

ENSICAEN - Projet 2A Informatique
Reconnaissance faciale : anti-fake

VIMONT Ludovic & KOTULSKI Guillaume
Promo 2016



Une grande école pour réussir

Remerciements

Nous tenons à adresser nos remerciements aux personnes qui ont permis que ce projet se déroule dans les meilleures conditions possibles.

Nous remercions M. Jean-Jacques Schwartzmann pour le temps et les conseils qu'il nous a accordé, mais surtout pour nous avoir permis de travailler sur un projet innovant et intéressant.

Nous remercions également Mme. Estelle Cherrier pour son travail dans l'organisation des différents projets.

Enfin, nous remercions M. Ndiaga Faye pour son aide.

Sommaire

1 Introduction

1.1 Contexte

Dans le cadre des cours de l'ENSICAEN, nous devions réaliser un projet de deuxième année. Nous avons choisi de prendre le projet reconnaissance faciale anti-fake proposé par M. Schwartzmann Jean-Jacques.

1.2 Objectifs

L'objectif du projet est la réalisation d'un moyen d'authentification par reconnaissance faciale et par différenciation de l'homme de la photo grâce à l'utilisation d'une capture vidéo. En effet, actuellement ce genre d'applications qui se base seulement sur la reconnaissance faciale sont facilement attaquables. Prenons l'exemple d'un smartphone qui se déverrouille grâce à ce moyen. Il suffit pour un attaquant de disposer de la photo de la victime pour contourner la protection.

Ce projet consiste donc à utiliser une capture vidéo et être capable de différencier un humain d'une photo grâce à cet enregistrement. Afin de réaliser cette différenciation, nous devions baser notre travail sur une [recherche](#) du MIT. Enfin, nous devons intégrer notre résultat dans une application Android de reconnaissance faciale fournie par le [GREYC](#).

1.3 Contraintes

La seule contrainte dont nous disposions était le temps de reconnaissance de la personne, en effet pour l'application Android au-dessus de 4 secondes cela était beaucoup trop long, d'un point de vue utilisateur.

2 Développement

2.1 Outils mis en place

- Site web en ligne (CMS Wordpress): <http://www.ecole.ensicaen.fr/~lvimont/>

The screenshot shows a WordPress blog page titled "Projet 2A". The header features a large image of a green Android robot wearing sunglasses. Below the header is a menu bar with "Accueil" and "Contact". The main content area displays a blog post with the title "[12-03-2015] Implémentation algorithme sur caméra android et webcam". It includes a timestamp ("Posted on 12 mars 2015 by Guillaume Kotulski") and a short text about improving the Android application and its Qt backend. Below the post is a sidebar with a calendar for March 2015, a search bar, and a section titled "Articles récents" listing recent posts.

Figure 2.1: Capture d'écran du site web.

- Gestionnaire de versions (Git)
 - Application Android: <https://github.com/F4r3n/Vamp>
 - Logiciel de traitement d'images: <https://github.com/F4r3n/ImageProject.git>
- Gestionnaire de projets (Trello)

2.2 Technologies utilisées

- Qt : Nous avons utilisé la librairie Qt pour pouvoir créer notre application qui permettait de tester les algorithmes.
- Android : La programmation sous Android s'est faite en utilisant Java et de l'XML.
- OpenCV : Utilisation de la webcam et application utilisant la caméra gérée par OpenCV
- Python : Traitement des données pour création d'un graphique et permet une meilleure vérification des résultats

2.3 Algorithme mis en place

L'œil humain est limité, en effet il est incapable de voir les subtils changements temporels. Alors que justement en effectuant des traitements sur une vidéo. On est capable de révéler ces changements comme par exemple, la circulation du sang. On peut même grâce à ça réussir à connaître le pouls d'une personne. De plus le sang monte jusqu'à la tête puis descend, le fait que notre tête bouge imperceptiblement, ses différents changements peuvent être capturés avec un téléphone portable.

Mais comment obtenir ces changements? Comment savoir où se situe le visage?

Dans un premier temps nous devons détecter si quelque chose ayant un visage humain se tenait devant l'appareil, ceci se fait très bien par l'api de Google sur Android et même par OpenCV. Après détection de la personne nous pouvons obtenir des coordonnées nous disant où se situe le visage.
Maintenant que nous savons où se situe le visage, nous nous demandons comment reconnaître une fréquence?

2.3.1 Choix de l'espace de couleur

Le choix de l'espace de couleur est primordial. C'est en fonction de celui-ci que nous obtiendrons des résultats cohérents ou non.

Un des points importants à prendre en compte fut le changement de luminosité, en se placant dans les espaces HSV, HSL ou YUV et en ne sélectionnant que les canaux Hue et Saturation par exemple pour HSV. Nous pensions obtenir des résultats ne dépendant pas de la luminosité. Sauf que les résultats obtenus par ces espaces de couleur n'étaient pas cohérents.

Le second espace de couleur choisi fut RGB, en effet même si celui-ci dépend fortement de la luminosité, nos résultats étaient meilleurs. Nous pouvions visualiser plus efficacement les changements de variation avec une dérivée.

Après le choix de cet espace de couleur nous devions voir quel canal permettrait d'obtenir des variations les plus fortes. Pour ce faire nous avons visualisé les résultats de notre algorithme sur des personnes ayant des couleurs de peau différente et avec les différents canaux (Red, Green, Blue). Nous remarquions que les résultats étaient meilleurs si nous utilisions seulement le canal Green.

