave pyramids, which have more than one band per octave of the frequency domain. However, widening the Gaussians comes at the cost of increased computational complexity, besides the problem of joining features with different motions.

The method developed by Wadhwa et al. (2013) (and phase-based approaches in general) are more resistant against noise then the method by Wu et al. (2012), since by modifying the phases of the bands we are only translating the noise and not amplifying it. Nevertheless, the phase obtained from the pyramids will be noisy if the input video is noisy. This problem can be reduced by spatially filtering the phases. Therefore, Wadhwa et al. (2013) also apply an amplitude-weighted Gaussian filter, so as to ignore low-amplitude points in the sub-band, whose phase values are probably irrelevant. For the i-th band and k-th frame with a Gaussian kernel K_ρ, where ρ is the kernel standard deviation, blurring the local phase $\phi_{i,k}$ takes the form

$$\frac{\phi_{i,k} A_{i,k} * K_\rho}{A_{i,k} * K_\rho}.$$

It was mentioned before that widening the Gaussian envelope also increases the computational complexity. In fact, building the Steerable Pyramid is much more computationally expensive than the Laplacian Pyramid from the linear Eulerian magnification algorithm, because the Steerable Pyramid is significantly more over-complete. The over completeness of the pyramid with respect to the Laplacian pyramid is given by $2k/(1 - 2^{-2/n})$, for k orientations and n filters per octave.

Wadhwa et al. (2013) also notice that motion can be attenuated by using $\alpha = -1$. This, however, is not the same as making the video completely static, since the envelope of the feature still moves. It does however produces an effect of reduced motion.

In the case of large motions, the model of temporal filtering does not capture the motion correctly, since the motion will be larger than the spatial support of the filter, and the value of the local phases will be meaningless. The authors avoid amplifying the phases in these situations by not amplifying the pixels where the phase differences are higher than a certain threshold.

Wadhwa et al. (2013) have therefore described a phase-based algorithm for motion magnification or attenuation. The phases are obtained from the complex coefficients of the oriented filters of a complex steerable pyramid. Those filters look like sinusoids attenuated by a Gaussian curve, which, together with the fact that they have an orientation, implies that the phase can be seen as an approximation for the phase of a Fourier base. Temporally filtering the phases then isolates the motion. Moreover, the temporal filter can be chosen such as to isolate only the desired frequencies of motion, in a similar way as with the Linear method, thus allowing frequency-based motion selection.

B.1.1 More Details on the Meaning of the Local Phases

In frequency, the real steerable pyramid filters are like Gaussian curves across the radial frequency and also across the angular frequency (they are not really built with Gaussians, but it is easier to imagine them like that). Those Gaussians are centered at some radial frequency and also at some angular frequency (they are 2-dimensional Gaussian in polar coordinates). That is, they are not in the center of the spectrum. In frequency, this means that they are band-passing a set of adjacent radial frequencies and orientations. A Gaussian in frequency is also a Gaussian in space. Moreover, translating a base-band Gaussian to f_c and $-f_c$ is equivalent to multiplying this Gaussian by a cosine with frequency f_c in space. Hence, the real steerable pyramid filters are like cosines attenuated by a Gaussian.

The complex steerable pyramid, in turn, does not have the $-f_c$ band (similarly to how the analytic signal would be built in one dimension). Therefore, its corresponding Gaussian in space will be multiplied only by $e^{i2\pi f_c t}$. Therefore, the complex steerable pyramid filters are complex sinusoids attenuated by Gaussians. These complex sinusoids have the same instantaneous phase as that of their real counterparts. Moreover, the instantaneous phases of the real counterparts correspond approximately to the perfect sinusoid with frequency f_c, as the filter in frequency is centered around it. Thus, the real steerable pyramid afford a way to get a space localized measure of the amplitude and phase of a given wave f_c. The complex steerable pyramid in turn provides an easy way to get access to the instantaneous phase $2\pi f_c x + \phi$

B.2 Quaternionic Representation of the Riesz Pyramid for Video Magnification

In Wadhwa et al. (2014a), the authors show how Equation (2.24) develops naturally given a quaternionic representation of the Riesz pyramids coefficients signal. This Appendix gives some information on this representation.

B.2.1 Quaternions

The quaternions are a generalization of the complex numbers containing three imaginary units. A quaternion q is written

$$q = q_0 + iq_1 + jq_2 + kq_3, \tag{B.3}$$

where the imaginary units satisfy

$$i^2 = j^2 = k^2 = ijk = -1. \tag{B.4}$$

From Equation B.4, it follows that

$$\begin{aligned} ij &= k = -ji \\ jk &= i = -kj \\ ki &= j = -ik. \end{aligned}$$

The sum of two quaternions p and q is given by

$$p + q = (p_0 + q_0) + i(p_1 + q_1) + j(p_2 + q_2) + k(p_3 + q_3).$$

The product $pq = (p_0 + ip_1 + jp_2 + kp_3)(q_0 + iq_1 + jq_2 + kq_3)$ between two quaternions however is non-commutative and is given by

$$\begin{aligned} pq = p_0q_0 - (p_1q_1 + p_2q_2 + p_3q_3) + p_0(iq_1 + jq_2 + kq_3) + q_0(ip_1 + jp_2 + kp_3) \\ + i(p_2q_3 - p_3q_2) + j(p_3q_1 - p_1q_3) + k(p_1q_2 - p_2q_1) \end{aligned}$$

or $pq = p_0q_0 - \boldsymbol{p} \cdot \boldsymbol{q} + p_0\boldsymbol{q} + q_0\boldsymbol{p} + \boldsymbol{p} \times \boldsymbol{q}$, where $\boldsymbol{q} = iq_1 + jq_2 + kq_3$ and $\boldsymbol{p} = ip_1 + jp_2 + kp_3$ are the imaginary parts of the quaternions. The conjugate, norm and inverse of a quaternion q are given by the equations

$$q^* = q_0 - iq_1 - jq_2 - kq_3, \tag{B.5}$$

$$|q| = q^*q = \sqrt{q_0^2 + q_1^2 + q_2^2 + q_3^2}, \tag{B.6}$$

$$q^{-1} = \frac{q^*}{|q|}. \tag{B.7}$$

The exponential of a quaternion is defined using its Taylor Series,

$$e^q = \sum_{n=0}^{\infty} \frac{q^n}{n!} = e^{q_0} \left(\cos|\boldsymbol{q}| + \frac{\boldsymbol{q}}{|\boldsymbol{q}|} \sin|\boldsymbol{q}| \right). \tag{B.8}$$

and its logarithm is given by

$$\ln q = \ln|q| + \frac{\boldsymbol{q}}{|\boldsymbol{q}|} \arccos\frac{q_0}{|q|}. \tag{B.9}$$

An arbitrary quaternion q can also by written in an exponential (polar) form as

$$q = |q|\frac{q}{|q|} \tag{B.10}$$

$$= |q|\left(\cos\theta + \frac{\boldsymbol{q}}{|\boldsymbol{q}|}\sin\theta\right) \tag{B.11}$$

$$= |q|\exp\left(\frac{\boldsymbol{q}}{|\boldsymbol{q}|}\theta\right), \tag{B.12}$$

where $\cos\theta = \frac{q_0}{|q|}$ and $\sin\theta = \frac{\boldsymbol{q}}{|q|}$.

