UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Facultat d'Informàtica de Barcelona

FIB

CONTACTLESS POLYGRAPH

MILENA MORI YUKIE KIMURA

Thesis supervisor: JOAN CLIMENT VILARÓ (Department of Automatic Control)

Degree: Master Degree in Innovation and Research in Informatics (Advanced Computing)

Thesis report

Facultat d'Informàtica de Barcelona (FIB)

Universitat Politècnica de Catalunya (UPC) - BarcelonaTech

28/06/2023

Abstract

The constant transformation of our surroundings often goes unnoticed, but it occurs nonetheless. Imperceptibly, our skin undergoes slight color changes as our hearts beat, and our heads subtly move with each breath. While these alterations may escape our eyes, they are captured by cameras. We can magnify the variation in those small signals to extract important information such as heartbeat rate, which in turn can be used in many real-life applications such as polygraph tests. The polygraph, or lie detector test, is a tool that measures physiological changes in an individual to detect deception. This thesis aimed to explore the feasibility of using video information to extract heartbeat rates, replacing the traditional sensors that are commonly used to extract this information. Through a series of experiments involving videos of people standing still, and a video of a real-life suspect that was lying, it was consistently possible to enhance the color changes that occur in a person's skin when their heartbeats. Though it was not possible to develop a fully-functioning lie detector test, this thesis proves that the motion and color magnification technique is reliable even when applied to videos of individuals engaged in speech and movement. It also proved that it is an interesting solution that has the potential of revolutionizing how interviews and interrogations are conducted.

Chapter 1

Introduction

The ability to tell whether one is lying or not is one that many have coveted throughout history. With that intention, we, as humans, have invented many contraptions and methods to try to extract information from a person. Many of those methods are unethical and unreliable, such as the truth serum and other methods of torture that should not be used anymore [5].

With that in mind, Dr. John Augustus Larson proposed a machine capable of aiding detectives in the detection of lies in persons of interest[4]. These machines are called polygraphs, or more commonly, "lie detectors". Albeit controversial, they are commonly used by many police departments all over the world, and they have helped to solve many cases.

The polygraph was devised as a way to measure physiological changes in a person when asked pressing questions, which can indicate deceit. This is done by connecting electrodes to different parts of a person's body. The main things that are measured in a polygraph are:

- Heart rate;
- Blood pressure;
- Respiration;
- Skin conductivity;

When the polygraph was first created, the electrodes and wires, and sensors that collected the data would be properly placed on the suspect, and then plugged into a machine, which would, in turn, register the values on paper. In more modern polygraph machines, everything is connected directly to a computer, so we do not need a specific machine to apply the lie detector test. We only need the wires and measurement tools and the adaptor to connect to the computer.

However, one thing has not changed with time. We still need to attach all of the measurement equipment to the person being evaluated, which might make them nervous and botch the results of the test. It would also be interesting to try and detect lies in a person that is not expecting to be evaluated.

With that in mind, we thought it would be interesting to develop a *Contactless Polygraph*, extracting all the necessary information from videos, using Eulerian video magnification[9] to amplify minute movements and color changes which can be useful to obtain a person's heartbeat and breathing rate.

However, it is important to keep in mind that this is purely an academic study, and it is not suitable to be used in real-life situations just yet.

1 The Polygraph

As previously mentioned, the polygraph is a **lie detector machine**. It has been used in many high-profile criminal cases to uncover the truth. However, it is not a fool-proof method and it is also inadmissible in court in most countries. It is a difficult tool to use, and it requires the interviewer to have special training and certification for it to be effective.

That being said, it can be a useful resource to help identify persons of interest and extract difficult truths, provided that its results are not used by themselves, but are taken into account when analyzing other bits of evidence and data related to the crime.

1.1 Traditional data collection

As previously mentioned, the polygraph works by measuring mainly the suspect's heart rate, respiratory activity, and electrodermal activity (also known as Galvanic Skin Response or GSR). Different measurement tools can be used to obtain data for the three different channels.

To measure blood pressure and heart rate, a sensor to measure blood flow is placed on the suspect's fingers, and a regular blood pressure cuff is placed on their arm, with the latter being the most common.

To measure breathing rate, two sensors are placed around the interviewed person's chest, one above the heart and one below it.

To measure GSR, a sensor is placed on the suspect's palm, and any changes in perspiration can be detected by the lie detector. In other words, if their hand is suddenly drier or more sweaty, the interviewer will know. This is done by placing two electrodes in different fingers in the suspect's hand, and a small current is applied. We can then measure conductance to determine changes in the level of perspiration.

Motion detector sensors are also usually used to alert the interviewer in case the interviewed person moves in a way that might influence the test. [3]

1.2 Issues

As mentioned in the previous section, most of the sensors used in the process of data collecting can be easily influenced by the sudden movements of the suspect, which can make applying the test uncomfortable and complicated.

These measurements are also not always good indicators of deception. Heart rate and blood pressure can be influenced by factors such as any perceived threat, increase in physical or mental activity, the anticipation of a threat or activity, or more generally, any form of arousal. The "baseline" of any cardiovascular activity is not the same between individuals, as it is not even the same for a single individual in different settings. This can make it difficult for interviewers to determine which patterns are actually signs of deception.[10]

Breathing rate is also subject to the same limitations. And it is also influenced by different factors, which can also alter the results of the test in different ways. For example, the person's autonomous nervous system might "choose" to change the person's breathing rate because of a difference in oxygen or carbon dioxide concentrations in the air. On the other hand, the suspect might try to make his breathing deliberately slow to attempt to seem calm to the investigator.

According to Synnott et. al., the most reliable of the three channels of the polygraph is the ectodermal activity[10]. But, alas, it is not something we can effectively measure using video, so the person using the contact-less polygraph in a real-life setting would still have to use the sensor to measure their GSR.

Chapter 2

Background

In order to fully understand the theory behind the concepts that will be explained in this thesis, there are some basic ideas that need to be remembered or learned, and I will specify them in the following chapters.

1 Markov Random Field

A **Markov Random Field** (or MRF) is a probabilistic graphical model used in the fields of physics and probability, computer vision, and image processing tasks. It can be used to graph the relationships and statistical dependencies among random variables in an image or signal.[12].

A random field refers to a collection of random variables defined on a specific domain, such as an image grid. Each random variable represents a value associated with a particular location in the domain.

This type of graphical model follows the **Markov Property**, which states that a value of a random variable is only dependent on the values of its neighbors. In more technical terms, a random variable's value is conditionally independent of all other variables in the field, if its neighboring variables are given. We can write this equation in terms of probabilities in the equation 2.1.

$$P(X_n|X_V - X_n) = P(X_n|X_{N_n}) \tag{2.1}$$

An MRF is a type of undirected graph $G = (V, \epsilon)$, which means that its edges do not have an associated direction, which serves to indicate the symmetry of the dependencies between random variables. V represents the nodes that correspond to random variables in the graph.

In an MRF, given the set of independent random variables, X_s, associated with the set of nodes S. Let the edges between said nodes, denoted by the letter ϵ, represent dependence relationships. Let A, B, and C be disjointed subsets of nodes. If there is no path from subset A that does not pass through any node of X_C to get to X_B, then X_A is conditionally independent of X_B given X_C.

The neighbor set N_n of a node n can be defined by the set of nodes that are connected to n via edges ϵ of the graph:

$$N_n = \{m \in V|(n, m)in\epsilon\}$$

If we have a node n's neighbor set, then we can say that n is independent of all other nodes in the graph. And we can write the **Markov Property**:

Inference is one of the main tasks of an MRF. It involves the estimation of the most likely configuration of a set of random variables, given observed data or constraints provided by the potential function. Inference in MRFs can be often challenging and can involve methods such as graph cuts or belief propagation (which is used in Rubinstein's study).

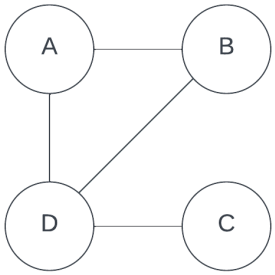


Figure 2.1: An example of a clique in a Markov Random Field graph. A, B, and D form a clique.

1.1 Potential Function

The **Potential Function** ϕ (also known as energy function or cost function) is what characterizes the MRF, and describes the compatibility or agreement between the values of neighboring variables. It assigns lower energy to configurations that are more likely or desirable.

ϕ is defined for neighboring nodes or variables in the graph, which are typically nodes that are connected to each other by edges in the graphical map. It assigns a cost or energy to each possible configuration of the neighboring variables, and lower values are associated with more desirable or more likely configurations, and the opposite is true for higher values of ϕ.

By manipulating the potential function ϕ, researchers can shape the behavior and properties of the MRF, such as encouraging smoothness, promoting certain configurations, or incorporating prior knowledge about the previous domain.

The formulation of the potential function ϕ is specific to each specific application and problem being addressed. And different tasks might require different types of potential functions designed to fulfill specific requirements.

1.2 Clique

A **clique** in the context of an MRF is a subgraph of G whose nodes are completely connected to each other, as represented in Figure 2.1.

A **clique** can be understood as a group of variables in the Markov Random Field that interact directly with each other, based on the underlying dependencies defined by the MRF structure.

Cliques provide a structured way to represent the interactions and dependencies between variables in an MRF. By defining potential functions associated with cliques, MRFs can model complex relationships and capture the local interactions between variables.

MRFs are often defined based on the concept of factorization. The joint probability distribution of variables in the MRF can be factorized as a product of other potential functions, each associated with a clique in the graphical model. The potential function describes the compatibility between the variables within the clique.

The potential function associated with each clique is ϕ_c, which is real and non-negative. ϕ_c does not need to correspond to conditional probability distributions of any kind.

ϕ_c usually takes the form:

$$\phi_c(x_c) = e^{-\frac{1}{T}V_c(x_c)} \tag{2.2}$$

T is called *temperature*, but it is often assumed to be 1. $V_c(x_c)$ can also be referred as the *clique potential*.

A maximum clique is a clique that cannot be extended by adding another variable without violating the property that every pair of nodes within the clique has to be connected.

Cliques play a crucial role in performing efficient interference in MRFs. Inference algorithms such as belief propagation or belief passing operate on cliques to compute messages or propagate information through the graph. The connectivity of the cliques and the Markov property makes it convenient for computations to be performed on them.

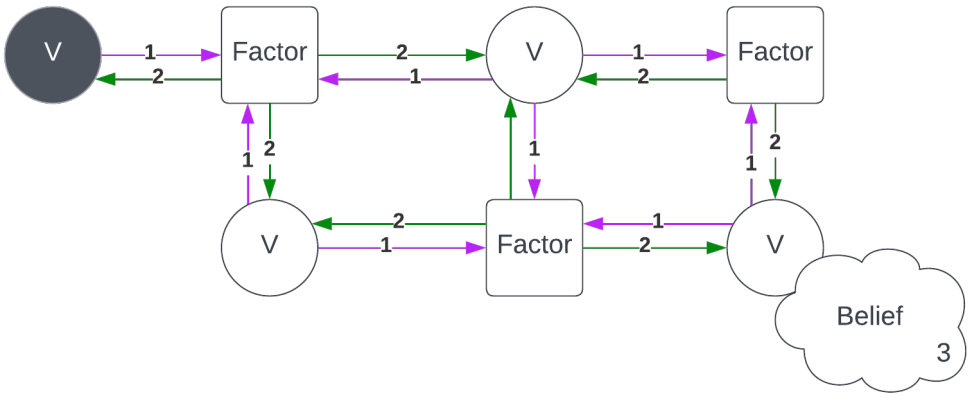


Figure 2.2: An example of belief propagation being applied on a graph. In purple, we have the messages initially sent by the variable nodes (step 1). In green, we have the messages sent back by the factor nodes (step 2). The thought bubble illustrates the belief calculated by one of the variable nodes (step 3).