2.3.2 Obtention des données

Nous avons donc fait une moyenne du canal vert (c'est le canal vert qui varie le plus) pour chaque pixel se situant dans un rectangle (en l'occurrence le rectangle délimitant le visage). Et ceci pour chaque frame que l'on capturait, nous obtenions une certaine valeur. Après obtention de ces valeurs nous pouvions les traiter. La variation des valeurs étant très grande, nous avons dans un premier temps fait une moyenne glissante. Puis pour amplifier les variations nous avons fait une dérivée de Taylor. Nos valeurs étant prêtes à être analysées, nous les avons traitées de deux façons différentes. La première, la plus simple fut le Trigger, qui nous permettait d'avoir une fourchette de pulsation.

La deuxième plus longue, une FFT, nécessitait des étapes de calcul supplémentaires telles qu'un fenêtrage

de Hamming ou encore un padding, mais donnaient des résultats plus précis.

Ces deux analyses nous permettent d'avoir le pouls de notre utilisateur (dans le cas idéal, c'est à dire lorsque la caméra et l'utilisateur ne bougeaient presque pas).

Nous avons remarqué que la collecte des données n'accentuait pas assez les variations, nous avons donc créé une deuxième méthode qui en restant basé sur le même principe que la première technique, nous découpons notre image par zone de petits carrés de pixels. Par exemple une zone serait d'une taille de 5×5 . On réalise ainsi une moyenne de chaque zone, puis par la suite une moyenne de ces moyennes. Enfin on rapplique les mêmes fonctions citées précédemment.

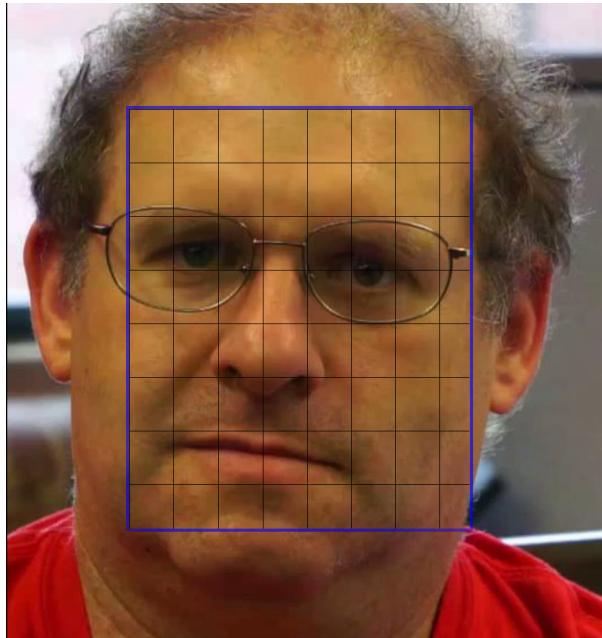


Figure 2.2: Schéma de l'algorithme précédent

2.4 Solutions mis en place

Durant le projet nous avons développé plusieurs outils pour répondre au besoin. Une application Android qui permettrait d'utiliser le moyen d'authentification désiré et ainsi déverrouiller le smartphone d'une personne. Un logiciel C++ utilisant la librairie Qt a également été mis en place, afin de pouvoir tester plus facilement les algorithmes que nous voulions développer. Dans la suite du projet, il a également permis d'utiliser la webcam.

2.4.1 Développement Android

Notre application Android est capable d'utiliser la caméra frontale de n'importe quel smartphone. L'api d'Android propose une classe appelée FaceDetectionListener qui va permettre de capter, les visages présent devant une caméra, grâce à ce système il est possible de dessiner un Rect (c'est à dire un rectangle) autour du visage de la personne. C'est notre classe DisplayedFace qui se charge de ce travail. Une fois la reconnaissance mis en place, nous avons utilisé la fonction `onPreviewFrame`. Cette dernière permet d'enregistrer en temps réel, les données. On lance un compteur quand la première reconnaissance d'un visage a lieu, ce dernier va durer 5 secondes.

Les données ainsi récoltées sont au format de l'espace de couleur **YUV**. Nous devons donc convertir ces données au format RGB qui se révèle plus évocateur au niveau des variations. A la fin du compteur, on réalise une dérivée, puis une amplification des valeurs obtenues. On observe alors les variations qu'on

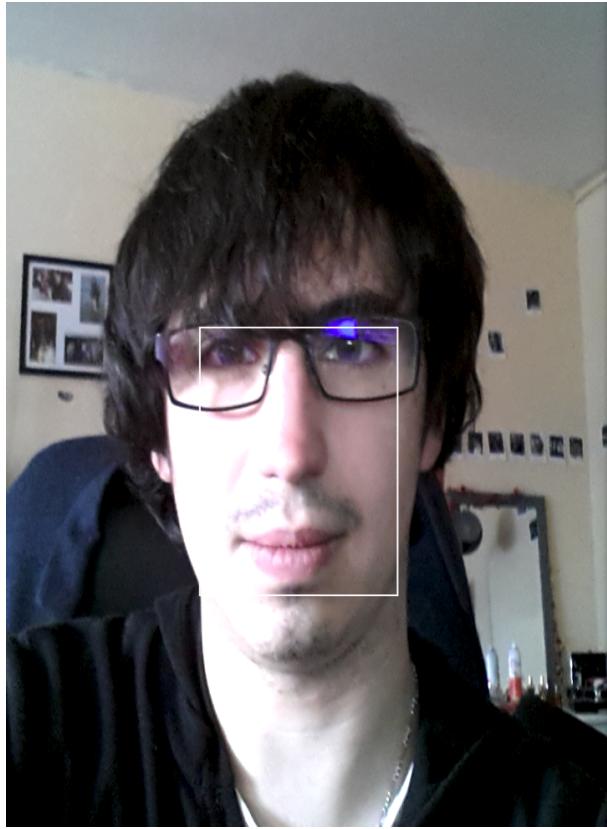


Figure 2.3: Aperçu de l'application Android

obtient et ont conclu alors sur la présence ou non d'un humain.