B.2.2 Riesz Motion Magnification with Quaternions

Instead of using the vector representation of the Riesz pyramid coefficients in Equation 2.12, Wadhwa et al. (2014a) write them in a quaternionic form as

$$\mathbf{r} = I + iR_1 + jR_2. \tag{B.13}$$

where $\mathbf{r}(x, y)$ is an arbitrary coefficient of the Riesz pyramid and I, R_1 and R_2 are defined as in Section 2.2.1. Similarly as before, this can also be expressed in terms of the local phase $\phi(x, y)$, local orientation $\theta(x, y)$ and local amplitude $A(x, y)$, resulting in

$$\mathbf{r} = A\cos(\phi) + iA\sin(\phi)\cos(\theta) + jA\sin(\phi)\sin(\theta). \tag{B.14}$$

When working with complex numbers, the phase can be computed using the logarithm of the normalized complex number. Here, similarly, Wadhwa et al. (2014a) compute the *quaternionic phase* using the logarithm of the normalized quaternion. The norm $||\mathbf{r}||$ is given by A and the quaternionic phase is therefore

$$\log\left(\frac{\mathbf{r}}{||\mathbf{r}||}\right) = \frac{\mathbf{v}}{||\mathbf{v}||}\arccos(q_0) = i\phi\cos(\theta) + j\phi\sin(\theta), \tag{B.15}$$

where $\mathbf{v} = i\phi\cos(\theta) + j\phi\sin(\theta)$ and q_0 is the real part of $\mathbf{r}$. This is the same quantity that was used in 2.24 for filtering in place of the phase and it has no discontinuities in the phase arising from whether orientation is represented by θ or by $\theta + \pi$.

However, there are still wrapping problems occasioned by the fact that ϕ corresponds to $\phi + 2\pi$. The authors solve this by unwrapping the quaternionic phase. This is performed first taking the difference of the phases and converting them to their principle values. That is, the difference is chosen such that it is always in $[-\pi, \pi)$. (Notice that this assumes that the phase never changes by more than π after each frame.) Then the principle values of the differences are summed up to get the unwrapped phases. After the unwrapping, time filtering and spatial filtering, one ends up with a value analogous to that in Equation 2.24, given by

$$i\phi'' \cos(\theta) + j\phi'' \sin(\theta), \tag{B.16}$$

where $\phi'' = \frac{A\phi * K_\rho}{A * K_\rho}$. This is the local quanternionic phase already spatially smoothed and temporally filtered to select the desired motion frequencies.

Finally, this value is magnified by multiplication by a parameter α and added back to the pyramid level by exponentiating it (according to Equation B.8) and multiplying it by the monogenic signal of the sub-band currently being modified. The magnified sub-band corresponds to the real part of the resulting magnified monogenic signals. Therefore, the pyramid composed of the real parts of the Riesz pyramid is collapsed in order to obtain the magnified frame.

B.3 Subtle Motion Magnification in the Presence of Large Motions

A major drawback in the video magnification techniques is the fact that larger variations will influence the subtle signal amplification process and distort the magnified video. In this Appendix, we describe how this problem was taken care of by different authors.

B.3.1 Dynamic Video Magnification (DVMAG)

Elgharib et al. (2015) propose a solution for dealing with larger displacements in videos that is based on temporally stabilizing a user defined region of the video and in using matting in order to magnify only the foreground and the alpha matte. The stabilization is performed by registering each frame to a reference frame, such as to remove the large motions of the region of interest. This registration step uses feature points which are acquired either using KLT tracks (Jianbo Shi; Tomasi, 1994) or optical flow. Having the larger motion removed, a given region of interest is decomposed using alpha matting and the foreground and opacity matte are magnified using either the linear Eulerian technique (WU et al., 2012), in the case of color variations, or the steerable pyramids phase-based method (WADHWA et al., 2013). Finally, each frame is de-warped using the same motion estimations obtained during the registration step.

Their algorithm allowed video magnification to be used even in the presence of large motions. Their results, however, are dependent on user input. Moreover, the warping used in the registration step is specified to be either affine or translation-only, which limits the array of possible larger motions that can be accounted for. It is not able to handle, for instance, different large motions on the object of interest.

B.3.2 Acceleration Magnification

Another approach for magnifying subtle signals in the presence of larger changes was proposed by Zhang, Pintea and Gemert (2017). Their idea was to magnify only the deviations from the changes, instead of magnifying all changes within a certain temporal band as done by Wu et al. (2012) and Wadhwa et al. (2013). That is, their temporal filter was based on magnifying the second temporal derivative of the pixels of the video: the acceleration. If we assume that the larger variations are almost linear with respect to time, then magnifying only those terms that remain after a second order derivative will not alter the larger motions or intensity variations.

In order to obtain a second derivative of the video and still maintain the frequency-based change selection property, the authors use a Laplacian of Gaussian as their temporal filter. Specifically, the standard deviation σ of the Gaussian filter is chosen in order to select for a certain frequency. Moreover, the linearity property of the Gaussian filter and the second derivative is used so that the Laplacian of Gaussian can be applied instead of directly taking the second derivative of the video. That is,

$$\frac{\partial^2 I(x,t)}{\partial t^2} * G_\sigma(t) = I(x,t) * \frac{\partial^2 G_\sigma(t)}{\partial t^2}. \tag{B.17}$$

The authors also mention that they use a temporal window of size $\frac{r}{4\omega}$, where ω is the desired frequency and r is the frame rate. Moreover, the value of σ is chosen to be $\sigma = \frac{r}{4\omega\sqrt{2}}$ and the filter is centered at the current frame being filtered.

The procedure also differs depending on whether it is desired to magnify color changes or motion, but both processes occur in the YIQ color space. In the case of color changes, the procedure is similar to the linear Eulerian color-change magnification described in Section 2.1. On the other hand, motion magnification uses the Steerable Pyramid (Appendix B.1). In this case, the Laplacian of Gaussian filter is applied to the phases of the pyramid.

The algorithm proposed by Zhang, Pintea and Gemert (2017) was able to achieve similar results to those of Elgharib et al. (2015) without requiring user input. Nevertheless, it assumed that the larger signals are approximately linear with respect to time. If non-linear motion or motions with multiple different speeds are present, which is often the case, then magnification will cause blurring.

B.4 Deep Learning Approaches

All the preceding techniques rely on hand-designed filters and neglect non-idealities such as occlusion in the video. Oh et al. (2018) argue that this makes them prone to noise and other artifacts. Therefore, they propose a new Eulerian video magnification technique that learns the filters for isolating the small motions. Specifically, the authors use CNNs and Residual Blocks in the three major steps of motion processing algorithms in general: an encoder network that decomposes the image into a useful representation for motion processing, a manipulator that manipulates this representation such as to magnify the motion, and finally a decoder that synthesizes a motion-modulated frame from the modified representation. These networks are trained on a synthetic dataset generated by the authors containing sequences of frames representing small motions.

Their results proved to be an improvement over the previous techniques, by reducing noise and artifacts. However, their networks give motion representations which are approximately linear only for small amplification factors. This means that the use of temporal filtering for motion selection is restricted. Moreover, deep convolutional neural networks can be computationally expensive which limits its use in real-time applications.