2 Message Passing

Message passing is a category of algorithms used to perform inference in graphical models, such as Bayesian Networks or Markov Random Fields. They carry that name because it is a method used to exchange information or propagate messages between nodes in a graphical model.

The overall structure of message-passing algorithms is:

1. **Initialization of Messages** Messages from variable nodes to factor nodes are initialized. They can be set to a random value, or to non-informative values from the hidden nodes; these non-informative values set the costs of each possible state to be equal.

   Messages can be thought of as a variable telling its neighbors what it thinks its state is, or what it thinks the cost would be for it to change from its current state to its other possible states.

2. **Computation of Factor Node Messages** The factor nodes receive all the messages from their neighboring variable nodes, and compute messages that they send to their neighboring hidden variable nodes. This message tells the neighboring nodes what state they should be in, and how much it would cost for them to change states.

3. **Computation of Hidden Node Beliefs**The hidden nodes receive the messages from the factor nodes and compute a belief, what they believe their state should be. In belief propagation algorithms, this takes the form of a cost, or a probability, associated with each of its possible states.

4. **Selection of Single Guess**We use a threshold on the values calculated in the previous step to choose a single guess for a variable node.

5. **Termination Check** We now have a guess for the overall configuration of the factor graph, and we can use this to check for a termination condition. If the termination condition has been satisfied, we output the current guess.

   If not, the variable nodes will compute new messages and send them to the factor nodes, thus initiating a new iteration

In Figure 2.2, we can see a small example of a factor graph and some of the stages of belief propagation.

2.1 Loopy Belief Propagation

Loopy Belief Propagation or LBP is named "loopy" because it allows messages to circulate through the loops in the graphical model, despite the cycles. The solutions found by the LBP are approximations, but it is effective for a wide range of problems and can provide reasonable approximations in many cases. [14]

Because convergence is not guaranteed, an additional step is required to be added at the end of the message-passing algorithm: convergence check. In this step, a convergence check is performed to determine whether the algorithm has reached a *stable* state.

We can verify whether the algorithm has reached a stable state by comparing the current messages with the messages from the previous iterations and measuring the difference between them. If the difference falls below a predefined threshold, or if the maximum number of iterations has been reached, the algorithm is terminated.

3 Kullback-Leibler Divergence

Kullback-Leibler divergence (also known as relative entropy) is a type of statistical distance, and it is often used as a metric of how different two probability distributions are from each other. It is commonly used in various fields, including information theory, statistics, and machine learning. [2]

It is a non-symmetric measure of relative entropy, or difference in information, between two distributions. When the relative entropy between two distributions is 0, they both have identical quantities of information.

Given two probability distributions P and Q, the KL-divergence from P to Q is denoted by $KL(P||Q)$, and it is calculated as the expectation of the logarithmic difference between the probabilities of the two distributions:

$$KL(P||Q) = \sum_X P(X) log \frac{P(X)}{Q(X)} \tag{2.3}$$

This quantity is asymmetric, which means that $KL(P||Q)$ is often different from $KL(Q||P)$. It is also non-negative and has a minimum value when $P = Q$, or when they are equal up to a constant factor.

4 Correlation in Image Processing

Correlation in image processing refers to the process of moving a filter mask (i.e. a kernel) over an image and computing the sum of products at each location. It is a function of displacement of the filter. [6]

In other words, correlation refers to a technique that measures the similarity or relationship between two images or different parts of the same image. It involves comparing the pixel values or intensity patterns in the image to determine their degree of similarity or correspondence.

Its mathematical formula is given by:

$$F \circ I(x, y) = \sum_{j=-N}^{N} \sum_{i=-N}^{N} F(i, j) I(x + i, y + i) \tag{2.4}$$

Cross-correlation is a common form of correlation used in image processing. It involves sliding a smaller "template" image or kernel over a larger image and computing the similarity between the template and the corresponding image in the source image at each position.

The correlation coefficient is a measure of the linear relationship between two variables, such as pixel intensities in two images. It ranges from -1 to 1, and if it is 1 the correspondence between the two variables is perfect, while -1 indicates perfect negative correlation. A correlation close to 0 indicates little to no relationship between them.

Cross-correlation has two very interesting properties:

- Translation Invariance: Therefore, we can use this to find an object anywhere in the image.

- Locality: Allows our system to focus on an area of interest in an image, ignoring the rest of it. In other words, different areas of images (usually) don't interfere with each other.

Correlation has various applications in image processing, including object detection, image recognition, motion tracking, image registration, and image quality assessment. It is a fundamental tool for analyzing and comparing images in many computer vision and image analysis tasks.

5 Fourier Analysis

Fourier Analysis is a technique used by many different areas of science, including mathematics, physics, engineering, and signal processing. It is a study of general functions that have had the Fourier Transform applied to them, which allows us to understand the frequency content and amplitude of a signal.

The Fourier Analysis is used to decompose a complex signal or function into a sum of simpler sinusoidal components with different values of frequency and amplitude. In other words, it provides a representation of a signal in the frequency domain. By decomposing a signal into its frequency components, we can analyze the contribution of different frequencies to the overall signal. The frequency domain representation is often visualized using a graph called a frequency spectrum or power spectrum, which shows the amplitude or power of each frequency component.

The Fourier transform is an extension of the Fourier series for non-periodic or time-limited signals. It allows us to analyze signals with finite duration or non-repeating patterns. The Fourier transform converts a signal from the time domain to the frequency domain by representing it as a sum of complex exponential functions. The resulting representation provides information about the magnitude and phase of different frequency components.

After applying the Fourier transform to a function, we can then proceed to analyze and filter the data, using the values for the frequencies obtained.

Fourier analysis has numerous applications in different domains. It is used for signal-processing tasks such as filtering, compression, noise removal, and modulation. In image processing, Fourier analysis is employed in tasks like image enhancement, image reconstruction, and pattern recognition. It also has applications in physics (e.g., studying waves and oscillations), audio engineering, telecommunications, and many other fields.

A technique that is very useful for signal and image processing is the FFT, or *fast Fourier transform*. It is an efficient algorithm used to compute the Discrete Fourier Transform (DFT) of a sequence or signal.

Analogously to the continuous Fourier Transform, the DFT is a mathematical transformation that converts a discrete-time signal from the time domain into the frequency domain. It expresses the signal as a sum of complex sinusoidal components at different frequencies.

The FFT algorithm revolutionized the computation of the DFT by proving an efficient way to analyze the frequency content of signals. By enabling the transformation of time-domain signals into the frequency domain, it reveals important information about the signal's spectral components.

5.1 Ideal Band-pass Filtering

An Ideal Band-pass Filter is a *theoretical* filter, which allows a specific range of frequencies to pass through while attenuating all other frequencies to zero. It is characterized by having a perfectly flat *pass-band*, where the desired frequencies are passed without any distortion, and an infinitely sharp transition between the *pass-band* and *stop-band*.

The pass-band of an ideal filter refers to the range of frequencies that the filter allows to pass through without attenuation. It is usually defined by a lower cutoff frequency and an upper cutoff frequency.

The stop-band in an ideal filter includes all frequencies outside the pass-band. In the stop-band, the filter will reject all frequencies in this range.

$$B_I(\omega) = \begin{cases} 1 & \omega_H \leq |\omega| \leq \omega_L \\ 0 & otherwise \end{cases} \tag{2.5}$$

This filter is periodic and has a period of 2π. An example of this filter can be seen in Figure 2.3

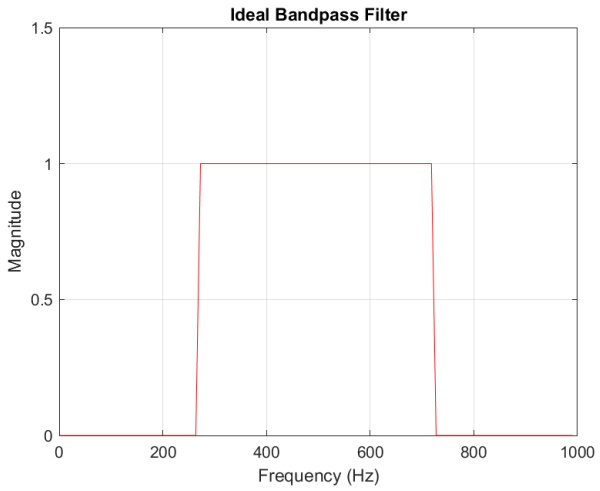


Figure 2.3: Example of a band-pass filter

6 First-order Taylor Expansion

The first-order Taylor expansion, also known as the **linear approximation**, is a method in calculus to approximate a function around a specific point using its derivative. It involves expressing a function as a linear combination of the function's value at a chosen point and its derivative at that point.

Mathematically, the first-order Taylor expansion at a function $f(x)$ around a point a is given by:

$$f(x) \approx f(a) + (x - a)\frac{\partial f(a)}{\partial a}$$

In this approximation, the function $f(x)$ is approximated by a linear function that consists of the function's value at the point a, and the product of the derivative of the function evaluated at a, $\frac{\delta f(a)}{\delta a}$, and the difference between the variable x and the point a, $(x - a)$.

The first-order Taylor expansion provides a good approximation of a function in the vicinity of a chosen point, particularly when the difference between x and a is small. It is a fundamental concept in calculus and serves as a building block for higher-order approximations and mathematical analysis.

7 Motion and Color Amplification

The world around us is always changing, even if we do not always notice. Plants grow ever so slowly, our muscles contract and expand microscopically when we move, our pupils dilate when the lighting in the environment changes slightly, and machines can wobble ever so slightly if there is a mechanical failure waiting to happen. Things that cannot be noticed with the naked eye.

With this in mind, a Ph.D. student from the Massachusetts Institute of Technology, Michael Rubinstein, and his team proposed a technique to amplify minimal changes that happen over time in videos. Their work focused on amplifying movements or objects of interest while removing distracting information using signal processing and filtering.[9]

By studying these techniques, we can understand how to process videos in a way that will help reveal hidden information, which can be used in real-life applications.

7.1 Introduction

Signal processing is used in many different areas of study, one of them being image and video processing. Some of the techniques that were devised to process videos are more complex than others, but a lot of them were so optimized that they are now commonly used in smartphone apps to enhance or edit videos.

In this section, I will be exploring three of these different techniques, all of which involve either removing or magnifying signals.

Removing Irrelevant Information

Time-lapse videos are videos that are recorded over a long period of time, with a low frame rate. When it is played, a higher frame rate is used, giving the viewer the impression that time is accelerated. It has many different applications in many industries, for example:

- In the entertainment industry, it can be used as a way to show the passage of time in movies.
- In the medical industry, one of its many uses is that it can be used as a way to monitor embryo development when they are being incubated for fertility treatments such as IVF. [1]
- In the agronomy industry, it can be used to monitor plant growth.

In many of those cases, lighting can suddenly change, or irrelevant objects might appear and disappear, which can make analyzing such videos much more difficult. This is what this technique proposes to eliminate.

The video processing system treats all short-term visual changes as noise, and long-term changes as a signal, which reveals the underlying long-term events. In other words, it automatically separates the short-term and long-term motion components in a video, making it easier to analyze what is actually relevant information.[9]

This is not the focus of this paper, but it is interesting to explain and analyze the techniques used to achieve this, so I will detail it more in further chapters.

Different Approaches

When a person breathes, their body undergoes slight movements, and their facial color changes in response to their heartbeat. However, these changes in motion and color are minimal, requiring the use of specialized techniques to extract and analyze this valuable information.