L'api de Google est très rapide pour détecter un visage et pour nous fournir ces coordonnées en faisant très peu de calcul. Mais celle-ci ne suit pas les petits changements de position, par exemple lorsque l'utilisateur prend une vidéo il bouge légèrement ce qui peut changer le résultat final.

Pour résoudre ce manque nous avons codé une application fonctionnant avec OpenCV, celle-ci suit parfaitement le visage lorsqu'il bouge. Mais le temps de traitement d'une image par OpenCV est beaucoup plus lent que l'api de Google. Ce qui fait que nous tournons à environ 7 FPS en back-caméra et 2 FPS en front-caméra ce qui n'est pas idéal.

L'application sous OpenCV permet de faire une FFT et un système de Trigger, en croisant les résultats donnés par ces deux systèmes nous obtenons un meilleur résultat que s'il n'y avait que le Trigger.

2.4.2 Développement C++

Nous avons créé un logiciel C++ nous permettant de tester nos algorithmes avant de les implémenter sous Android. Ce logiciel a de nombreuses fonctionnalités, notamment le découpage d'une vidéo en images pour permettre une analyse plus poussée. Nous utilisons avconv qui permet de découper une vidéo de n'importe quelle taille en une multitude d'images. (15 images * taille de la vidéo) Après avoir découpé la vidéo en images nous faisons une analyse sur ces images. Cette analyse se découpe en plusieurs étapes. Dans un premier temps nous dessinons un rectangle autour d'un visage (si la fonction du rectangle automatique n'est pas enclenchée) Puis nous choisissons le type d'analyse à effectuer, il y en a deux types:
-L'une fait la moyenne de tous les pixels qui se situe dans le carré
-La seconde découpe le rectangle en plusieurs zones de 5*5 puis amplifie les variations et fait une moyenne des valeurs Après avoir lancé l'analyse, nous affichons les courbes ainsi obtenues par notre application.

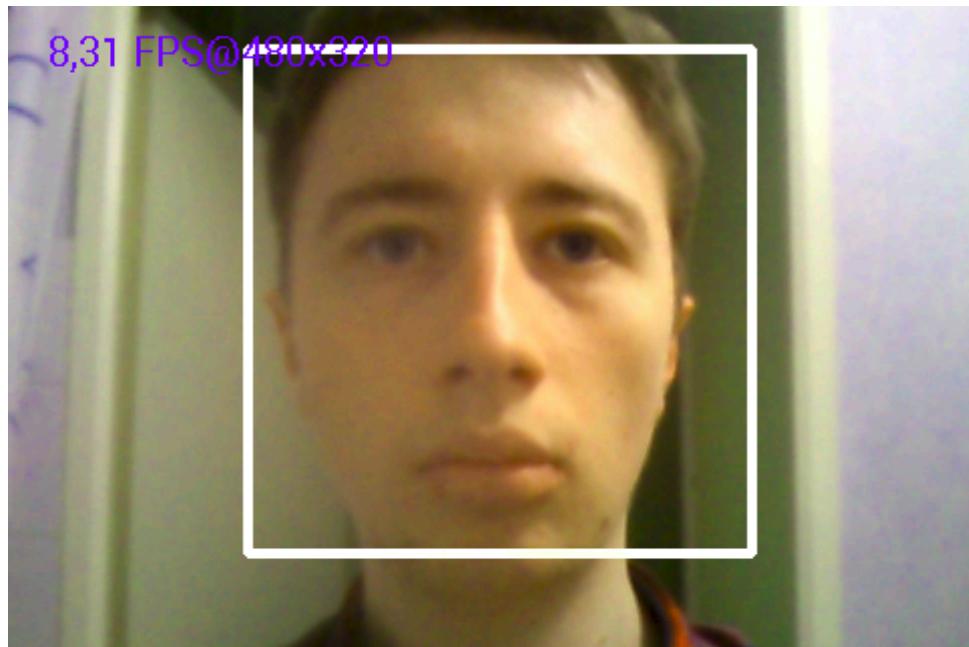


Figure 2.4: Aperçu de l'application OpenCV

Nous pouvons voir la FFT, l'amplification, la dérivée ou une combinaison de plusieurs méthodes. L'amplification utilise la dérivée de Taylor du premier degré. La FFT utilise un filtre passe-bas combiné avec une fenêtre de Hamming et un padding. La FFT permet de connaître le rythme cardiaque d'une personne de manière précise si l'image est assez stable. Nous avons créé un Trigger qui permet aussi de connaître la fréquence cardiaque d'une personne mais de manière moins précise, le Trigger nous donne une fourchette de valeurs.

Si l'espace de couleur que nous avons choisie ne donne pas des résultats corrects nous pouvons en choisir un autre. Nous avons différents espaces de couleur disponibles dont RGB, HSL, HUV et nous pouvons lancer une analyse avec un espace de couleur différent de RGB.