Chen and McDuff (2018) proposed a new method for video magnification which was based on applying gradient ascent in Convolutional Neural Networks (CNN). Their technique required a CNN trained to extract the signal of interest from an input video. For instance, if the goal was to magnify the facial color changes caused by the cardiac cycle, then a CNN needs to be trained to extract the cardiac cycle from videos of faces. The authors then fix the CNN weights and update each frame of the input video such as to increase the L2 norm of the output of the network. This maximization process is done through multiple iterations of gradient ascent. The result is a modified video that when passed through the CNN produces an amplified version of the signal of interest.

Although the proposed framework is general, the authors tested it in the tasks of extracting facial color changes related to the heart rate and in body motions associated with respiration. In both tasks, their results are comparable to the earlier approaches. Moreover, their technique is robust against a wider range of larger motions. When magnifying facial color changes, for instance, the algorithm performs well even when the subject is rotating its face. Nevertheless, the procedure can only be applied in tasks where it is possible to find a labeled dataset for extracting the signal of interest, which might restrict its applicability, and the use of CNN also imposes constraints for real-time applications.

APPENDIX C — IMPLEMENTATIONS

We have implemented both a Python application which performs the simultaneous motion and color-change magnification and an Android application which independently magnifies motion and color changes using different methods. In this Appendix, we describe in more details the Android implementations. We do not give a detailed description of the Python implementation, first because many parts of it are very similar to the Android implementations, only adapted to Python, second because the simultaneous motion and color-change magnification theme is more interesting for its conceptual development, described in the main part of this document, and not for its strict implementation in Python. Finally, because we do not wish to write these descriptions, as this is of little value compared to the effort it would take.

C.1 Android Implementations

The Android implementations of the magnification algorithms are based on the abstract class shown in Listing C.1 (we repeat the Listing shown earlier for convenience). The *update* method should take an OpenCV matrix as the input frame, whose type is not specified in this context, and update its states such that a magnified version of the same frame is available. This magnified version can then be recovered through the *getMagnifiedFrame* method.

Listing C.1 – Subtle Motions Abstract Class

```
public abstract class SubtleSignalMagnification {
   /**
    * Update internal states of the algorithm with given frame
      and generate the magnified version
    * of the given frame.
    * @param frame openCV Mat.
    */
   public abstract void update(Mat frame);

   /**
    * Store in the given Mat the current magnified frame which
      was generated in the update method.
```

```
     * @param dst openCV Mat of same dimensions and type as the
        frames given in the update method.
     */
    public abstract void getMagnifiedFrame(Mat dst);
}
```

We've also used an abstract class for the temporal filters, thus adding the possibility for later adding new types of band-pass filters. The class is shown in Listing C.2. The only requirement for the temporal filters is to have the *filterFrame* public method, which filters a given input OpenCV matrix and stores the result in the output matrix of same type and dimensions. A temporal filter implementing this class should therefore be a recursive filter, as it must output the magnified frame before all the input frames are given.

Listing C.2 – Temporal Filters Abstract Class

```
public abstract class TemporalFilter {

    /**
     * Filter Mat in frameSrc, update the internal states of the
        filter and store the resulting filtered Mat in frameDst.
     * @param frameSrc input frame
     * @param frameDst output frame of the same size and type as
        frameSrc
     */
    public abstract void filterFrame(Mat frameSrc, Mat frameDst);
}
```

Moreover, the magnification algorithms will in general apply temporal filters in many diferents matrices whose sizes and quantity is not known beforehand. For instance, in the Riesz magnification method, it is necessary to filter the quaternionic phases in all the levels of a Riesz pyramid. The number of levels, however, is defined inside the magnification method itself. Therefore, the magnification algorithms need to have the ability to create the temporal filters themselves. In order to allow the filter properties to be defined isolated from the magnification method, while still allowing for the magnification to create the filter objects when needed, we have also introduced the *TemporalFilterBuilder* class. The more relevant parts of this class are shown in Listing C.3. The filter parameters such as its cut-off frequencies and its type are specified in the constructor. The temporal filter itself, however, is only built by the *build* method, with the appropriate dimensions and OpenCV type. A temporal filter builder object can then be passed to the magnification algorithm which then uses it to build the filters only when needed.

Listing C.3 – Temporal Filter Builder class

```java
public class TemporalFilterBuilder {
   public enum Type {
      DIFF_BUTTER_1
   }

   ...

   /**
    * Specify the parameters of the temporal filter that should
       be built by the build method.
    *
    * @param type type of temporal filter
    * @param frameRate sampling rate of the signal to be filtered
    * @param freqLow low cut-off frequency of the filter
    * @param freqHigh high cut-off frequency of the filter
    */
   public TemporalFilterBuilder(Type type, double frameRate,
      double freqLow, double freqHigh) {
      ...
   }

   /**
    * Builds a temporal filter with the parameters specified in
       the constructor and with the given dimensions and type.
    * @param size size of the frames to be filtered
    * @param cvType type of the frames to be filtered
    * @return temporal filter
    */
   public TemporalFilter build(Size size, int cvType) {
      if (this.type == Type.DIFF_BUTTER_1) {
         return new DifferenceOfButterworths1(size, cvType,
            frameRate, freqLow, freqHigh);
```

```
        }

        return null;
    }
}
```

C.2 Android Temporal Filters

We have chosen to implement the computation of the coefficients for the needed temporal filters in the code itself instead of using external libraries. The reason for this is that most of the mathematical and image processing algorithm needed were already provided by OpenCV and introducing another large library would increase the size of the application unnecessarily and also its complexity and dependencies.

A band-pass filter $H(z)$ was implemented using the difference of two low-pass Butterworth filters with cut-off frequencies ω_l and ω_h. The values of ω_l and ω_h are chosen such that they specify the low and high cut-off frequencies of the pass band, respectively:

$$H(z) = B_H(z) - B_L(z). \tag{C.1}$$

Following A.2, a first order low-pass Butterworth filter with unitary cut-off frequency can be written as $\frac{1}{s+1}$. In order to obtain from it a digital filter with cut-off frequency ω_c, we apply the transformation in Equation A.11, which results in the following frequency responses

$$B_H(z) = \frac{\alpha_H(1+z^{-1})}{(1+\alpha_H)-(1-\alpha_H)z^{-1}} \tag{C.2}$$

and

$$B_L(z) = \frac{\alpha_L(1+z^{-1})}{(1+\alpha_L)-(1-\alpha_L)z^{-1}}, \tag{C.3}$$

with $\alpha_H = \tan\frac{\omega_H}{2r}$ and $\alpha_L = \tan\frac{\omega_L}{2r}$ and r is the sampling rate of the video.

These equations can be turned into recursive relations which can be implemented in the computer through the inverse z-transform and remembering that $B_H(z) = \frac{Y_H(z)}{X(z)}$ and $B_L(z) = \frac{Y_L(z)}{X(z)}$. This results in

$$y_H[n] = \frac{(1-\alpha_H)}{(1+\alpha_H)}y_H[n-1] + \frac{\alpha_H}{(\alpha_H+1)}(x[n]+x[n-1]) \tag{C.4}$$

and

$$y_L[n] = \frac{(1-\alpha_L)}{(1+\alpha_L)} y_L[n-1] + \frac{\alpha_L}{(\alpha_L+1)}(x[n] + x[n-1]). \tag{C.5}$$

Finally, the temporally band-passed signal can then be written as

$$y_{BP}[n] = A(y_H[n] - y_L[n]), \tag{C.6}$$

where A is a normalization factor. We can set this A such that $|H(e^{-j\theta_c})| = 1$ for a given θ_c. A possible value for θ_c is the average between the low and high digital cut-off frequencies. The calculation for the value of A is tedious and therefore is omitted here, but is presented in Appendix E. We could also have ignored this normalization factor and compensated in the value of the amplification parameter α in the next steps of the algorithm. It's important to notice also that the frequency where the given condition is satisfied is not necessarily the frequency with the highest magnitude response, although it will be close to it.