The *Eulerian* video magnification technique analyzes the color values of a pixel through time and amplifies frequency variations in a given band of interest.

There are two different techniques to extract this information. Both of them used *Eulerian specification* of the changes in the videos, amplifying the changes that would be seen in fixed locations in space.

Eulerian Specification is a term used in *continuum mechanics*, and refers to the analysis of changes that happens in a specific location of space through which fluid passes over time.

In other words, by using these techniques, we want to monitor and amplify the changes that happen around fixed locations in the scene i.e. pixels.

The first method we are going to discuss is **the linear method**. Spatial decomposition is first applied on a video, followed by temporal filtering. When the result of the temporal filter is amplified, hidden information is shown. This technique can be used to extract small hidden motions.

However, **the linear method** has two major drawbacks: the first one is that noise is amplified linearly with the motion, and the second is that this technique does not work with higher spatial sequences and large motions.

For this reason, a second method has been proposed, in which the linear approximation is replaced with a localized Fourier decomposition, using complex-valued pyramids. The phase variations of the Fourier coefficients over time can be attributed to motion, and we can modify them to manipulate motion.

Changes in the phase coefficient of the values in the pyramid over time correspond to motion, and we can process them to either remove unwanted noise or amplify them to reveal hidden motion.

Although it is more computationally expensive, higher ranges of motion are supported and less noise is produced.

Processing Videos in Space-time

If we fix our sights on a specific area of a video, and the color in it changes, it can mean one of two things. Either the color of the object depicted in that part of the image has changed, or the object in the region moved, and there is now another object in that area of the video or another part of the same object. These changes are what we want to amplify.

The process to amplify these signals goes as follows:

1. Decomposing the video into different spatial frequency bands.

   The magnification in all bands is not uniform, as they may each have a different signal-to-noise ratio, or the linear approximation may not be accurate. If the linear approximation is not accurate for a specific band, we want to reduce the amplification as much as possible to reduce the noise produced.

2. Temporal Processing on each spatial band.

   We examine the values of a pixel in a frequency band over time and apply a band-pass filter to extract the frequency bands of interest. The temporal processing is uniform in all spatial levels and for all pixels in each one.

   The signal that resulted from the band-pass filter is then multiplied by a magnification factor α, which can be attenuated.

3. Add magnified signal to the original signal and collapse the spatial pyramid to obtain the final output

But how does this temporal processing magnify motion? Even though we want to study the two-dimensional case, we can study the case of a one-dimensional signal, and generalize it.

We can write the image intensity I at position x and time t as a function of the current location and the displacement $\delta(t)$ it suffers through time:

$$I(x,t) = f(x + \delta(t)) \tag{2.6}$$

In which α is the magnification factor. We can rewrite this as an approximation of the first-order Taylor expansion about x:

$$I(x,t) = f(x + \delta(t)) \approx f(x) + \delta(t)\frac{\partial f(x)}{\partial x} \tag{2.7}$$

At the beginning of time, there is no displacement, so $\delta(t = 0) = 0$ and $I(x, 0) = f(x)$.

Our goal is to amplify the motion by the amplification factor α, and therefore, we want to acquire the signal:

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

If we apply a temporal band-pass filter to $I(x,t)$ at every position x, rejecting $f(x)$ at every point, we can write the resulting signal, $B(x,t)$ as:

$$B(x,t) = \delta(t)\frac{\partial f(x)}{\partial x} \tag{2.9}$$

As previously explained, the original signal is processed, and the band-pass filter is applied and amplified by a factor α. The resulting signal of the summation of both components is then:

$$\tilde{I}(x,t) = I(x,t) + \alpha B(x,t) \tag{2.10}$$

Substituting $B(x,t)$ for equation 2.9 and $I(x,t)$ as in equation 2.7 (after the first-order Taylor expansion), we get:

$$\tilde{I}(x,t) \approx f(x) + (1 + \alpha)\delta(t)\frac{\partial f(x)}{\partial x} \tag{2.11}$$

Once again, assuming that the first-order Taylor expansion holds in this case (for the amplified larger perturbation $(1 + \alpha)\delta$), we can obtain the signal that we wanted originally:

$$\tilde{I} \approx f(x + (1 + \alpha)\delta(t)) \tag{2.12}$$

Limits The approximations used to hold for smooth videos and small motions, but if the image functions change too fast (have high spatial frequencies), then we cannot assume that the first-order Taylor approximations hold because the value of the perturbation $(1+\alpha)\delta(t)$ becomes too large.

We can calculate the limits for which this solution works, in other words, for which ranges of values that the processed signal $\tilde{I}(x,t)$ (equation 2.11) is equal to the true magnified motion, $\hat{I}(x,t)$ (equation 2.8):

$$\tilde{(}x,t) \approx \hat{I}(x,t)$$
$$f(x) + (1+\alpha)\delta(t)\frac{\partial f(x)}{\partial x} \approx f(x + (1+\alpha)\delta(t)) \tag{2.13}$$

If $f(x) = cos(\omega x)$, for spatial frequency ω, and we call $1+\alpha$ β, then we can rewrite equation 2.13 as:

$$cos(\omega x) - \beta\omega\delta(t)sin(\omega x) \approx cos(\omega x + \beta\omega\delta(t)) \tag{2.14}$$

The additional law for cosines states that:

$$cos(\alpha+\beta) = cos(\alpha)cos(\beta) - sin(\alpha)sin(\beta)$$

Therefore, we can rewrite equation 2.14 as:

$$cos(\omega x) - \beta\omega\delta(t)sin(\omega x) \approx cos(\omega x)cos(\beta\omega\delta(t)) - sin(\omega x)sin(\beta\omega\delta(t)) \tag{2.15}$$

Therefore, we can assume that $\beta\omega\delta(t) \approx sin(\omega x)$ and $cos(\beta\omega\delta(t)) \approx 1$.

The small angle approximation rule states that for small angles, the function sine is approximately equal to its angle, or if θ is small enough then $sin(\theta) = \theta$.

The approximation holds for values within 10% of $\beta\omega\delta(t) \leq \frac{\pi}{4}$. We know that the wavelength λ is inversely dependent on the frequency of the signal ω and the $\lambda = \frac{2\pi}{\omega}$.

Remembering that $\beta = (1+\alpha)$, we can manipulate the inequality to obtain the limit for which the small signal approximation holds:

$$(1+\alpha)\delta(t) < \frac{\lambda}{8} \tag{2.16}$$

7.2 Obtaining Heart Rate and Pulse

A very common device used to extract heartbeat from a patient is the ECG, which measures the electrical activities of the person's heart [8]. This device requires electrodes to be attached to a person's skin, usually using sticky patches. Although effective, this type of device requires the sensors to have good contact with the patient's skin and can yield inaccurate measurements should they be poorly attached.

This means that the patient might have to shave his or her chest and his or her skin has to be free from oils and sweat. This is inconvenient, and therefore, it is interesting to consider other methods to extract a patient's heartbeat.

The technique of magnifying small changes in videos could be an interesting solution, depending on the context.

As stated earlier, a person's skin color changes slightly with blood flow. The signal corresponding to the blood flowing through a person's corresponds to a variation of roughly 0.5 intensity units in an 8-bit scale (with the pixel's values ranging from 0 to 255).

The process to extract the heartbeat from a person's face is as follows:

1. Spatial decomposition.

   Similarly to the process of amplifying small motions, this process begins with the decomposition of the video into different spatial frequency bands, but a Gaussian pyramid is used instead. The number of levels in the pyramid is usually limited to 3 or 4.

2. Filtering.

   Each level of the pyramid is temporally filtered, and we select frequency values that are compatible with the human heart rate, 0.4 to 4 Hz (or 24 to 240 beats per minute). Temporal filtering removes some of the noise.

3. Frequency Analysis.

   In order to detect the heartbeat rate, we can either:

   (a) Perform a frequency analysis of color variation in the video.

   (b) Detect the maxima points within a local window of time in the band-passed signal, detecting the pulse onset times at each location on the face in the coarse pyramid level.

   The quality of the pulse extraction can vary depending on the area of the face being used, and sometimes errors can occur when detecting the peaks in the band-passed signal.

To get a good estimation of when the pulse onset happens, we integrate the values of the signal extracted from different parts of the video, ignoring all points that deviate too much from the expected values of frequency.

We then consider that the points that were not ignored contain a pulse signal, and the pulse locations are now determined to be the local (temporal) centers of mass of the detected peaks in all the points.

The pulse estimation at particular points may be noisy, but the method previously stated of combining different estimations at different points can give a robust estimation of the pulse signal.

From the estimated pulse signal, the heartbeat rate can be established within a small temporal window.

8 Motion Denoising

In this section, we will explore a technique to remove "motion noise" from videos. As previously mentioned, motion noise can include random objects appearing and disappearing, sudden changes in lighting, or irrelevant information that come from a one-time event, such as wind.

Randomness can be a form of noise, and we might want to remove undesired random motions from our videos, such as jitters from the camera or the wind. In other words, we want to remove the temporally inconsistent motion.

The input of the system is a video $I(x, y, t)$ in the shape $[M \times N \times T]$, with RGB intensities in the interval $[0, 255]$. The desired output $J(x, y, t)$ also has the shape $[M \times N \times T]$.

8.1 Background

Temporal Filtering

A solution to temporal filtering is to pass the sequence of signals through a temporal low-pass filter:

$$J(x, y, t) = f(I(x, y, \{k\}_{t-\delta_t}^{t+\delta_t})) \tag{2.17}$$

f would be the "filter" operator, and δ_t would determine the temporal window size. If the video had a static viewpoint, a median operator would often be used as f , because it is good to reduce noise and separate background and foreground elements.

This approach is simple to implement and fast, but it has the downside of performing the filtering pixel-by-pixel. When applied to a moving scene, pixels that belong to different objects are averaged, and that can lead to a blurred or discontinuous output[9].

A strategy to try and fix the issue was devised: to filter along motion trajectories. In other words, to integrate pixels along their estimated motion path. This approach is useful in compressing video and removing noise, but it does not filter the actual motion, which can lead to unwanted artifacts and imperfect results.

Mathematical Justification

The mathematical model used for the *Eulerian* magnification of signals considers that the world evolves slowly with time. And because the changes happen so slowly, we can use the redundant data from the sequential frames to infer information about the static data in a video.

Considering the output video $J(x, y, t)$, and the input video $I(x, y, t)$, we want to minimize the energy given by equation 2.18:

$$E(J) = A + \alpha B \tag{2.18}$$

In which:

$$A = \sum_{x,y,t} |J(x,y,t) - I(x,y,t)| \tag{2.19}$$

$$B = \sum_{x,y,t} |J(x,y,t) - J(x,y,t+1)| \tag{2.20}$$

The energy $E(J)$ between the input and output videos can be described as a sum of the *fidelity term* (A, equation 2.19) and the *temporal cohesion term* (B, equation 2.20) multiplied by α.

The *fidelity term* describes the similarity between the two videos throughout their location and time, while the *temporal cohesion term* describes how temporally smooth the video is. α controls the trade-off between the two components.