Les analyses avec des vidéos classiques n'étant pas suffisantes pour nos tests, nous avons dû faire une analyse avec des vidéos de la webcam. La webcam est configurée et lancée via la librairie OpenCV et nous enregistrons les images de la webcam avec une fréquence de 15 frames par seconde.

2.5 Problèmes rencontrés

2.5.1 Limitation des sessions à l'ENSICAEN

Lors du projet, nous avons rencontré de nombreux problèmes. L'un des plus embêtants est la limite des sessions à l'Ensicaen. En effet, nos sessions disposent d'une taille limitée, nous pouvons uniquement travailler sous Windows (car c'est seulement sous cet environnement que sont installés les outils Android) hors nous disposons d'uniquelement 150 Mo, or rien qu'avec Firefox, si nous ne nettoyons pas régulièrement l'historique, la session se retrouve complète. Nous avons malheureusement expérimenté le souci et perdu ainsi, une après-midi de travail.

2.5.2 Problèmes rencontrés lors du développement Android

Nous avons eu un problème de rotation, les coordonnées que l'on converser se révélaient mauvaise. Afin d'éviter cela, nous avons décidé de forcer le mode portrait.

La première version de notre application était capable d'enregistrer 15 frames par seconde, or au final on obtenait seulement une trentaine de valeurs, cela était dû à la conversion du format YUV au format RGB que l'on effectuer à chaque fois que nous rentrons dans la fonction `onPreviewFrame`, comme la conversion est coûteuse $O(n^2)$, on perdait des valeurs. Pour optimiser, le nombre de valeurs nous effectuons maintenant la conversion une fois le timer fini. On réalise alors dans la boucle une simple sauvegarde des données brutes. Toutefois cette méthode, nous a causé pas mal de soucis, notamment des out of memory, c'est-à-dire, que nous réalisions une allocation trop grande par rapport à la mémoire disponible ...

Même si l'algorithme fonctionnait avec des vidéos, ceci ne voulait pas dire qu'il fonctionnerait sous Android. En effet lorsque avec la caméra on se prend en vidéo on bouge légèrement, ce qui pose de nombreux problèmes de stabilité. Les variations que nous obtenons alors ne seraient plus celle de la pulsation mais celles du mouvement régulier de la main.

3 Résultats

3.1 Cadre idéal

3.1.1 Définition du cadre idéal

Notre algorithme fonctionne parfaitement dans le cadre idéal, une fois sortie de ce cadre les résultats sont moins précis et peuvent être incohérents.

Ayant choisi un espace de couleur RGB, le moindre changement de luminosité nous donne des résultats incohérents.

Il faut que l'appareil tel que la webcam ou le smartphone ne bouge pas lors de la collecte des données. Si celui-ci bouge de trop nous allons détecter des variations et notre application la percevra comme des fréquences cardiaques.

L'appareil doit enregistrer les données avec une résolution supérieure à 480*320.

La personne filmée ne doit être pas trop éloignée de l'objectif pour avoir des résultats optimaux.

3.1.2 Résultats dans le cadre idéal

Nous avons effectué différents tests durant le projet. En utilisant les vidéos fournies sur le site du MIT. Notre logiciel arrive à obtenir des résultats très correct. En effet, par exemple comme on peut le voir sur cette capture d'écran. Après notre magnification réalisée, on retrouve une fréquence cardiaque d'environ 53 battements par minute (BPM), l'article du MIT lui trouve une fréquence de 54 BPM.

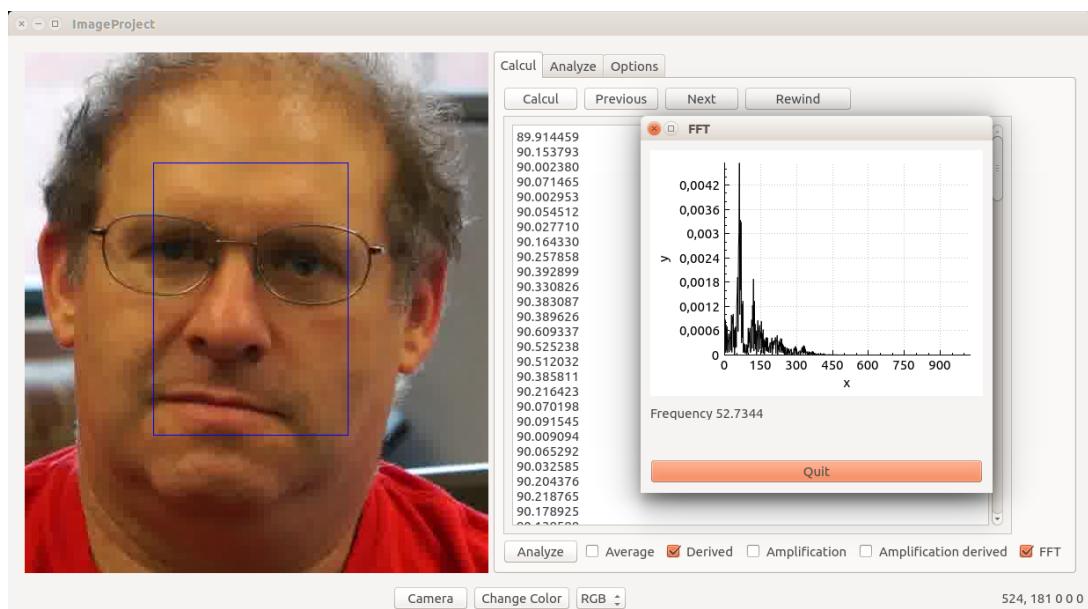


Figure 3.1: Vidéo source du MIT analysé avec notre logiciel.