C.3 Android Color-change Magnification

The color-change magnification method implemented here uses the linear Eulerian approach for color-change magnification. This consists in constructing the Gaussian pyramid for each frame and filtering its highest (coarsest) level temporally. The resulting band-passed level is then multiplied by a magnification factor and collapsed in order to obtain a band-passed image with the same dimensions as the original frame. This band-passed and magnified frame is finally added back to the original frame.

The Listing C.4 shows the public methods of the class implemented for color-change magnification, which extends the abstract class *SubtleSignalMagnification* explained earlier. In the constructor, the value of *level* specifies which level of the Gaussian pyramid is to be used for magnification and the *chrominanceAttenuation* is a factor multiplied to the chrominance channels of the specified level. Since in the color-change magnification algorithm the three channels of the colorspace are processed, the given amplification and chrominance attenuation values are used to construct a 'three element' scalar in OpenCV. Notice that the chrominance channels are attenuated by the given factor. A temporal filter builder is also passed to the constructor.

Listing C.4 – Color-change Magnification Class

```
public class ColorChangeMagnification extends
   SubtleSignalMagnification {

   public ColorChangeMagnification(int level, double alpha,
      double chrominanceAttenuation, TemporalFilterBuilder
      builder) {
      this.bandLevel = level;
      this.alpha3f = new Scalar(alpha, alpha *
         chrominanceAttenuation, alpha *
         chrominanceAttenuation);
      this.builder = builder;
   }

   @Override
   public void update(Mat frame) {...}

   @Override
   public void getMagnifiedFrame(Mat dst) {...}
}
```

The *update* method is shown in Listing C.5. When the first frame is passed to it (the attribute *firstFrame* has default initial value *True*), the memory for the object variables used later is initialized. This includes the memory space for the Gaussian pyramid and the initialization of the temporal filters with the temporal filter builder.

Listing C.5 – Color-change Magnification Update Method

```
public void update(Mat frame) {
   if (this.firstFrame) {
      this.firstFrame = false;
      this.initializeMemory(frame);
   }

   rgb2yiq32f(frame, this.frameYIQ);

   this.buildGaussianPyramid(this.frameYIQ,
      this.gaussianPyramid, this.bandLevel);
   this.temporalFilter.filterFrame(this.gaussianPyramid.get(bandLevel),
       this.gaussianPyramid.get(this.bandLevel));
11   multiply(this.gaussianPyramid.get(this.bandLevel),
       this.alpha3f, this.gaussianPyramid.get(this.bandLevel));
12   this.collapseGaussianPyramid(this.gaussianPyramid,
       this.gaussianPyramid);
13   add(this.gaussianPyramid.get(0), this.frameYIQ,
       this.frameYIQ);
14
15   cvtColor(this.frameYIQ, this.frameYIQ,
       Imgproc.COLOR_YUV2BGR);
16   this.frameYIQ.convertTo(this.magnifiedFrame, CvType.CV_8UC3,
       255.0);
17 }
```

The processing of the frame happens from line 7 to 16. The given frame is first converted to a 32 bit float and to the YIQ colorspace. The Gaussian pyramid (implementation explained in Section C.3.1) is built for each channel of this frame and its highest level is filtered temporally and magnified by multiplication with the scalar *alpha3f*. The pyramid is then collapsed. Since its highest level is filtered and magnified, the resulting image has the same dimensions as the original frame, but contains the magnified temporal band of interest. The band is finally added back to the YIQ frame, which is converted back to the RGB color space. The RGB representation is then converted to an unsigned 8-bit matrix whose values are in the range from 0 to 255 and stored in the object attribute *magnifiedFrame*.

Lastly, the *getMagnifiedFrame* simply has to copy the *Mat* in the object attribute *magnifiedFrame* to the output matrix passed as parameter.

C.3.1 Gaussian Pyramid

The implementations of the building and collapsing of the Gaussian pyramids is shown in Listing C.6.

Listing C.6 – Gaussian Pyramid Implementation

```
public void buildGaussianPyramid(Mat frameSrc, List<Mat>
   pyramid, int levels) {
    frameSrc.copyTo(pyramid.get(0));
    for (int i=1; i <= levels; i++) {
        Imgproc.pyrDown(pyramid.get(i-1), pyramid.get(i));
    }
}

public void collapseGaussianPyramid(List<Mat> pyramidSrc,
    List<Mat> pyramidDst) {
    int level = pyramidSrc.size()-1;
    for(int k=level; k > 0; k--) {
        Imgproc.pyrUp(pyramidSrc.get(k), pyramidDst.get(k-1),
            pyramidSrc.get(k-1).size());
    }
}
```

The *buildGaussianPyramid* method builds the pyramid corresponding to the given *frameSrc* and uses the memory space from a pre-allocated pyramid. This pyramid pre-allocation step can be performed as in Listing C.7. The same steps can be used for Laplacian and Riesz pyramids. In the case of the Riesz pyramid, however, it's also necessary to allocate memory for the x and y components of the Riesz transform (that is, the i and j components in the quaternionic representation).

Listing C.7 – Pyramids Memory Initialization

```
GaussianPyramid = new LinkedList<>();
int levelRows = frame.rows();
int levelCols = frame.cols();
for(int k = 0; k <= bandLevel; k++) {
    gaussianPyramid.add(Mat.zeros(levelRows, levelCols,
        CvType.CV_32FC3));
    levelRows = (levelRows + 1)/2;
    levelCols = (levelCols + 1)/2;
}
```

The bulk of the Gaussian pyramid construction and collapsing is performed in the OpenCV methods *pyrUp* and *pyrDown*, which, given and input (the first argument) construct the next or previous level of the pyramid, respectively. The output is stored in the second argument and the third argument specifies the size of the output. This specification of the output size serves only to resolve the ambiguity that arises when building the next level of a pyramid with odd sizes.

C.4 Android Riesz Motion Magnification

Following the pseudo code provided by the original authors, we have implemented the Riesz pyramids approach for motion magnification. This consists first in constructing the Riesz pyramid for the frames being processed. Then, the quaternionic phase of each coefficient of the pyramid needs to be computed. In order to produce and unwrapped version of these phases, we compute the quaternionic phase difference between frames and accumulate them in another variable which represents the current unwrapped quaternionic phase. The resulting quaternionic phase is filtered temporally and spatially and used for shifting the original quaternionic phase of the coefficients of the pyramid. Collapsing the real part of the Riesz pyramid gives the magnified frame.

The Listing C.8 shows the public methods implemented for the Riesz motion magnification class, which extends *SubtleSignalMagnification*. In the constructor, the *numLevels* specifiy the number of levels of the Riesz pyramid and *alpha* is the magnification factor. A temporal filter builder is also passed to the constructor.