We can describe the output video $J(x,y,t)$ as it is written in equation 2.21. $I(x,y,t)$ is the input video and $\omega(x,y,t)$ is the displacement field in time and space.

$$J(x,y,t) = I(x + \omega_x(x,y,t), y + \omega_y(x,y,t), t + \omega_t(x,y,t)) \tag{2.21}$$

As 2.21 is only described in terms of x, y, and t, we can parameterize $p = (x,y,t)$ and use it in equation 2.18, substituting p into $J(x,y,t)$, which becomes $J(p)$. Therefore, we would have:

$$E(\omega) = \sum_{p} |I(p + \omega(p)) - I(p)| + \tag{2.22}$$

$$\alpha \sum_{p,r \in \mathcal{N}_t(p)} ||I(p + \omega(p)) - I(r + \omega(r))||^2 + \tag{2.23}$$

$$\gamma \sum_{p,q \in \mathcal{N}_t(p)} \lambda_{pq} |\omega(p) - \omega(q)| \tag{2.24}$$

Equation 2.24 has not changed much from equation 2.18, but it has one key difference: now there is a term that is written in terms of the displacement window ω (equation 2.24), and it serves to smooth it. It is being multiplied by γ, which is the *regularization term*.

$\mathcal{N}(p)$ refers to the spatio-temporal neighborhood of the pixel p, $\mathcal{N}_s(p)$ refers to the spatial neighborhood of p, and $\mathcal{N}_t(p)$ refers to the temporal neighborhood of p. Using the six pixels that are directly connected to p, we can construct a neighborhood system.

Each element of the sum in equation 2.24 is multiplied by the weight $\lambda_{pq} = e^{-\beta||I(p)-I(q)||^2}$. β can be deduced using the method described by Szeliski et. al. in the paper "A Comparative Study of Energy Minimization Methods for Markov Random Fields with Smoothness-Based Priors"[11]. The weight factor λ_{pq} serves as a smoothing factor, and it assigns different weights to discontinuities in the displacement map, using the similarity between the surrounding pixels in the original video.

8.2 Optimization

Using a 3D Markov Random Field to represent the three-dimensional space of the video volume, we can optimize the equation 2.24. In the MRV, each node p corresponds to a pixel p in each video frame and represents the latent variables $\omega(p)$.

Latent variables are variables that are not directly observable or measurable but inferred based on their relationship with the observable variables. They are often used to simplify complex systems or explain patterns in data.

In the *Eulerian* video magnification model, the **state space** is defined as a collection of all potential three-dimensional displacements that can occur within a predefined search region.

The best optimization technique for this application is the Loopy Belief Propagation (LBP), because it produces more sequences that were more visually appealing and that achieved lower energy solutions compared to the other methods[9].

Chapter 3

Proposal

1 Introduction

The primary goal of this thesis is to investigate the potential use of physiological information extracted from video data within the framework of the polygraph test. The aim is to assess its effectiveness in distinguishing between truthful and deceptive responses from the subjects.

To accomplish that, I conducted an in-depth study on the topic of amplification of small signals in videos[9], to gain a comprehensive understanding of how to amplify subtle changes in motion and color within videos.

Using this information, I wanted to be able to detect the pulse and magnify breathing rates of subjects in videos, in order to explore how such techniques could be employed in real-life situations.

I also wanted to discuss the feasibility of these techniques in criminal investigations.

2 Objectives

The objectives I have set for this project were mainly:

1. **To gain a deep understanding of the process of magnifying small signals in videos**: Unquestionably, this step was of utmost significance, consuming a significant portion of my time and attention.

   Initially, I intended to reconstruct the open-source Matlab code-shared by Rubinstein using Python as a means to gain a hands-on comprehension of the practical implementation. However, I encountered significant challenges during this endeavor. Consequently, after a few weeks, I decided to discontinue that approach and shifted my focus toward studying the original code and paper in detail.

2. **To use the available code in videos I took myself and of real-life suspects.**

   Although it may seem straightforward, I faced significant obstacles during this phase. It was quite challenging to determine the exact settings and functions to use. I conducted numerous exhaustive tests, trying out various combinations of parameters. The difficulty increased when I attempted to apply these methods to a real-life video of a suspect who was talking and moving.

   However, these endeavors did produce interesting results, which will be discussed in more detail in later chapters.

3. **To study how polygraphs work and their validity**

   Although not the most critical aspect of my research, this step was still highly engaging. It provided valuable insights into the importance of the polygraph test, despite the ongoing debates surrounding its reliability. It also highlighted its continued usage as a crime-fighting tool in different parts of the world.

3 Methodology

This thesis was mainly focused on researching and understanding different concepts. It was comprised of two main parts:

1. Research.

   As previously stated, the first part was the most time-consuming for me. It required me to read and reread different articles, in particular articles about different motion magnification techniques.

   I also had to study foreign and forgotten concepts used in the articles, read some of the referenced works, and take many notes.

2. Understanding and testing code.

   One of the main sources of inspiration for this project was the open-source Motion Magnification project provided by the Massachusetts Institute Of Technology[7].

   In order to fully understand what was done by them, and how the theory translated into practice, I decided to attempt to implement it into Python from scratch. However, I did not realize how much of a grueling task it would be. My code ended up having too many bugs, and in the end, I could not make it work.

   I, unfortunately, realized it was against my better judgment to proceed with the development of the Python application because I would not have had enough time to complete my thesis had I done so.

   After my failed attempt at implementing my solution in Python, I decided it was a better use of my time to work with what was already available.

   For that reason, I used the open-source code I mentioned previously. Altering the code to better suit my needs, I initially tested it as-was using the videos for which the best parameters were already known.

   After I made it work as expected, I started working with the videos I sourced. I understood the theory, but it was initially very difficult to find the ideal combination of parameters to magnify colors.

   I did eventually learn how to properly quantify the necessary parameters to process the videos in the way that I wanted.

   The results I obtained will be discussed in later chapters.

Chapter 4

Implementation Details

1 Python: Initial Attempts

As previously stated, I initially tried to translate the original code into Python. Even though it ended up not yielding the desired results, I think it is still relevant to discuss the structure of the code I developed to properly understand the most important parts and functionalities of the original code.

My purpose behind this "translation" was to better understand what was being done. Therefore, I tried to simplify the original code as much as I could, which might have been what made my version of the code not work in the first place.

Even if it does not work as it should have, the logic behind the operations being applied is correct, and the simplification makes it easier to understand what is being done and what the intentions behind each function in Rubinstein's code were.

For this reason, I will explain my version of the code, so you might notice some dissimilarities in the order of some operations or how they are being applied.

To properly implement the functions I needed, I used the libraries OpenCV, NumPy, and SciPy. I separated the different functions into different files according to their purpose.

I developed the class `video`, to simplify the process of importing videos and processing their frames. I developed two methods of initializing a `video` object, I could either import an existing video from a path, or I could create a video from a stack of images.

Other than that, the `video` class had the `disp` method, which would display the video until a key was pressed, and the `getFrames` method, which would extract the frames from the video, saving them in an attribute of the class to save some time if I wanted to perform more than one operation on the same video.

The class `video` can be described as follows, and it is found in the file `video.py`:

```
class video():
    frames = []
    def __init__(self, path = "", stack = None):
        self.path = path
        if stack != None:
            self.width = len(stack[1])
            self.height = len(stack[2])
            self.fps = 30
            self.frames = np.asarray(stack)
            self.len = len(self.frames)

            self.displayInfo()
        else:
            cap = cv2.VideoCapture(self.path)
            self.width = cap.get(cv2.CAP_PROP_FRAME_WIDTH)
            self.height = cap.get(cv2.CAP_PROP_FRAME_HEIGHT)
            self.fps = cap.get(cv2.CAP_PROP_FPS)
            self.len = cap.get(cv2.CAP_PROP_FRAME_COUNT)
```

```
        self.getFrames()
    def displayInfo(self):
        print("Length: ",self.len)
        print("Width: ",self.width)
        print("Height: ",self.height)
        print("# Frames: ",len(self.frames))
    def export(self, new_file_name=''):
        if(new_file_name==''):
            new_file_name=self.path

        height, width = self.frames[0].shape[:2]
        result = cv2.VideoWriter('./results/' + new_file_name,
                                 cv2.VideoWriter_fourcc(*'DIVX'),
                                 self.fps,
                                 (width, height))

        for f in self.frames:
            result.write(f.astype('uint8'))

        result.release()
    def disp(self):
        cap = cv2.VideoCapture(self.path)
        if(cap.isOpened == False):
            print("Error opening video!")
        while(cap.isOpened()):
            ret, frame = cap.read()
            if ret == True:
                cv2.imshow('Frame',frame)
                if cv2.waitKey(25) & 0xFF == ord('q'):
                    break
            else:
                break
        cap.release()
        cv2.destroyAllWindows()
    def getFrames(self):
        cap = cv2.VideoCapture(self.path)
        count = 0
        self.frames = []
        while(cap.isOpened()):
            ret, frame = cap.read()
            count +=1
            if ret == True:
                self.frames.append(frame)
                if count == self.len:
                    break
            else:
                break
        cap.release()
```

Color Magnification The color magnification function, `colorAmplification`, takes as an input an object of the class `video`, the magnification factor α, the band-pass range, the sampling rate, and the attenuation factor λ. It uses that information to build the Gaussian stack and then applies the band-pass filter to each item of the stack. The stack is of size $L \times N \times M \times 3$, with L being the number of levels, and N and M corresponding to the dimensions of each frame (3 is the number of color channels). After that, the filtered stack is multiplied by the magnification and attenuation factor and then added to the original signal. This is all done in the YIQ color space. A diagram showing the

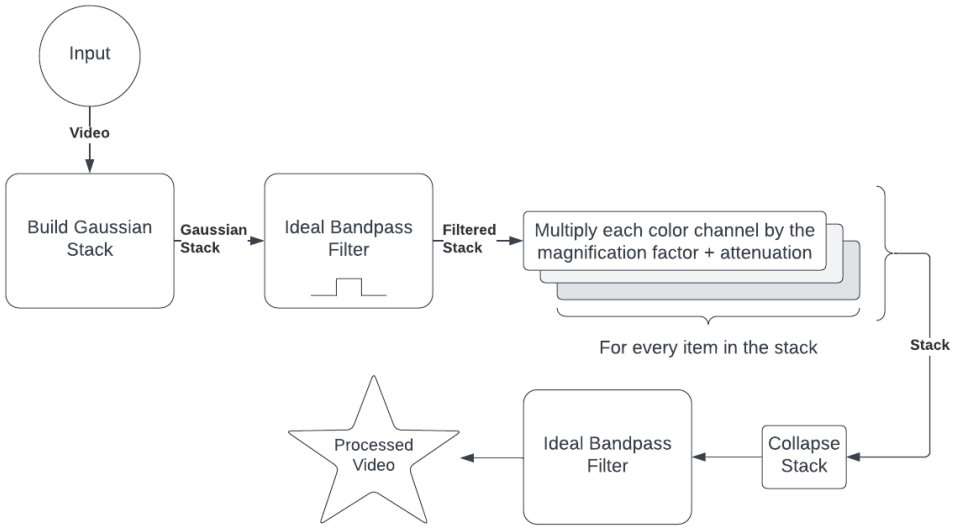


Figure 4.1: Diagram illustrating the process of magnifying the colors in a video.

process can be observed in Figure 4.1

The code for the color amplification function is in the file `colorAmplification.py` and it goes as follows:

```
# colorAmplification()
# Inpired by the function amplify_spatial_Gdown_temporal_ideal
# Applies Gaussian blur as the spatial filtering, followed by an ideal bandpass filter
# The idea is to amplify color variations, I.E. from this we will extract heart rate.
# alpha: amplification factor
# level: How many levels are in our Gaussian Decomposition stack?
# bandpassRange: [wLow, wHigh] range in which our bandpass filter will operate
# chromAttenuation: attenuate color?
def colorAmplification(video, alpha, level, bandpassRange, samplingRate, chromAttenuation):
    temporalWindow = [0, int(video.len)]

    print("Spatial Filtering...")
    gaussianStack = stackBuilders.buildGaussianStack(video, temporalWindow, level)
    print("Done!")

    print("Temporal filtering...")
    filteredStack = filters.idealBandPassing(gaussianStack, bandpassRange[0], bandpassRange[1],
    ↪  samplingRate)
    print("Done!")

    stack = []

    print(len(filteredStack))
    for i in range(len(filteredStack)):

        filtered = filteredStack[i].squeeze()

        filtered[:,:,0] = filtered[:,:,0] * alpha
        filtered[:,:,1] = filtered[:,:,1] * alpha * chromAttenuation
        filtered[:,:,2] = filtered[:,:,2] * alpha * chromAttenuation

        f = cv2.resize(filtered,(int(video.width), int(video.height)))
```

```
        f = f + lib.rgb2ntsc(video.frames[i])

        f = lib.ntsc2rgb(f)

        f = lib.normalizedImage(f)

        f[f<0] = 0
        f[f>255] = 255

        stack.append(f)
    v = vid.video(stack=stack,path="face.avi")
    v.export()
```

Building the Gaussian Stack The function that builds the Gaussian Stack is called `buildGaussianStack`, and it takes as an input a `video` object, the temporal window, and the number of levels we want in our Gaussian Stack. After that, it converts each frame of the video in the temporal window to the YIQ color space and applies the function `blurDownsample` to each one separately. The result of this operation is added to the output stack.