On pouvait se poser la question, des personnes de couleur, est-ce qu'une couleur de peau différente aurait pu avoir des conséquences? Nous avons donc tester avec une autre vidéo fournie par le MIT avec une personne de peau noir, on peut voir sur la capture suivante que cela n'a en rien influencer sur notre algorithme. On retrouve donc une fréquence comprise entre 47 et 59 BPM.

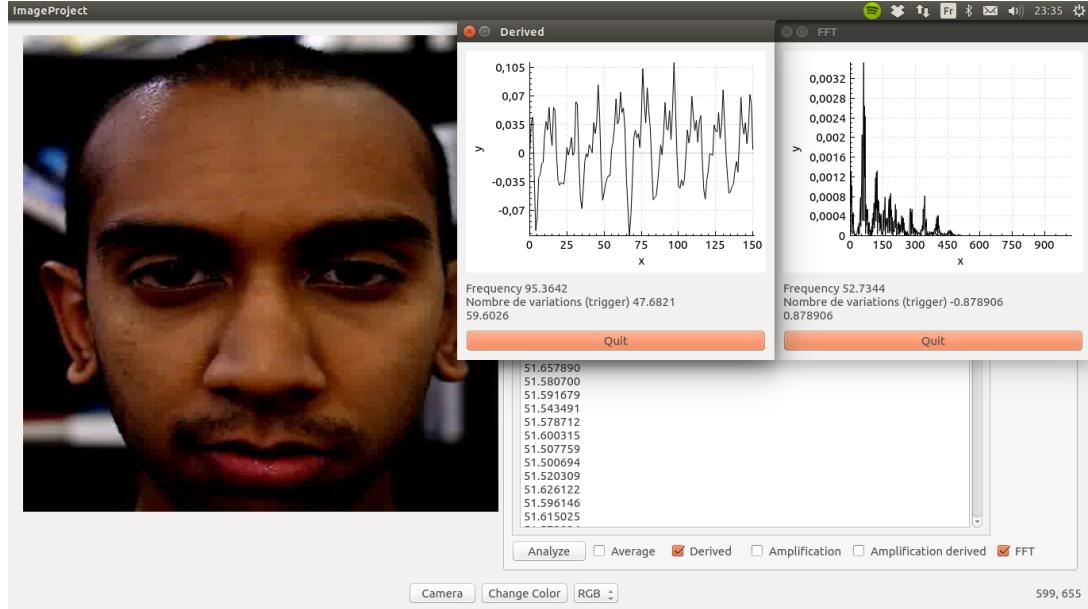


Figure 3.2: Autres test avec une personne de couleur de peau différente.

3.2 Webcam

Avec une webcam, nos résultats sont moins précis, mais sont assez correct pour être utiliser.

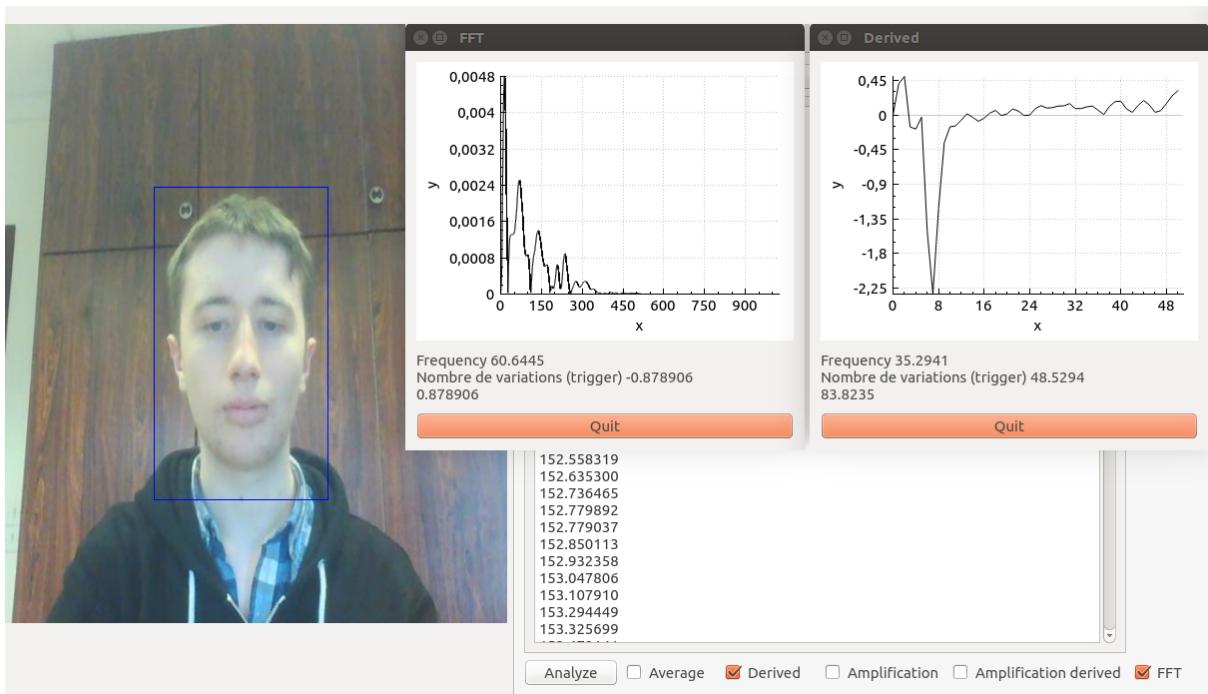


Figure 3.3: Vidéo pris par la webcam et analysé avec notre logiciel.