Listing C.8 – Android Riesz Motion Magnification Public Methods

```java
public class RieszMotionMagnification extends
   SubtleSignalMagnification{
   ...
   public RieszMotionMagnification(int numLevels, int alpha,
      TemporalFilterBuilder builder) {...}

   @Override
   public void update(Mat frame) {...}

   @Override
   public void getMagnifiedFrame(Mat dst) {...}
```

A slightly modified version of the *update* function is shown in Listing C.9. The method behaves differently when the first frame is passed to it (the initial value of the attribute *firstFrame* is $True$). In this case, it has to initialize the memory for the objects that will later be used in other methods. This includes allocating memory for the Riesz pyramids and building all the necessary temporal filters. The given frame is then converted to the YIQ color space and to 32 bits floats. Each pixel is also scaled down to the range $[0, 1]$ by dividing each of them by 255.0. The luminance channel is extracted from this representation and stored in the first level of the Laplacian pyramid (its base). The Riesz pyramid is then built taking the first element of the given pyramid as the original frame of which to build the pyramid. Further explanation on the construction and collapsing of the Riesz Pyramid can be seen in Section C.4.1.

Since it is the first frame that is being processed, there is no phase difference to compute and the method returns after switching the pointers of the current and previous pyramids. That is, the current pyramid becomes the previous.

Listing C.9 – Riesz Motion Magnification Update

```java
public void update(Mat frame) {
  if (this.firstFrame) {
      this.firstFrame = false;
      this.initializeMemory(frame);

      rgb2yiq32f(frame, this.frameYIQ);
      extractChannel(this.frameYIQ,
         this.laplacianPyramid.get(0), 0);
      this.buildRieszPyramid(this.laplacianPyramid,
         this.rieszPyrX, this.rieszPyrY, this.numLevels);
      this.switchCurrentPreviousPyramids();
      return;
   }

   rgb2yiq32f(frame, this.frameYIQ);
   extractChannel(this.frameYIQ, this.laplacianPyramid.get(0),
      0);
   buildRieszPyramid(this.laplacianPyramid, this.rieszPyrX,
      this.rieszPyrY, this.numLevels);
   for(int k=0; k < numLevels; k++) {
      Mat phaseDiffCos = Mat.zeros(this.phaseCos.get(k).size(),
         this.phaseCos.get(k).type());
```

```
        Mat phaseDiffSin = Mat.zeros(this.phaseCos.get(k).size(),
           this.phaseCos.get(k).type());

        this.computeDifference(phaseDiffCos, phaseDiffSin, k);

        add(this.phaseCos.get(k), phaseDiffCos,
           this.phaseCos.get(k));
        add(this.phaseSin.get(k), phaseDiffSin,
           this.phaseSin.get(k));

        this.temporalFilterCos.get(k).filterFrame(this.phaseCos.get(k),
           phaseCosFiltered);
        this.temporalFilterSin.get(k).filterFrame(this.phaseSin.get(k),
           phaseSinFiltered);

        amplitudeWeightedBlur(phaseCosFiltered, phaseCosFiltered,
           this.amplitude.get(k), this.kSize, this.kSigma);
        amplitudeWeightedBlur(phaseSinFiltered, phaseSinFiltered,
           this.amplitude.get(k), this.kSize, this.kSigma);

        this.shiftPhase(phaseCosFiltered, phaseCosFiltered,
           this.magnifiedLaplacianPyramid, k);
    }

    this.collapseLaplacianPyramid(this.magnifiedLaplacianPyramid,
       this.frameYMag);
    insertChannel(this.frameYMag, this.frameYIQ, 0);
    this.switchCurrentPreviousPyramids();
}
```

The next frames are processed further. After applying similar steps as those done with the first frame processed, we loop over each level of the pyramid computing its magnified version. The first step is to compute the difference between the quaternionic phases of the coefficients of this level in the current pyramid and of the coefficients of the same level in the previous pyramid. The difference is a quaternion whose only non-zero units are the i and j imaginary units and therefore two variables are necessary to store it, *phaseDiffCos* and *phaseDiffSin*. This is done in Line 20 with the compute difference method. Being an object method, it always computes the difference using the current and previous Riesz pyramids, which are stored as attributes in the object. The compute difference method is further explained in Section C.4.2.

The differences are summed to the object attributes *phaseCos* and *phaseSin*, which store, for each level of the pyramid, the current unwrapped quaternionic phase. Their initial values are zeros. The current values of the quaternionic phase is then passed through a temporal filter and the filtered values are stored in the variables *phaseCosFiltered* and *phaseSinFiltered*. After the temporal filtering, the Amplitude Weighted Blur (implementation shown in Section C.4.3) is applied, using the amplitude values that were computed in the *computeDifference* method.

The last pyramid-level-specific operation is the shifting of the quaternionic phase of each level of the frame Riesz Pyramid in Line 31. This is where the motion is actually magnified according to the filtered local phases (in this case, the quaternionic phases) and it is further explained in Section C.4.4. The real part of this computation is stored in the corresponding level of the *modifiedLaplacianPyramid*.

Finally, the real part of the magnified Riesz pyramid (which correspond to the magnified Laplacian pyramid) is collapsed to produce a magnified luminance channel. The luminance channel is then inserted in the luminance channel of the YIQ representation generated earlier and the RGB value can be recovered when calling the *getMagnifiedFrame* method.

C.4.1 Riesz Pyramids

The implementation of a method for building the Riesz Pyramid is shown in Listing C.10. The method takes as inputs pre-allocated pyramids for the real, i and j components of the Riesz Pyramid and the number of levels to be built. Moreover, it expects the frame whose Riesz pyramid is being built to be stored in the first level of the real pyramid.

Listing C.10 – Construction of the Riesz Pyramid

```
1
2 private void buildRieszPyramid(List<Mat> pyrReal, List<Mat>
      pyrX, List<Mat> pyrY, int levels){
3     for(int k=0; k < levels; k++) {
```

```
4        pyrDown(pyrReal.get(k), pyrReal.get(k+1));
5        pyrUp(pyrReal.get(k+1), this.pyrTmp.get(k),
            pyrReal.get(k).size());
6        subtract(pyrReal.get(k), this.pyrTmp.get(k),
            pyrReal.get(k));
7
8        filter2D(pyrReal.get(k), pyrX.get(k), -1, this.kernelX);
9        filter2D(pyrReal.get(k), pyrY.get(k), -1, this.kernelY);
10    }
11 }
12
13 private void initKernels() {
14    this.kernelX = Mat.zeros(new Size(3, 3), type);
15    this.kernelX.put(1, 0, 0.5);
16    this.kernelX.put(1, 2, -0.5);
17
18    this.kernelY = Mat.zeros(new Size(3, 3), type);
19    this.kernelY.put(0, 1, 0.5);
20    this.kernelY.put(2, 1, -0.5);
21 }
```

Lines 4-6 construct the Laplacian pyramid over the memory space of *pyrReal* using the OpenCV *pyrDown* and *pyrUp* methods. Furthermore, an auxiliary memory space is used in the form of a pyramid in the object attribute *pyrAux*. Without this pre-allocated attribute, it would be necessary to initialize a number of Mat objects during the execution of the method, which would unnecessarily increase its processing time. Then, Lines 8-9 convolve using the OpenCV *filter2D* function the kernels used for building the i and j components of the pyramid with the current level of the Laplacian pyramid. These kernels can be initialized as in the method shown in Line 13. We observe also that the real pyramid has one more level than the i and j components and that corresponds to the residue of the Laplacian pyramid.