I also created the class `buildLaplacianStack`, which would, as the name suggests, create a Laplacian Pyramid based on the inputted video.

The code for the two functions is:

```
# buildGaussianStack()
# Build gaussian pyramid with level layers
# video: video class object
# temporalWindow: How big of a timeframe (in the video) are we taking
# in consideration?
# level: how many layers in our laplacian decomposition pyramid?
def buildGaussianStack(video, temporalWindow, level):
    stack = []
    for i in range(temporalWindow[0], temporalWindow[1]):
        image = lib.rgb2ntsc(video.frames[i])
        stack.append(filters.blurDownsample_(image, level))
    return stack
    # cv2.destroyAllWindows()

#buildLaplacianStack()
# Build Laplacian Pyramid. The second dimension is the color channel,
# The third dimension is time.
# - video
# - window [startIndex, endIndex]
def buildLaplacianStack(video, window, height=None):
    stack = []
    stackId = []
    for i in range(window[0], window[1]):
        image = lib.rgb2ntsc(video.frames[i])
        lapl, idx = laplacian.buildLaplacian_(np.array(image))
        stack.append(lapl)
        stackId.append(idx)
    return stack, stackId
```

Blurring and Down-sampling The function `blurDownsample` takes three different inputs: the image to be processed (`image`), the number of levels in the Gaussian pyramid (`levels`), and the filter

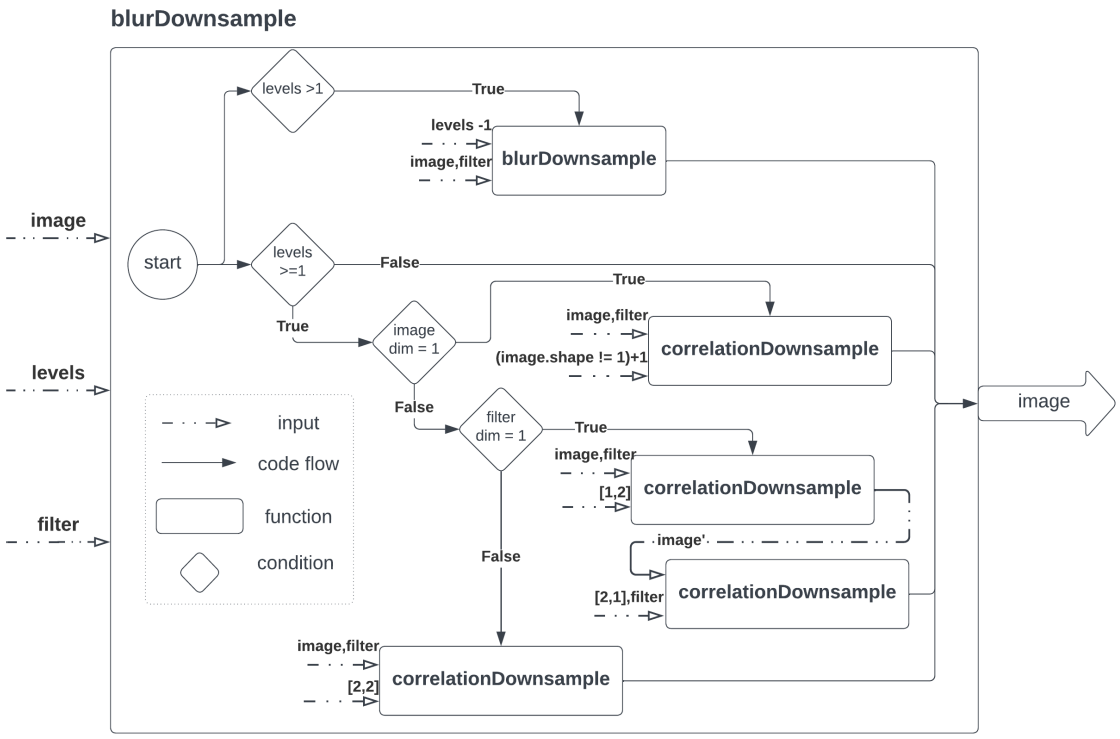


Figure 4.2: The function `blurDownsample` takes `image`, `levels` and `filter` as an input. For simplicity reasons, the process of filter selection is not depicted, but it depends on whether `filter` is a string or not. The dotted lines represent inputs and the solid lines the flow of the code. The diamonds represent conditions and the rectangles represent functions.

to be used to blur the image (`filter`]). This function is recursive, and its stopping condition depends on the `levels` variable.

It initially verifies whether the `filter` variable is a string, if so, the program will try to find a filter with that name and set it as the value of `filter`. If it is not a string, it will normalize it by dividing each member of the filter by the sum of all its members.

After that, it verifies if `levels`>1. If `True`, the function will recursively call `blurDownsample` with `levels = levels - 1`, with the same image, and updated `filter`.

Afterward, the function verifies if `levels`≥ 1, if it is `False` (i.e. if `levels == 0`, the stopping condition), then it will return the input `image`. If `True`, then it will verify the size of the image or filter, and call the function `correlationDownsample`, which will be described in the following section. If the image is 1 dimensional, the down-sampling step will be 1, otherwise, it will be 2. If the filter is one dimensional, the program will first perform down-sampling with a step value of `[1,2]` and then again with a step value of `[2,1]`. If both the image and the filter are two-dimensional, then the program will perform down-sampling with step `[2,2]`.

This process is described in Figure 4.2 and in the code below. The function `blurDownsample` is in the file `filters.py`.

```
# blurDownsample()
# Recursively blurs and downsamples the image levels times.
# The downsampling is always done by 2 in each direction.

def blurDownsample_(image, levels,filter='binom5'):
    tmp = blurDownsample(image[:,:,1], levels)
    out = np.zeros((tmp.shape[0],tmp.shape[1],image.shape[2]))
    out[:,:,0] = tmp
```

```
    out[:,:,1] = blurDownsample(image[:,:,1], levels)
    out[:,:,2] = blurDownsample(image[:,:,2], levels)

    return out

def blurDownsample(image, levels, filter = 'binom5'):
    if(isinstance(filter, str)):
        filt = getFilter(filter)
    else:
        filt = filter
    filt = filt/filt.sum()

    if levels > 1:
        image = blurDownsample(image, levels - 1, filt)


    if levels >= 1:
        if(image.shape[0] == 1 or image.shape[1] == 1):
            return correlationDownsample(image,
                                         filt,
                                         ((image.shape[0]!=1) * 1+1,(image.shape[1]!=1) * 1 +1))
        elif(len(filt.shape)==1):
            return correlationDownsample(correlationDownsample(image, filt, [2,1],axis=1),
                                         filt,
                                         [1,2])
        else:
            return correlationDownsample(image,
                                         filt,
                                         [2,2])
    else:
        return image
```

Correlation and Down-sampling The function `correlationDownsample` receives as an input the `image` we want to process, the `filter` it is going to be correlated with, the `axis` the correlation is going to be done in, and the last input,`step`, determines how much down-sampling is done.

The program first determines if the filter is one-dimensional. If so, it will apply SciPy's `correlate1d` function to the image in the given `axis`.

If the filter is two-dimensional, it inverts the order of the elements of the filter in both axes and applies the function `convolve` from the SciPy library to the image.

The resulting array from either operation is then down-sampled according to `step` and returned.

The code below can be found in the file `filters.py`.

```
# correlationDownsample()
#  Calculates the cross-correlation between the image and the filter.
#  filter: which filter to filter with
#  step: how much to downsample
#  window_start and window_stop: determine the window in which the image will be filtered
def correlationDownsample(image,
                          filter,
                          step = [2,2],
                          window_stop = (-1,-1),
                          window_start = (0,0),
                          axis=0):
    if(window_stop == (-1,-1)):
        window_stop = (image.shape[0], image.shape[1])
```

```
    if len(filter.shape) == 1:
        # filter = filter[len(filter)-1:0:-1]
        if(axis==1):
            image = scipy.ndimage.correlate1d(image*1.0, filter,axis=0)
        else:
            image = scipy.ndimage.correlate1d(image*1.0, filter,axis=1)
    else:
        filter = filter[len(filter)-1:0:-1, len(filter)-1:0:-1]
        image = scipy.ndimage.convolve(1.0*image, filter,mode='reflect')

    return image[window_start[0]:window_stop[0]:step[0], window_start[1]:window_stop[1]:step[1]]
```

Ideal Band-passing As previously explained in Section 5.1, the ideal band-pass filter is a theoretical filter that blocks out all signals outside of a specific frequency range.

The `idealBandPassing` function receives as input the signal to be filtered (`input`), the lower (`wLow`) and upper (`wUpper`) bound of the pass-band, the sampling rate and the number of dimensions to shift the `input`.

The program first shifts the dimensions of `input` by `dim`.

It then determines the temporal frequencies of the input's Fourier Transform, using the number of dimensions of the shifted input n, and the sampling rate. If we call the sampling rate s, we can write that the frequencies for this input are as described in equation 4.1.

$$freq = \frac{[1, 2, 3, ..., n-1, n]}{n * s} \tag{4.1}$$

After that, it creates a mask by replicating the array `freq` until the mask has the same number of dimensions as the shifted input.

The program then will set the value of the elements in the mask to 1 or 0 according to whether they are within the boundaries of the pass-band or not.

Following this operation, the program will perform an FFT on the shifted input, and then use the mask previously calculated to set all values outside of the transformed range of the transformed shifted input to 0.

Finally, we perform an IFFT and return all real values.

The code below can be found in the file `filters.py`.

```
# idealBandPassing:
# Applies ideal bandpass filter on input
# wLow: lower cutoff region
# wUpper: upper cutoff region

def idealBandPassing(input, wLow, wUpper, samplingRate,dim=1):
    f = np.roll(input,dim-1)
    input = np.asarray(f)

    dimensions = list(f.shape)

    n = dimensions[0]
    dn = len(dimensions)

    freq = (np.linspace(1, n, n) - 1)/n*samplingRate
    if(len(dimensions) == 4):
        mask = np.asarray((freq > wLow) & (freq <
        ↪  wUpper))[np.newaxis][np.newaxis][np.newaxis].transpose()
    elif(len(dimensions) == 3):
```

```
        mask = np.asarray((freq > wLow) & (freq < wUpper))[np.newaxis][np.newaxis].transpose()
    else:
        return
    dimensions[0] = 1

    mask = np.tile(mask, dimensions)

    f = scipy.fft.fft(f,axis=0)


    f[~mask] = 0

    out = np.real(scipy.fft.ifft(f,axis=0))

    return out
```

2 Matlab

As previously stated, I could not make my code work, despite my best efforts. For this reason, I decided to use the Matlab code provided by Rubinstein to magnify motions and colors.