3.3 Mobile

3.3.1 Android

Notre plus gros problème a était lors de nos tests sur Android, la stabilité de la vidéo et le nombre de frames capturés. Actuellement notre application est capable de capturer des frames et appliquer notre algorithme mais nos résultats restent erronées.

3.3.2 OpenCV

L'utilisation d'une FFT et d'un Trigger permet de mieux différencier l'homme de la photo. Même si les résultats sont meilleurs il reste de nombreuses erreurs et le temps de calcul est beaucoup plus long.

3.3.3 Ressources utilisées

La mémoire accordé à notre application par le système d'exploitation de l'ordinateur est fixe et ne peut être augmenté.

Donc nous devons faire nos enregistrements avec une mémoire limitée, c'est à dire que le nombre d'image capturée était limité. Une plus grande résolution de la caméra implique donc moins d'images à enregistrer. De plus le nombre d'opération a effectué étant important, nous avions un temps de calcul assez important sous OpenCV. Pour 10 secondes d'enregistrement avec une résolution de 480*320 à raison de 6 images par seconde, soit 60 images nous obtenions un temps de calcul de 20 secondes. Ce qui est beaucoup trop important pour une application qui se veut rapide.

Sous android natif le problème fut que le nombre d'images capturé n'était pas suffisant pour permettre une analyse correcte, un enregistrement d'un nombre trop important d'image (supérieur à 30) engendrait un crash de notre application.

3.4 Bilan

Notre différenciation entre un humain et une photo fonctionne correctement dès lors où la stabilité est assurée. En effet, prenons Android, nous arrivons à capturer plus de frames qu'en utilisant OpenCV, toutefois le tracking d'OpenCV est plus efficace. De ce fait, malgré une meilleure capture sous Android comme la stabilité est plus mauvaise, nos résultats sont moins bons. Mais à contrario, lorsqu'on utilise l'application réalisée avec OpenCV, le nombre d'images capturé est seulement de 20 pour 4 secondes, avec si peu d'images à traiter notre magnification n'est pas correcte. C'est seulement au bout de 10 secondes lorsqu'on capture entre 50 à 70 frames qu'on a des résultats cohérents.

4 Conclusion

4.1 Apports techniques

Ce projet nous a permis de mettre en œuvre l'enseignement que nous a donné l'ENSICAEN, l'utilisation de Github, les cours de Java/C++ et de conception d'interface graphique. Il nous a également permis d'apprendre le développement mobile et l'utilisation de librairie comme OpenCV.

4.1.1 Android

Android est un système d'exploitation sur lequel nous n'avions jamais programmé, l'appréhender par nous même fut difficile. Le placement des widgets sur l'écran et même le système d'activité, nous étaient inconnus et nous avions passé de nombreuses heures à le comprendre complètement. De plus notre projet utilise la caméra, l'un des périphériques, le plus difficile à mettre en place.

L'un des plus gros avantages d'android, c'est la documentation qui est bien écrite, ce qui est très appréciable. Grâce à ce projet, nous avons pu avoir un aperçu du système, de ces forces et de ces faiblesses.

4.1.2 Qt

Nous avons déjà eu quelques expériences avec cette bibliothèque, mais nous avons réellement pu appliquer nos connaissances avec ce projet. Notre logiciel de test fut d'une grande aide pour l'intégration de nos algorithmes sous Android.

Ce logiciel comprenant de nombreux modules, nous avons dû appliquer des méthodes vue en génie logiciel pour que celui-ci fonctionne correctement.

4.1.3 OpenCV

Nous avons dû utiliser la bibliothèque OpenCV, ce qui était une nouveauté. Celle ci nous a permis d'avoir un meilleur contrôle des images prises par la webcam.

4.2 Apports scientifiques

Nous avons également pu mettre à profit les cours que nous avons eus sur les différents espaces de couleurs, surtout utile comme nous l'avons vu lors du développement Android. Le plus gros défi fut la mise en place de techniques du traitement du signal, telle que la FFT ou encore la réalisation de filtre basse fréquence.

4.3 Bilan

Ce projet a donc été un vrai aboutissement de notre formation. Il nous a forcé à appréhender de nouvelles technologies et librairies mais également à réutiliser tout ce que nous connaissons. Nous avons pu voir les nombreux avantages d'une documentation bien construite et bien imaginée, seulement permettant de bien sur s'approprier rapidement la technologie étudiée.

5 Bibliographie

http://en.wikipedia.org/wiki/Fast_Fourier_transform

http://en.wikipedia.org/wiki/Window_function

<https://developer.android.com/guide/index.html>

A Annexes

Voici l'article du MIT sur la magnification d'une vidéo, nous avons choisi de prendre juste le début du billet car il correspond à leur développement sur la couleur, la partie sur laquelle nous avons travaillé.

Eulerian Video Magnification for Revealing Subtle Changes in the World

Hao-Yu Wu¹ Michael Rubinstein¹ Eugene Shih² John Guttag¹ Frédo Durand¹ William Freeman¹
¹MIT CSAIL ²Quanta Research Cambridge, Inc.