Collapsing a Riesz Pyramid in order to obtain the frame which would originate it is equivalent to collapsing the Laplacian pyramid stored in its real part. The implementation of collapsing a Laplacian Pyramid is shown in Listing C.11. In this case, we collapse the given pyramid over the pre-allocated *pyrTmp* and store the resulting frame in the Mat *dst* given.

Listing C.11 – Collapsing a Laplacian Pyramid

```
private void collapseLaplacianPyramid(List<Mat> pyramid, Mat
   dst) {
   int sizePyr = pyramid.size() - 1;

   pyrUp(pyramid.get(sizePyr), this.pyrTmp.get(sizePyr-1),
      pyramid.get(sizePyr-1).size());
   add(pyramid.get(sizePyr-1), this.pyrTmp.get(sizePyr-1),
      this.pyrTmp.get(sizePyr-1));

   for(int k=sizePyr-1; k > 1; k--) {
      pyrUp(this.pyrTmp.get(k), this.pyrTmp.get(k-1),
         pyramid.get(k-1).size());
      add(pyramid.get(k-1), this.pyrTmp.get(k-1),
         this.pyrTmp.get(k-1));
   }

   pyrUp(this.pyrTmp.get(1), this.pyrTmp.get(0),
      pyramid.get(0).size());
   add(pyramid.get(0), this.pyrTmp.get(0), dst);
}
```

C.4.2 Computation of the Quaternionic Phase Difference

The *computeDifference* (signature shown in Listing C.12) function computes the difference between the quaternionic phase of the current and previous Riesz pyramids coefficients in the given level. The phases can then be used to compute the unwrapped sequence of quaternionic phases for each pixel over time. The differences are computed between the current and preivous Riesz Pyramids coefficients, which are attributes of the Riesz Magnification object in use, and are stored in the given variables *phaseDiffCos* and *phaseDiffSin*, whose values correspond respectively to the *i* and *j* units of the quaternionic phase difference, which is a quaternion.

Listing C.12 – Declaration of Compute Difference Method

```
private void computeDifference(Mat phaseDiffCos, Mat
   phaseDiffSin, int level) {...}
```

We’ve implemented this method using the JNI and C++. The reason for doing this is that it is a computationally expensive operation which has to be repeated multiple times (for each coefficient in each level of the pyramid). Furthermore, some of the mathematical operations used are not available as matrices operations in OpenCV. The C++ implementation used is shown in Listing C.13. The parameters *lapPyrLevel*, *rieszXPyrLevel* and *rieszYPyrLevel* are the real, i and j components of the quaternionic representation of the coefficients of the current level of the Riesz pyramid. *lapPyrPrevLevel*, *rieszXPyrPrevLevel* and *rieszYPyrPrevLevel* are similar, but point to the pyramid of the previous frame. The *amplitudeLevel* is where to store the magnitudes of the Riesz Pyramid coefficients. The method loops over each element of the matrices by setting pointers to each row and the actual computation happens in Lines 26 - 36.

Listing C.13 – C++ Implementation of computeDifference

```
1  void compute_difference_loop(Mat& phaseDiffCos, Mat&
      phaseDiffSin, Mat& lapPyrLevel, Mat& rieszXPyrLevel, Mat&
      rieszYPyrLevel, Mat& lapPrevPyrLevel, Mat&
      rieszXPyrPrevLevel, Mat& rieszYPyrPrevLevel, Mat&
      amplitudeLevel)
2  {
3     float min_val = 1e-7;
4     int nRows = phaseDiffCos.rows;
5     int nCols = phaseDiffCos.cols;
6
7     int i,j;
8     float *pcos, *psin, *plap, *prieszx, *prieszy;
9     float *plapprev, *prieszxprev, *prieszyprev, *pamp;
10    float q_conj_prod_real, q_conj_prod_x, q_conj_prod_y,
         q_conj_prod_amplitude;
11    float phase_difference, den;
12
13    for( i = 0; i < nRows; ++i)
14    {
15       pcos = phaseDiffCos.ptr<float>(i);
16       psin = phaseDiffSin.ptr<float>(i);
```

```
      plap = lapPyrLevel.ptr<float>(i);
      prieszx = rieszXPyrLevel.ptr<float>(i);
      prieszy = rieszYPyrLevel.ptr<float>(i);
      plapprev = lapPrevPyrLevel.ptr<float>(i);
      prieszxprev = rieszXPyrPrevLevel.ptr<float>(i);
      prieszyprev = rieszYPyrPrevLevel.ptr<float>(i);
      pamp = amplitudeLevel.ptr<float>(i);

      for (j = 0; j < nCols; ++j) {
         q_conj_prod_real = plap[j] * plapprev[j] + prieszx[j]
            * prieszxprev[j] + prieszy[j] * prieszyprev[j];
         q_conj_prod_x = plapprev[j] * prieszx[j] - plap[j] *
            prieszxprev[j];
         q_conj_prod_y = plapprev[j] * prieszy[j] - plap[j] *
            prieszyprev[j];
         q_conj_prod_amplitude = sqrt(q_conj_prod_real *
            q_conj_prod_real + q_conj_prod_x * q_conj_prod_x +
            q_conj_prod_y * q_conj_prod_y) + min_val;

         phase_difference = acos(q_conj_prod_real /
            q_conj_prod_amplitude);
         den = sqrt(q_conj_prod_x * q_conj_prod_x +
            q_conj_prod_y * q_conj_prod_y) + min_val;
         pcos[j] = phase_difference * q_conj_prod_x / den;
         psin[j] = phase_difference * q_conj_prod_y / den;

         pamp[j] = sqrt(q_conj_prod_amplitude);
      }
   }
}
```

The method computes Equation C.7, where r_m and r_{m-1} represent a coefficient of the current and previous Riesz Pyramid. Moreover, $\mathbf{v}$ and q are such that $r_m r_{m-1}^* = q + \mathbf{v}$ and $\mathbf{v}$ is the imaginary part of the quaternion. The imaginary component k of the product is ignored, since it's very small under the assumption that the orientation of the pixel is not changing. That is, the θ in the spherical representation of the Riesz pyramid coefficients.

$$\frac{\mathbf{v}}{\|\mathbf{v}\|} \arccos \frac{q}{\|r_m r_{m-1}^*\|} \tag{C.7}$$

The equivalence between Equation C.7 and the difference of the quaternionic phases is derived in Appendix D.

Lines 26 - 28 compute the real, i and j values of $r_m r_{m-1}^*$, while the k component is ignored and its norm is calculated in Line 29. The values of *phaseDiffCos* and *phaseDiffSin* (the i and j components of Equation C.7) are computed in Lines 31 - 34. Finally, the magnitude of the coefficient is approximated in Line 36. This approximation is based on the idea that the amplitude changes only slightly and is used later for an amplitude-weighted blurring of the quaternionic phases.

C.4.3 Amplitude-weighted Blur

The implementation of the amplitude-weighted blur is shown in Listing C.14. It uses the OpenCV *GaussianBlur* methods in order to apply the Gaussian blurring.