The function used to magnify motions and colors is `amplify_spatial_Gdown_temporal_ideal`, which does the process I described in Section 1. First decomposing the signal into a Gaussian pyramid before filtering and summing the results to the original video.

This function loads the video whose path was given and processes it, saving the result where the user specifies.

The code I used to generate the videos can be found in the file `getResults.mlx`, and the necessary videos to run it can be found in the folder "data_".

If these folders do not exist yet, the program creates a folder with the current date and time inside the folder `results` (if they do not exist yet). It then runs the aforementioned function with the given parameters.

```
clear;

dataDir = './data';
resultsDir = 'results/' + string( datetime('now','TimeZone','local','Format','d_MM_HH'));

mkdir(resultsDir);

% Jodi Arias (Rest)
video = fullfile('data_/jodi_rest_sq.mp4');

amplify_spatial_Gdown_temporal_ideal(video, resultsDir, 120, 7, 65/60, 70/60, 30, 0.9);

% Jodi Arias (Lie)
video = fullfile('data_/jodilie.mp4');

amplify_spatial_Gdown_temporal_ideal(video,resultsDir, 50, 7, 70/60, 80/60, 30, 0.9);

% Me
video = fullfile("data_/me6.mp4");

amplify_spatial_Gdown_temporal_ideal(120, 5, 70/60, 80/60, 30, 1)
```

Another function I have developed for this application was the `heartbeat` function, which reads data from a video and returns an array with the average value of each frame. Using this function, we can determine when the peaks in coloration happen in a color-magnified video.

Using that information, the number of frames, and the video's frame rate, we can have a good idea of what the person's heartbeat rate is.

The `heartbeat function` is as described in the code below:

```
function avg = heartbeat(vidFile)
    [~,vidName] = fileparts(vidFile);

    % Read video
    vid = VideoReader(vidFile);
    % Extract video info
    vidHeight = vid.Height;
    vidWidth = vid.Width;
    nChannels = 3;
    fr = vid.FrameRate;
    len = vid.NumberOfFrames;
    temp = struct('cdata', zeros(vidHeight, vidWidth, nChannels, 'uint8'), 'colormap', []);

    startIndex = 1;
    endIndex = len-10;

    %% Render on the input video

    % output video
    k = 0;

    avg = zeros(1, endIndex-startIndex);
    for i=startIndex+1:endIndex
        k = k+1;
        temp.cdata = read(vid, i);
        [rgbframe,~] = frame2im(temp);
        rgbframe = im2double(rgbframe);

        avg(k) = sum(rgbframe(:))/(size(rgbframe,1) * size(rgbframe,2));


    end


%     close(vidOut);

end
```

To generate the image with the spatiotemporal representation of the video, I created the function `ytslice`. This function extracts the array of pixels in the center of each frame and returns an array with the same height as the original video, but the width equal to the number of frames.

Its code is:

```
function out = ytslice(vidFile)
    [~,vidName] = fileparts(vidFile);

    % Read video
    vid = VideoReader(vidFile);
    % Extract video info
    vidHeight = vid.Height;
    vidWidth = vid.Width;
    nChannels = 3;
    fr = vid.FrameRate;
    len = vid.NumberOfFrames;
    temp = struct('cdata', zeros(vidHeight, vidWidth, nChannels, 'uint8'), 'colormap', []);
```

```
    startIndex = 1;
    endIndex = len-10;

    %% Render on the input video

    % output video
    k = 0;

    out = zeros(vid.Height,len-10,3);
    center = floor(vidHeight/2);
    for i=startIndex+1:endIndex
        k = k+1;
        temp.cdata = read(vid, i);
        [rgbframe,~] = frame2im(temp);
        rgbframe = im2double(rgbframe);

        out(:,i,:) = rgbframe(:,center,:);

    end
end
```

The script I used to generate the graphs used in Chapter 6 is given by the code below, and it can be found in the file `generateResults.mlx`:

```
% ===================
%% Face
vidfile = "./data_/face.mp4";
vidfile2 = "./results_/face-ideal-from-0.83333-to-1-alpha-100-level-4-chromAtn-1.avi";
a = heartbeat(vidfile);
b = heartbeat(vidfile2);

p = plot(a(1:end-10) - b,"-");
title("Intensity Face (Amplified Subtracted from Original)")
xlim([1,length(b)])
xlabel("Time")
ylabel("Intensity")

plot(a, '-');
title("Intensity Face (Original)")
xlim([1,length(a)])
xlabel("Time")
ylabel("Intensity")

plot(b, '-');
title("Intensity Face (Amplified)")
xlim([1,length(b)])
xlabel("Time")
ylabel("Intensity")

y = ytslice(vidfile);
p = imshow(y);
title("Face Original (YT Slice)")
ylim("auto")

y = ytslice(vidfile2);
p = imshow(y);
title("Face Amplified (YT Slice)")
```

```
ylim("auto")

% =================
%% Me
vidfile = "./data_/me6.mp4";
vidfile2 = "./results_/me6-ideal-from-1.1667-to-1.3333-alpha-120-level-5-chromAtn-1.avi";
a = heartbeat(vidfile);
b = heartbeat(vidfile2);

p = plot(a(1:end-10) - b,"-");
title("Intensity Me (Amplified Subtracted from Original)")
xlim([1,length(b)])
xlabel("Time")
ylabel("Intensity")

plot(a, '-');
title("Intensity Me (Original)")
xlim([1,length(a)])
xlabel("Time")
ylabel("Intensity")

plot(b, '-');
title("Intensity Me (Amplified)")
xlim([1,length(b)])
xlabel("Time")
ylabel("Intensity")

y = ytslice(vidfile);
p = imshow(y);
title("Me Original (YT Slice)")
ylim("auto")

y = ytslice(vidfile2);
p = imshow(y);
title("Me Amplified (YT Slice)")
ylim("auto")

% =================
% Jodi Rest
vidfile = "./data_/jodi_rest_sq.mp4";
vidfile2 = "./results_/jodi_rest_sq-ideal-from-1.0833-to-1.1667-alpha-120-level-7-chromAtn-0.9.avi";
a = heartbeat(vidfile);
b = heartbeat(vidfile2);

p = plot(a(1:end-10) - b,"-");
title("Intensity Jodi Rest (Amplified Subtracted from Original)")
xlim([1,length(b)])
xlabel("Time")
ylabel("Intensity")

plot(a, '-');
title("Intensity Jodi Rest (Original)")
xlim([1,length(a)])
xlabel("Time")
ylabel("Intensity")

plot(b, '-');
title("Intensity Jodi Rest (Amplified)")
xlim([1,length(b)])
```

```
xlabel("Time")
ylabel("Intensity")

y = ytslice(vidfile);
p = imshow(y);
title("Jodi Rest Original (YT Slice)")
ylim("auto")

y = ytslice(vidfile2);
p = imshow(y);
title("Jodi Rest Amplified (YT Slice)")
ylim("auto")

% =================
% Jodi Lie
vidfile = "./data_/jodilie.mp4";
vidfile2 = "./results_/jodilie-ideal-from-1.1667-to-1.3333-alpha-50-level-7-chromAtn-0.9.avi";
a = heartbeat(vidfile);
b = heartbeat(vidfile2);

p = plot(a(1:end-10) - b,"-");
title("Intensity Jodi Lie (Amplified Subtracted from Original)")
xlim([1,length(b)])
xlabel("Time")
ylabel("Intensity")

plot(a, '-');
title("Intensity Jodi Lie (Original)")
xlim([1,length(a)])
xlabel("Time")
ylabel("Intensity")

plot(b, '-');
title("Intensity Jodi Lie (Amplified)")
xlim([1,length(b)])
xlabel("Time")
ylabel("Intensity")

y = ytslice(vidfile);
p = imshow(y);
title("Jodi Lie Original (YT Slice)")
ylim("auto")

y = ytslice(vidfile2);
p = imshow(y);
title("Jodi Lie Amplified (YT Slice)")
ylim("auto")
```

Note The code used in this project can be found in this GitHub repository: Link

Chapter 5

Analysis of Sustainability and Ethical Implications

As with any new project idea or innovation, we must discuss the ethical and sustainability impact that the realization of this project could have. Because the idea proposed by this thesis would mainly be implemented through software, it is not relevant to discuss any kind of environmental impact, especially because no physical resources are used (except for electricity).

That being said, the economic and social implications are still important points to be explored.

There is no political or religious agenda associated with this project, and no intention to oppress, violate any rights or harm any individual, or institution in any way.

1 Economic Implications

The only costs associated with the development and execution of this thesis would be human resources. In order to reduce the amount of work I would have to do, it was necessary to read other people's previous work, so as to be able to add to that and not commit the same mistakes that they might have.

If 1 ECTS is equivalent to roughly 25 hours of work, the master's thesis is worth 30 ECTS, assuming I have worked exactly $30 \times 25 = 750$ hours, if the minimum wage in Spain is roughly 8 euros/hour, then the cost to develop this project was of approximately of $750 \times 8 = 6000$ euros.

A lot more work would have to be done to develop a product that could be commercialized, such as the development of a GUI, or being able to extract information from a live video. To implement all of that, I estimate that I would have to work for approximately an extra month. The minimum wage per month is 1080 euros. Therefore the cost associated with such a project would be around 7080 euros.

After the product was finished, though, the cost associated with maintaining it would be relatively low. There would be no need for constant updates or adjustments to the code, and the cost of running it would not be much greater than running a regular application on a computer.

Other projects could expand on the results obtained in this thesis, other researchers could implement the features I mentioned that are lacking for a final project, or they could simply use the analysis I have done to gain more understanding of the work they need to do.

2 Social Implications

The polygraph itself is a controversial tool. Even though it is widely used in many countries around the world, it is still cause for debate.

The lie detector test works by analyzing spikes of activity in physiological data, such as heartbeat rate, respiratory activity, and galvanic skin response (GSR, or electrodermal activity), in order to determine if a person is lying or not.

However, because we are measuring the physiological responses of a person, any number of factors can contribute to skew the results[10], including factors such as:

1. Anticipation of a threat or activity,

2. Increased physical or mental activity,

3. Voluntary control of respiration, which affects both GSR and heartbeat rate,

4. Changes in the concentration of carbon dioxide and oxygen in the air,

For this reason, they are not fool-proof, and if not properly administered, they can lead to people being unfairly charged with a crime they did not commit.

Also, if we were to further extend the project by extracting the respiration rate from the suspect using video, we would have most of the pieces necessary to build a functioning polygraph.

However, this could be used by persons with bad intentions or misinformed people to try and detect lies in people that might not know that they are being evaluated, which might compromise the validity of the test, leading to faulty conclusions and innocent people being arrested or unfairly accused of something they did not commit.

If this project is ever created as a finished product, it is very important that the people using it are well-trained, capable individuals, and that they are properly instructed on how to properly use this tool.

The program works by augmenting the changes in the color of people's skin, and it works with people of different ethnicities and skin colors. It makes no distinction of gender, sex, age, or any other factor.

If properly developed into a final product, this project could be used by smaller police departments and private investigators, as they would only need to obtain a good enough video camera and the GSR sensor, having no need to acquire the sensor to measure heartbeat rate, the blood pressure cuff or the sensors that measure respiration.