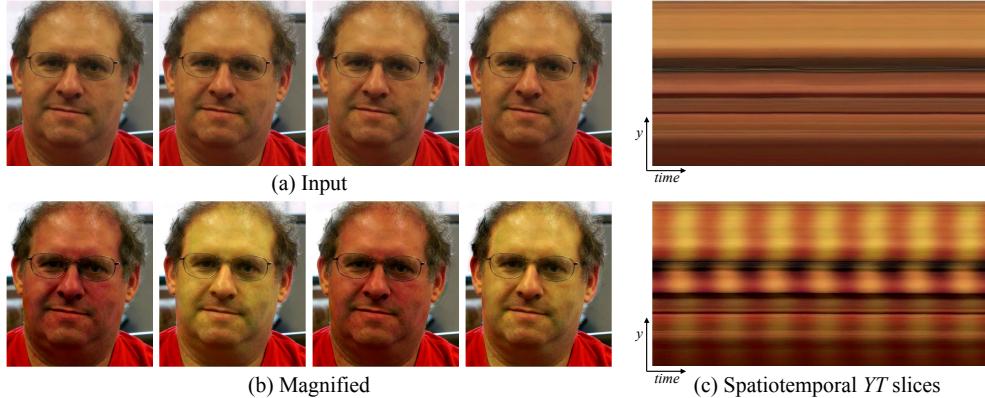


Figure 1: An example of using our Eulerian Video Magnification framework for visualizing the human pulse. (a) Four frames from the original video sequence (face). (b) The same four frames with the subject’s pulse signal amplified. (c) A vertical scan line from the input (top) and output (bottom) videos plotted over time shows how our method amplifies the periodic color variation. In the input sequence the signal is imperceptible, but in the magnified sequence the variation is clear. The complete sequence is available in the supplemental video.

Abstract

Our goal is to reveal temporal variations in videos that are difficult or impossible to see with the naked eye and display them in an indicative manner. Our method, which we call Eulerian Video Magnification, takes a standard video sequence as input, and applies spatial decomposition, followed by temporal filtering to the frames. The resulting signal is then amplified to reveal hidden information. Using our method, we are able to visualize the flow of blood as it fills the face and also to amplify and reveal small motions. Our technique can run in real time to show phenomena occurring at temporal frequencies selected by the user.

CR Categories: I.4.7 [Image Processing and Computer Vision]: Scene Analysis—Time-varying Imagery;

Keywords: video-based rendering, spatio-temporal analysis, Eulerian motion, motion magnification

Links: [DL](#) [PDF](#) [WEB](#)

1 Introduction

The human visual system has limited spatio-temporal sensitivity, but many signals that fall below this capacity can be informative.

For example, human skin color varies slightly with blood circulation. This variation, while invisible to the naked eye, can be exploited to extract pulse rate [Verkruyse et al. 2008; Poh et al. 2010; Philips 2011]. Similarly, motion with low spatial amplitude, while hard or impossible for humans to see, can be magnified to reveal interesting mechanical behavior [Liu et al. 2005]. The success of these tools motivates the development of new techniques to reveal invisible signals in videos. In this paper, we show that a combination of spatial and temporal processing of videos can amplify subtle variations that reveal important aspects of the world around us.

Our basic approach is to consider the time series of color values at any spatial location (pixel) and amplify variation in a given temporal frequency band of interest. For example, in Figure 1 we automatically select, and then amplify, a band of temporal frequencies that includes plausible human heart rates. The amplification reveals the variation of redness as blood flows through the face. For this application, temporal filtering needs to be applied to lower spatial frequencies (spatial pooling) to allow such a subtle input signal to rise above the camera sensor and quantization noise.

Our temporal filtering approach not only amplifies color variation, but can also reveal low-amplitude motion. For example, in the supplemental video, we show that we can enhance the subtle motions around the chest of a breathing baby. We provide a mathematical analysis that explains how temporal filtering interplays with spatial motion in videos. Our analysis relies on a linear approximation related to the brightness constancy assumption used in optical flow formulations. We also derive the conditions under which this approximation holds. This leads to a multiscale approach to magnify motion without feature tracking or motion estimation.

Previous attempts have been made to unveil imperceptible motions in videos. [Liu et al. 2005] analyze and amplify subtle motions and visualize deformations that would otherwise be invisible. [Wang et al. 2006] propose using the Cartoon Animation Filter to create perceptually appealing motion exaggeration. These approaches follow a *Lagrangian* perspective, in reference to fluid dynamics where the trajectory of particles is tracked over time. As such, they rely