Listing C.14 – Amplitude-weighted Blur

```java

public static void amplitudeWeightedBlur(Mat src, Mat dst, Mat
    amplitude, double kSize, double kSigma) {
    Mat den = Mat.zeros(src.size(), src.type());
    multiply(src, amplitude, dst);

    GaussianBlur(dst, dst, new Size(kSize, kSize), kSigma);
    GaussianBlur(amplitude, den, new Size(kSize, kSize), kSigma);

    divide(dst, den, dst);
}
```

C.4.4 Shifting the Quaternionic Phase

The *shiftPhase* method (declaration in Listing C.15) shifts the phase of the current coefficients of the determined Riesz pyramid level according to the given quaternionic phase. The arguments *phaseFilteredCos phaseFilteredSin* represent respectively the i and j units of the quaternionic phase. The *magPyr* argument specify where the resulting pyramid should be stored and *level* selects which level of the pyramid. The method always uses the current Riesz pyramid (which is an object attribute) as the pyramid whose phases should be shifted, but it stores the results in the given *magPyr*.

Listing C.15 – Declaration of the Shift Phase Method

```
private void shiftPhase(Mat phaseFilteredCos, Mat
   phaseFilteredSin, List<Mat> magPyr, int level) {
```

We've implemented this method using the Java Native Interface (JNI) and C++, since it's an expensive operation. Moreover, it uses methods that are not available as matrices operations in OpenCV. The C++ implementation is shown in Listing C.16. The parameters *lapPyrLevel*, *rieszXPyrLevel* and *rieszYPyrLevel* are the real, i and j components of the quaternionic representation of the coefficients of the current level of the Riesz pyramid. *alpha* is the magnification factor. The method loops over each element of the matrices by setting pointers to each row and the actual computation happens only in Lines 25 - 33.

Listing C.16 – C++ Implementation of shiftPhase

```
1  void shift_phase_loop(Mat& phaseFilteredCos, Mat&
      phaseFilteredSin, Mat& magPyrLevel,
2                Mat& lapPyrLevel, Mat& rieszXPyrLevel, Mat&
                    rieszYPyrLevel,
3                double alpha)
4  {
5     int nRows = phaseFilteredCos.rows;
6     int nCols = phaseFilteredCos.cols;
7
8     int i,j;
9     float *pcos, *psin, *pmag, *plap, *prieszx, *prieszy;
10    float magcos, magsin, phase_mag, exp_phase_real, exp_phase_x;
```

```
    float exp_phase_y;
    float min_val = 1e-7;

    for( i = 0; i < nRows; ++i)
    {
       pcos = phaseFilteredCos.ptr<float>(i);
       psin = phaseFilteredSin.ptr<float>(i);
       pmag = magPyrLevel.ptr<float>(i);
       plap = lapPyrLevel.ptr<float>(i);
       prieszx = rieszXPyrLevel.ptr<float>(i);
       prieszy = rieszYPyrLevel.ptr<float>(i);

       for (j = 0; j < nCols; ++j)
       {
          magcos = pcos[j] * alpha;
          magsin = psin[j] * alpha;

          phase_mag = sqrt(magcos * magcos + magsin * magsin) +
             min_val;
          exp_phase_real = cos(phase_mag);
          exp_phase_x = magcos / phase_mag * sin(phase_mag);
          exp_phase_y = magsin / phase_mag * sin(phase_mag);

          pmag[j] = exp_phase_real * plap[j] - exp_phase_x *
             prieszx[j] - exp_phase_y * prieszy[j];
       }
    }
}}
```

The method computes Equation C.8, where an arbitrary coefficient of the level of the Riesz pyramid is given by $I + r_x i + r_y j$ and the shifting quaternionic phase is given by $b_x i + b_y j$. Appendix D shows the equivalence between Equation C.8 and shifting the phases of a Riesz pyramid.

$$I \cos \sqrt{b_x^2 + b_y^2} - r_x \frac{b_x \sin\left(\sqrt{b_x^2 + b_y^2}\right)}{\sqrt{b_x^2 + b_y^2}} - r_y \frac{b_y \sin\left(\sqrt{b_x^2 + b_y^2}\right)}{\sqrt{b_x^2 + b_y^2}} \tag{C.8}$$

In Lines 25 and 26 the quaternionic phase is multiplied by the magnification factor α. Then Lines 28 - 31 calculate the exponential of the quaternionic phase. Finally, the real part of the product between the current pyramid and the exponential is given in Line 33. Only the real part of the product is necessary, since it constitutes the Laplacian pyramid of the frame which is then collapsed in order to obtain the magnified frame.

APPENDIX D — DIFFERENCE OF QUATERNIONIC PHASES AND SHIFTING THE QUATERNIONIC PHASE

In this Section, we show the equivalence between Equation C.7 and the difference of the quaternionic phases. Specifically, the difference between the quaternionic phases of the coefficient r_m of a Riesz Pyramid and the correspondent coefficient r_{m-1} of the previous Riesz Pyramid. We also let $r_m r^*_{m-1} = q + \mathbf{v}$, where **v** is the imaginary part of the quaternion and q is its real part.

$$\begin{aligned}
\log \frac{r_m}{\|r_m\|} - \log \frac{r_{m-1}}{\|r_{m-1}\|} &= \log \left(\frac{r_m}{\|r_m\|} \left(\frac{r_{m-1}}{\|r_{m-1}\|}) \right)^{-1} \right) \\
&= \log \frac{r_m r^*_{m-1}}{\left\|r_m r^*_{m-1}\right\|} \\
&= \log r_m r^*_{m-1} - \log \left\|r_m r^*_{m-1}\right\| \\
&= \log \left\|r_m r^*_{m-1}\right\| + \frac{\mathbf{v}}{\|\mathbf{v}\|} \arccos \frac{q}{\left\|r_m r^*_{m-1}\right\|} - \log \left\|r_m r^*_{m-1}\right\| \\
&= \frac{\mathbf{v}}{\|\mathbf{v}\|} \arccos \frac{q}{\left\|r_m r^*_{m-1}\right\|}
\end{aligned} \tag{D.1}$$

The real part of a Riesz Pyramid whose quaternionic phase was shifted is given by Equation C.8. In order to see this, we consider an arbitrary coefficient of the Riesz pyramid given by $I + r_x i + r_y j$ to be shifted by the quaternionic phase $b_x i + b_y j$. The first step is to exponentiate the phase, which results in

$$e^{b_x i + b_y j} = \cos \sqrt{b_x i + b_y j} + \frac{(b_x i + b_y j)}{\sqrt{b_x i + b_y j}} \sin \sqrt{b_x i + b_y j} \tag{D.2}$$

Multiplying this result by $I + r_x i + r_y j$ and taking only the real part gives Equation C.8.