3 Relationship with the Sustainable Development Goals

It project may contribute to goal 9.5 of the Sustainable Development Goals defined by the United Nations in 2015.

This goal is defined by: *"9.5 Enhance scientific research, upgrade the technological capabilities of industrial sectors in all countries, in particular developing countries, including, by 2030, encouraging innovation and substantially increasing the number of research and development workers per 1 million people and public and private research and development spending"*

It contributes towards this goal by suggesting an improvement of an existing tool and shedding light on how the use of a new technique can be used in real-life situations to improve the way that interrogations are conducted, in a way that can make them more comfortable and less intimidating for the person being investigated.

Chapter 6

Results

1 Introduction

The primary focus of this study was, as previously stated, research. To ascertain the viability of using the color magnification techniques to extract a person's heartbeat rate, instead of using more traditional methods.

In order to do this, I had to test Rubinstein's code with different videos. Testing with the videos and parameters provided did indeed yield the expected results.

The video where the magnification of the pulse could be seen more clearly was called "face" (a still frame of the video can be observed in Figure 6.1). Using $\alpha = 100$, $\omega_L = 50/60$, $\omega_H = 60/60$, 30 fps, and 4 levels, I obtained the expected results. Figure 6.2 depicts two different spatiotemporal representations of videos. Figure 6.2a represents the video before it was color-amplified, while Figure 6.2b represents the video after it was color-amplified.

In order to obtain this representation, I extracted the central vertical set of pixels from each frame of the video and joined them all together, so each vertical line in the resulting picture represents one frame, with time moving from left to right.

In Figure 6.2b, the vertical red lines appeared whenever blood flowed to the subject's face (making it red). It is clear to see that the color magnification effect was done successfully.

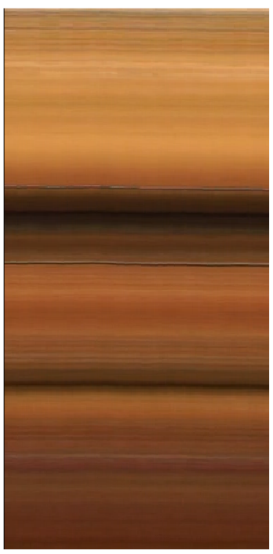
Figure 6.2c and Figure 6.2d are graphs representing the peaks of intensity over time in the video. When comparing the spatiotemporal representations of the videos with the graphs, we can clearly see that when the person's heart beats, the intensity of the image increases.

I calculated these values by simply computing the average value of each frame over time. Even though it is not the most robust method, since I am mostly using videos focused on the subject's face and with unchanging backgrounds, this is still a valid way to determine the peaks of intensity in videos.

Despite obtaining success in replicating Rubinstein's results, the same did not happen when I initially tried to magnify colors in my own videos. Indeed, when using the recommended parameters, my videos would be too noisy and I struggled to find the correct parameters that would only magnify color and not motion.

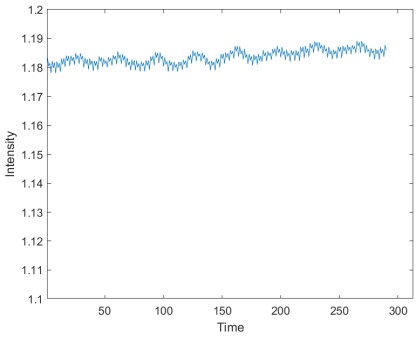
Although theoretically, we would need to magnify motion and color to implement a functioning

Figure 6.1: A still frame of the *face* video.

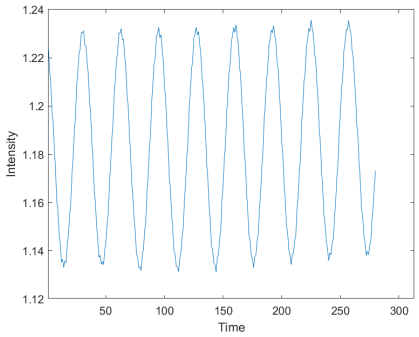


(a) Spatio-temporal representation of original *face* video.

(b) Spatio-temporal representation of processed *face* video.



(c) Graph representing the peaks of intensity in the original *face* video over time.



(d) Graph representing the peaks of intensity in the processed *face* video over time.

Figure 6.2

Figure 6.3: A still frame of the *me6* video.

polygraph test, I thought it was a better idea to try and focus on one of them at a time, and I thought that the color magnification aspect would be easier.

Human heartbeat rates tend to be from 60 to 100 bpm. To convert beatings-per-minute to Hertz, we have to divide the number in bpm by 60. Therefore, in order to extract the heartbeat rate of a person from a video, we want to amplify the changes in color in the temporal frequency of 1 to 1.67 Hz, approximately.

I have, however, also tested the outcome when using different values of frequency and will share my results in the following sections.

2 Results For Static Subjects

To try and amplify color in a subject that is not moving, I have filmed a few clips of myself trying not to move. However, even when I was trying my best not to move, the natural head bobbing that occurs when one breathes became perceptible when I magnified the changes in the video. The video that I am using as an example here is called *me6*, it only shows the subject in a white background. A still frame of the *me6* video can be seen in Figure 6.3.

In Figure 6.4, we can observe the results of the color magnification. For this case, I used the parameters $\alpha = 120$, $\omega_L = 70/60$, $\omega_H = 80/60$ and a Gaussian pyramid with 5 levels. The graphs depicting the intensity of the image over time can be seen in Figures 6.4c and 6.4d.

Although the results were not as obvious as in the *face* example, one can still see the vertical lines of darker color in the processed video.

As previously stated, the metric I am using is not the most robust. However, if we count the peaks of intensity in the graph in Figure 6.4d, we can more or less estimate the person's heartbeat rate. The video *me6* is approximately 15 seconds long, and we can count 13 peaks. If every peak is one heartbeat, then we have 13 beats in 15 seconds, and therefore the heartbeat rate in bpm would be around 52bpm.

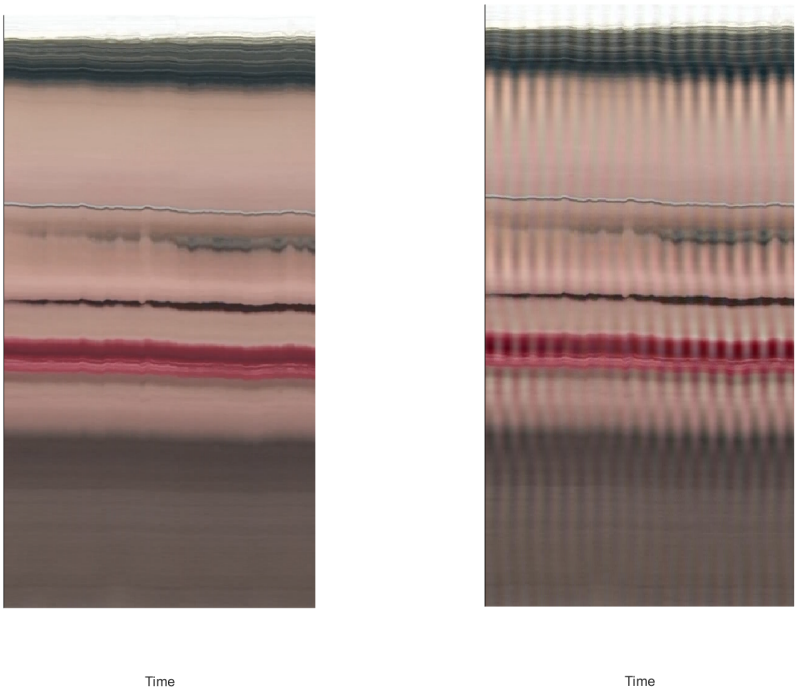
3 Results For Real-life Subjects

The previous two examples were of subjects that were as still as possible, but if we want to consider whether this method could be appropriate to measure a person's heartbeat rate during an interrogation, then we should test whether this method would still be a valid possibility if the subject is talking or blinking.

I used two videos to assess how plausible it would be to use this technique, as is, in a real-life situation. Both videos were extracted from an interview Jodi Arias gave to Fox News, right after she was convicted of murder in the first degree. Both clips are very short, each around 10 seconds long. The first one is of Arias blinking and not saying anything, and the second one is of her lying, so she is talking and moving.

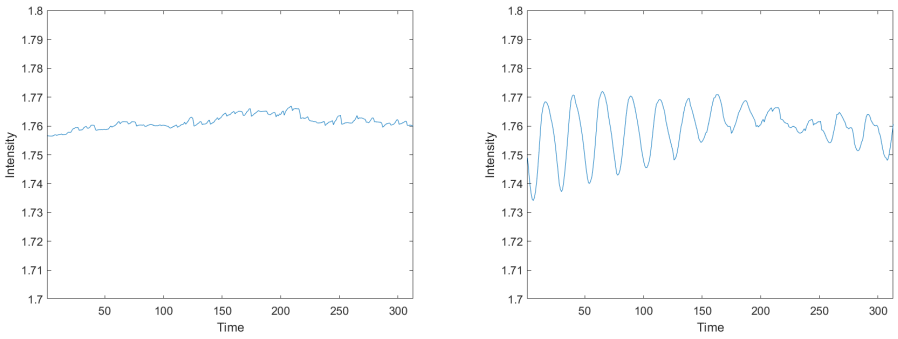
The video where she is not lying is called *jodi_rest* (Figure 6.5a) and the other is called *jodi_lie* (Figure 6.5b).

It was very difficult to select the parameters for either case. Using too few layers in the Gaussian pyramid would make the result extremely noisy and useless for this application, too many layers would make it so the whole image would change colors because too much pooling would be done to the video.



(a) Spatio-temporal representation of the video *me6* unprocessed.

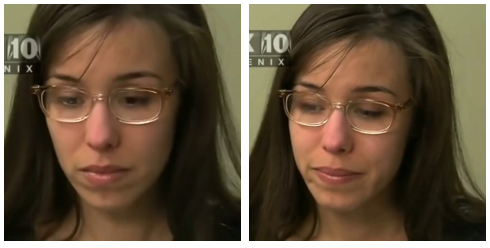
(b) Spatio-temporal representation of the video *me6* color-magnified.



(c) Graph representing the peaks of intensity in the unprocessed *me6* video over time.

(d) Graph representing the peaks of intensity in the color-magnified *me6* video over time.

Figure 6.4



(a) Still frame of the *jodi_rest* video.

(b) Still frame of the *jodi_lie* video.

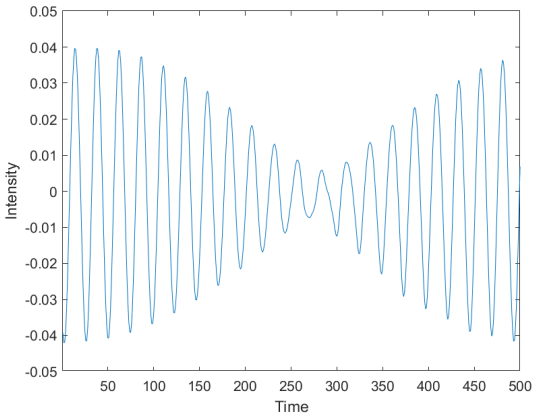


Figure 6.6: Graph showing the difference in intensity frame-by-frame between the processed *jodi_lie* video and the original one.

The pass-band range was also tricky to determine. I did not want to make it too wide to try and reduce the noise being amplified, but I also could not make it too narrow or I would not be able to amplify the signals I wanted.

To amplify the video *jodi_rest*, I used the parameters $\alpha = 120$, $\omega_L = 65/60$, $\omega_H = 70/60$, a Gaussian pyramid of 7 layers and a color attenuation of 0.9. Analyzing Figure 6.7a and 6.7b, one can clearly see that the subject has moved her head during the filming of the video, but the vertical strips can still be easily observed, and the intensity of the overall image fluctuates in time.

One can count around 12 peaks in the graph 6.7d, and the video *jodi_rest* is around 11 seconds long. So her heartbeat rate was around 65 bpm.

So now it would be interesting to compare the result obtained from when Arias was lying with when she was not lying.

To extract her heartbeat when she was lying in the video *jodi_lie*, I used the parameters $\alpha = 50$, $\omega_L = 70/60$, $\omega_H = 80/60$, a Gaussian pyramid of 7 layers and a chroma attenuation of 0.9. Right away, one can notice the stark difference between the spatiotemporal graph in Figure 6.8a and the others that came before them.

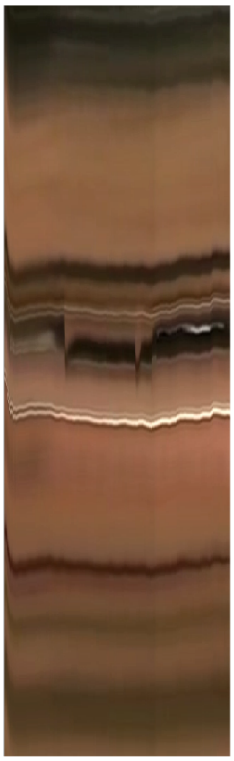
It is easy to see that the subject has moved her head while talking in the way that the lines are jagged and discontinuous. However, when analyzing Figure 6.8b, we can still clearly distinguish the vertical lines that represent the person's heartbeat.

It must be noted, though, that analyzing the intensity graphs, in this case, is not as straightforward. The peaks of intensity in the graph shown in Figure 6.8c impact the graph in Figure 6.8d significantly, and it makes it hard to determine whether it is the change in color on the subject's face that is causing that peak or if she has just moved.

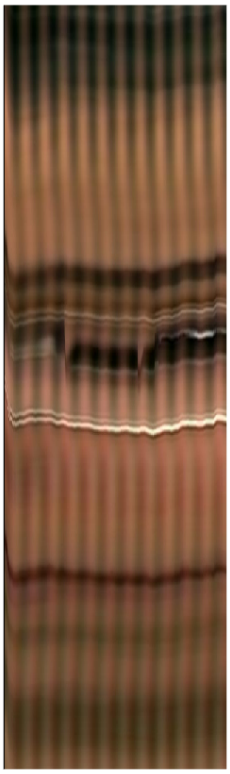
For that reason, I thought it would be interesting to subtract one signal from the other, as shown in Figure 6.6. Because the color magnification process is done by extracting the desired frequency band and adding it to the original video, it makes sense that if we subtract the intensity in the original video from the intensity in the processed video, we will obtain the signal that was added.

Using this logic, we can count 20 peaks in the graph depicted in Figure 6.6. The *jodi_lie* video is 17 seconds long. Therefore, her heartbeat rate was approximately 71 bpm, which is more than when she is not lying.

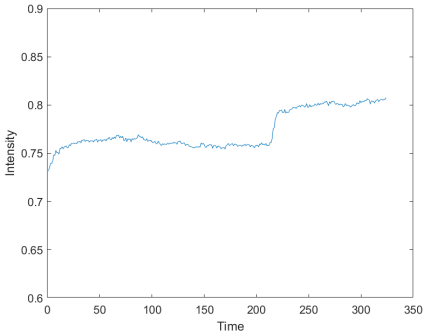
Of course, this does not prove anything, and many other factors could be playing a role in the heartbeat rate increase. This video was not extracted from an official polygraph interview, therefore other factors might have made her more nervous when answering that question. Moreover, the heartbeat rate was very roughly estimated. However, this shows that it is a very interesting idea that could be used in the future.



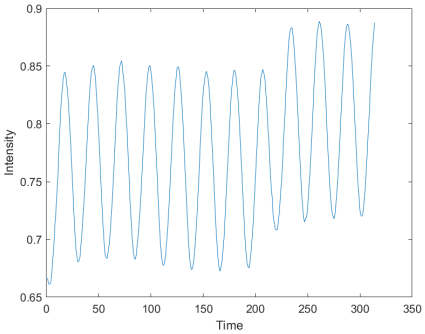
(a) Spatio-temporal representation of the video *jodi_rest* unprocessed.



(b) Spatio-temporal representation of the video *jodi_rest* color-magnified.

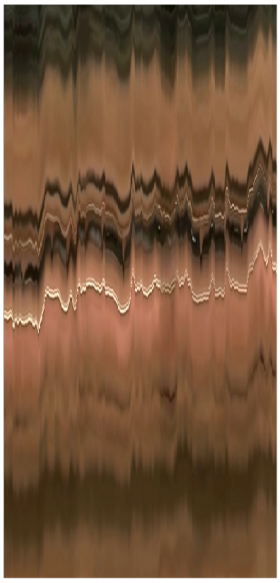


(c) Graph representing the peaks of intensity in the unprocessed *jodi_rest* video over time.

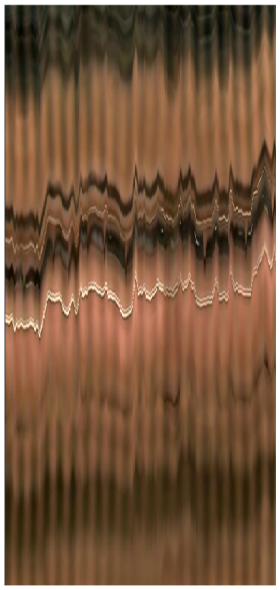


(d) Graph representing the peaks of intensity in the color-magnified *jodi_rest* video over time.

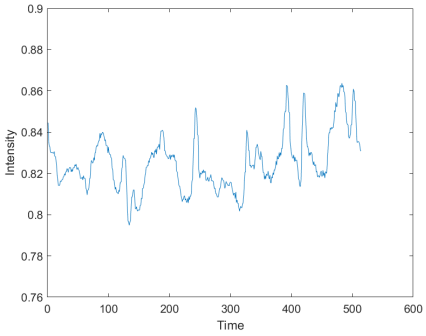
Figure 6.7



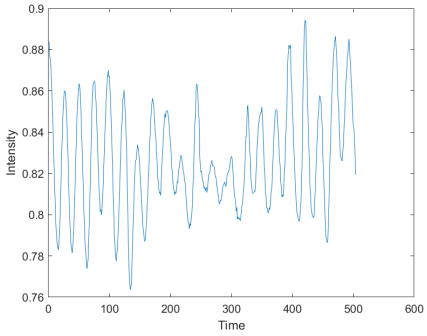
(a) Spatio-temporal representation of the video *jodi_lie* unprocessed.



(b) Spatio-temporal representation of the video *jodi_lie* color-magnified.



(c) Graph representing the peaks of intensity in the unprocessed *jodi_lie* video over time.



(d) Graph representing the peaks of intensity in the color-magnified *jodi_lie* video over time.

Figure 6.8

Chapter 7

Conclusion

In this study, I investigated the idea of using video to extract the heartbeat rate and breathing rates from a suspect in an investigative setting. Through research and analysis, I have gained valuable insight into the subject of signal magnification and its applications.

The results obtained in Section 6 are promising, and while not perfect, they show that it is indeed possible to extract valuable information from videos, even when the subjects are moving slightly and talking.

However, despite the contributions and insights gained from this study, it is important to highlight that there are many limitations to be considered.

First, the pass-band ranges and number of layers have to be manually selected.

Second, the video has to be of sufficiently high quality, and the subject has to be the focus of the footage.

Third, movement and color are amplified simultaneously depending on the pass-band frequency, which is not always desirable.

Fourth, the sample size in which tests were performed was small.

And finally, the magnification does not work if the subject moves too much or if the video is not smooth.

All of these points, coupled with the fact that the subject's heartbeat rate is not the only information that is used when conducting lie detection tests, make the use of these techniques, as-is, impractical in any real-life situations.

However, some other researchers have developed some techniques[13] to detect information in real-time that are promising, and that could show good results if applied in this context.

Despite its limitations, the color and motion magnification techniques described in this thesis are powerful tools that show great potential in many fields of study, including forensic science. I hope that this thesis research will serve as a foundation for further investigations and contribute to evidence-based decision-making in the future.

Chapter 8

Further Work

Despite having learned much from this study, there are still many interesting topics that I would have liked to expand on should I have had more time.

- Implementation of a more robust heartbeat rate extraction technique, as described by Rubinstein[9] and Wu in his 2013 Master Thesis "Eulerian Video Processing and medical applications"[13].
- Implementation of the motion amplification algorithm using Laplacian pyramids instead of Gaussian.
- Testing the algorithm on footage of a polygraph interviewee, with the heartbeat information to verify
- Extracting heartbeat in real-time.
- Making the Python version of the code work, to make the development of a GUI easier in the future.

Bibliography

[1] S. Apter, T. Ebner, T. Freour, Y. Guns, B. Kovacic, N. L. Clef, M. Marques, M. Meseguer, D. Montjean, I. Sfontouris, R. Sturmey, and G. Coticchio. Good practice recommendations for the use of time-lapse technology. *Human Reproduction Open*, 2020.

[2] A. Dhinakaran. Understanding kl divergence. https://towardsdatascience.com/understanding-kl-divergence-f3ddc8dff254. [Online; accessed May 15, 2023].

[3] D. Krapohl and P. Shaw. *Fundamentals of Polygraph Practice*. Elsevier, San Diego, 2015.

[4] J. A. Larson. Lying and its detection. *Journal or Mental Science*, 1933.

[5] J. M. MacDonald. Truth serum. *Journal of Criminal Law And Criminology*, 46, 1955.

[6] D. Mishra. Convolution vs correlation. https://towardsdatascience.com/convolution-vs-correlation-af868b6b4fb5, 2019. [Online; accessed May 15, 2023].

[7] MIT. http://people.csail.mit.edu/mrub/vidmag/. Online; accessed June 20, 2023.

[8] NLM. Cardiac event monitors. https://www.nlm.nih.gov/?_gl=1*gf1x2p*_ga*MjA3NDEzODM3Ni4xNjg2NjcwNDcy*_ga_P1FPTH9PL4*MTY4NjY3MDQ3MS4xLjAuMTY4NjY3MDQ3MS4wLjAuMA..*_ga_7147EPK006*MTY4NjY3MDQ3MS4xLjAuMTY4NjY3MDQ3MS4wLjAuMA..&_ga=2.61769259.68442854.1686670472-2074138376.1686670472. [Online; accessed June 13, 2023].

[9] M. Rubinstein. *Analysis and Visualization of Temporal Variations in Video*. PhD thesis, Massachusetts Institute of Technology, Feb 2014.

[10] J. Synnott, D. Dietzel, and M. Ioannou. A review of the polygraph: history, methodology and current status. *Crime Psychology Review*, 1(1):59–83, 2015.

[11] R. Szeliski, R. Zabih, D. Scharstein, O. Veksler, V. Kolmogorov, A. Agarwala, M. Tappen, and C. Rother. A comparative study of energy minimization methods for markov random fields with smoothness-based priors. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 30(6):1068–1080, 2008.

[12] University of Texas At Dallas. Markov random field optimisation. https://personal.utdallas.edu/~nrr150130/gmbook/mrfs.html. [Online; accessed May 15, 2023].

[13] H.-Y. Wu. Eulerian video processing and medical applications. 03 2013.

[14] E. P. Xing. Variational inference: Loopy belief propagation, 2014.