APPENDIX E — DERIVATION OF THE NORMALIZATION FACTOR FOR THE DIFFERENCE OF BUTTERWORTHS

In this Section, we wish to find the value of A such that $|H(e^{j\theta_c})| = 1$ for a given θ_c with

$$H(z) = A\left(B_H(z) - B_L(z)\right) \tag{E.1}$$

and $B_H(z)$ and $B_L(z)$ as defined in Equations C.2 and C.3.

$$\begin{aligned} H(z) &= B_H(z) - B_L(z) \\ &= \frac{\alpha_H(1+z^{-1})}{(1+\alpha_H)-(1-\alpha_H)z^{-1}} - \frac{\alpha_L(1+z^{-1})}{(1+\alpha_L)-(1-\alpha_L)z^{-1}} \\ &= \frac{(1+z^{-1})(\alpha_H((1+\alpha_L)-(1-\alpha_L)z^{-1}) - \alpha_L((1+\alpha_H)-(1-\alpha_H)z^{-1}))}{((1+\alpha_L)-(1-\alpha_L)z^{-1})((1+\alpha_H)-(1-\alpha_H)z^{-1})} \end{aligned} \tag{E.2}$$

By computing the norm of each element individually, we find

$$|H(e^{j\theta_c})| = \frac{\sqrt{2(1+\cos\theta_c)}\sqrt{2(\alpha_H-\alpha_L)(1-\cos\theta_c)}}{D_H D_L} \tag{E.3}$$

with

$$D_H = \sqrt{(1+\alpha_H)^2 - 2(1+\alpha_H)(1-\alpha_H) + (1-\alpha_H)^2}$$

and

$$D_L = \sqrt{(1+\alpha_L)^2 - 2(1+\alpha_L)(1-\alpha_L) + (1-\alpha_L)^2}$$

Finally, in order to normalize $H(z)$ according to the predefined constraint, we must have $A = \frac{1}{|H(e^{j\theta_c})|}$.

APPENDIX — REFERENCES

ARANGO, C. et al. Subtle motion analysis and spotting using the riesz pyramid. In: . [S.l.: s.n.], 2018.

BALAKRISHNAN, G.; DURAND, F.; GUTTAG, J. Detecting pulse from head motions in video. In: **Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition**. [S.l.: s.n.], 2013. p. 3430–3437.

BELAID, A. et al. Phase-based level set segmentation of ultrasound images. **IEEE Transactions on Information Technology in Biomedicine**, IEEE, v. 15, n. 1, p. 138–147, 2010.

CHEN, W.; MCDUFF, D. Deepmag: Source specific motion magnification using gradient ascent. **arXiv preprint arXiv:1808.03338**, 2018.

CHEN, W.; PICARD, R. W. Eliminating physiological information from facial videos. In: IEEE. **2017 12th IEEE International Conference on Automatic Face & Gesture Recognition (FG 2017)**. [S.l.], 2017. p. 48–55.

DAVIS, A. et al. Visual vibrometry: Estimating material properties from small motion in video. In: **Proceedings of the ieee conference on computer vision and pattern recognition**. [S.l.: s.n.], 2015. p. 5335–5343.

ELGHARIB, M. et al. Video magnification in presence of large motions. In: **Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition**. [S.l.: s.n.], 2015. p. 4119–4127.

FELSBERG, M.; SOMMER, G. The monogenic signal. **IEEE transactions on signal processing**, IEEE, v. 49, n. 12, p. 3136–3144, 2001.

FREEMAN, W. T.; ADELSON, E. H.; HEEGER, D. J. Motion without movement. **ACM Siggraph Computer Graphics**, ACM New York, NY, USA, v. 25, n. 4, p. 27–30, 1991.

FREEMAN, W. T.; ADELSON, E. H. et al. The design and use of steerable filters. **IEEE Transactions on Pattern analysis and machine intelligence**, v. 13, n. 9, p. 891–906, 1991.

Gautama, T.; Van Hulle, M. A. A phase-based approach to the estimation of the optical flow field using spatial filtering. **IEEE Transactions on Neural Networks**, v. 13, n. 5, p. 1127–1136, 2002.

Jianbo Shi; Tomasi. Good features to track. In: **1994 Proceedings of IEEE Conference on Computer Vision and Pattern Recognition**. [S.l.: s.n.], 1994. p. 593–600.

LAURIDSEN, H. et al. Extracting physiological information in experimental biology via eulerian video magnification. **BMC biology**, BioMed Central, v. 17, n. 1, p. 1–26, 2019.

Le Ngo, A. C. et al. Micro-expression motion magnification: Global lagrangian vs. local eulerian approaches. In: **2018 13th IEEE International Conference on Automatic Face Gesture Recognition (FG 2018)**. [S.l.: s.n.], 2018. p. 650–656.

LIU, C. et al. Motion magnification. **ACM transactions on graphics (TOG)**, ACM New York, NY, USA, v. 24, n. 3, p. 519–526, 2005.

LIU, L. et al. Enhanced eulerian video magnification. In: IEEE. **2014 7th International Congress on Image and Signal Processing**. [S.l.], 2014. p. 50–54.

OH, T.-H. et al. Learning-based video motion magnification. In: **Proceedings of the European Conference on Computer Vision (ECCV)**. [S.l.: s.n.], 2018. p. 633–648.

PERROT, V. et al. Video magnification applied in ultrasound. **IEEE Transactions on Biomedical Engineering**, IEEE, v. 66, n. 1, p. 283–288, 2018.

PORTILLA, J.; SIMONCELLI, E. P. A parametric texture model based on joint statistics of complex wavelet coefficients. **International journal of computer vision**, Springer, v. 40, n. 1, p. 49–70, 2000.

SIMONCELLI, E. P.; FREEMAN, W. T. The steerable pyramid: A flexible architecture for multi-scale derivative computation. In: IEEE. **Proceedings., International Conference on Image Processing**. [S.l.], 1995. v. 3, p. 444–447.

SIMONCELLI, E. P. et al. Shiftable multiscale transforms. **IEEE transactions on Information Theory**, IEEE, v. 38, n. 2, p. 587–607, 1992.

UNSER, M.; SAGE, D.; VILLE, D. V. D. Multiresolution monogenic signal analysis using the riesz–laplace wavelet transform. **IEEE Transactions on Image Processing**, IEEE, v. 18, n. 11, p. 2402–2418, 2009.

WADHWA, N. **Revealing and analyzing imperceptible deviations in images and videos**. Thesis (PhD) — Massachusetts Institute of Technology, 2016.

WADHWA, N. et al. Phase-based video motion processing. **ACM Transactions on Graphics (TOG)**, ACM New York, NY, USA, v. 32, n. 4, p. 1–10, 2013.

WADHWA, N. et al. Quaternionic representation of the riesz pyramid for video magnification. 2014.

WADHWA, N. et al. Riesz pyramids for fast phase-based video magnification. In: IEEE. **2014 IEEE International Conference on Computational Photography (ICCP)**. [S.l.], 2014. p. 1–10.

WU, H.-Y. et al. Eulerian video magnification for revealing subtle changes in the world. **ACM transactions on graphics (TOG)**, ACM New York, NY, USA, v. 31, n. 4, p. 1–8, 2012.

WU, X. et al. Amplitude-based filtering for video magnification in presence of large motion. **Sensors**, Multidisciplinary Digital Publishing Institute, v. 18, n. 7, p. 2312, 2018.

WU, X. et al. Pca-based magnification method for revealing small signals in video. **Signal, Image and Video Processing**, Springer, v. 12, n. 7, p. 1293–1299, 2018.

ZHANG, Y.; PINTEA, S. L.; GEMERT, J. C. van. Video acceleration magnification. In: **Computer Vision and Pattern Recognition**. [S.l.: s.n.], 2017.