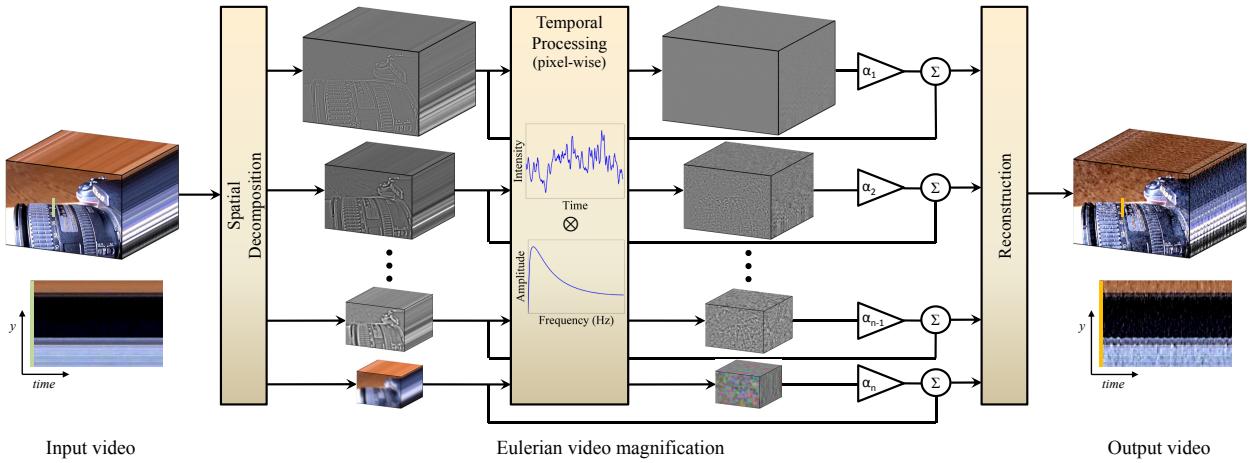


Figure 2: Overview of the Eulerian video magnification framework. The system first decomposes the input video sequence into different spatial frequency bands, and applies the same temporal filter to all bands. The filtered spatial bands are then amplified by a given factor α , added back to the original signal, and collapsed to generate the output video. The choice of temporal filter and amplification factors can be tuned to support different applications. For example, we use the system to reveal unseen motions of a Digital SLR camera, caused by the flipping mirror during a photo burst (camera; full sequences are available in the supplemental video).

on accurate motion estimation, which is computationally expensive and difficult to make artifact-free, especially at regions of occlusion boundaries and complicated motions. Moreover, Liu et al. [2005] have shown that additional techniques, including motion segmentation and image in-painting, are required to produce good quality synthesis. This increases the complexity of the algorithm further.

In contrast, we are inspired by the *Eulerian* perspective, where properties of a voxel of fluid, such as pressure and velocity, evolve over time. In our case, we study and amplify the variation of pixel values over time, in a spatially-multiscale manner. In our Eulerian approach to motion magnification, we do not explicitly estimate motion, but rather exaggerate motion by amplifying temporal color changes at fixed positions. We rely on the same differential approximations that form the basis of optical flow algorithms [Lucas and Kanade 1981; Horn and Schunck 1981].

Temporal processing has been used previously to extract invisible signals [Poh et al. 2010] and to smooth motions [Fuchs et al. 2010]. For example, Poh et al. [2010] extract a heart rate from a video of a face based on the temporal variation of the skin color, which is normally invisible to the human eye. They focus on extracting a single number, whereas we use localized spatial pooling and bandpass filtering to extract and reveal visually the signal corresponding to the pulse. This primal domain analysis allows us to amplify and visualize the pulse signal at each location on the face. This has important potential monitoring and diagnostic applications to medicine, where, for example, the asymmetry in facial blood flow can be a symptom of arterial problems.

Fuchs et al. [2010] use per-pixel temporal filters to dampen temporal aliasing of motion in videos. They also discuss the high-pass filtering of motion, but mostly for non-photorealistic effects and for large motions (Figure 11 in their paper). In contrast, our method strives to make imperceptible motions visible using a multiscale approach. We analyze our method theoretically and show that it applies only for small motions.

In this paper, we make several contributions. First, we demonstrate that nearly invisible changes in a dynamic environment can be revealed through *Eulerian* spatio-temporal processing of standard monocular video sequences. Moreover, for a range of amplification values that is suitable for various applications, explicit motion estimation is not required to amplify motion in natural videos. Our

approach is robust and runs in real time. Second, we provide an analysis of the link between temporal filtering and spatial motion and show that our method is best suited to small displacements and lower spatial frequencies. Third, we present a single framework that can be used to amplify both spatial motion and purely temporal changes, e.g., the heart pulse, and can be adjusted to amplify particular temporal frequencies—a feature which is not supported by Lagrangian methods. Finally, we analytically and empirically compare Eulerian and Lagrangian motion magnification approaches under different noisy conditions. To demonstrate our approach, we present several examples where our method makes subtle variations in a scene visible.

2 Space-time video processing

Our approach combines spatial and temporal processing to emphasize subtle temporal changes in a video. The process is illustrated in Figure 2. We first decompose the video sequence into different spatial frequency bands. These bands might be magnified differently because (a) they might exhibit different signal-to-noise ratios or (b) they might contain spatial frequencies for which the linear approximation used in our motion magnification does not hold (Sect. 3). In the latter case, we reduce the amplification for these bands to suppress artifacts. When the goal of spatial processing is simply to increase temporal signal-to-noise ratio by pooling multiple pixels, we spatially low-pass filter the frames of the video and downsample them for computational efficiency. In the general case, however, we compute a full Laplacian pyramid [Burt and Adelson 1983].

We then perform temporal processing on each spatial band. We consider the time series corresponding to the value of a pixel in a frequency band and apply a bandpass filter to extract the frequency bands of interest. For example, we might select frequencies within 0.4–4Hz, corresponding to 24–240 beats per minute, if we wish to magnify a pulse. If we are able to extract the pulse rate, we can use a narrow band around that value. The temporal processing is uniform for all spatial levels, and for all pixels within each level. We then multiply the extracted bandpassed signal by a magnification factor α . This factor can be specified by the user, and may be attenuated automatically according to guidelines in Sect. 3.2. Possible temporal filters are discussed in Sect. 4. Next, we add the magnified signal to the original and collapse the spatial pyramid to obtain

the final output. Since natural videos are spatially and temporally smooth, and since our filtering is performed uniformly over the pixels, our method implicitly maintains spatiotemporal coherency of the results.

3 Eulerian motion magnification

Our processing can amplify small motion even though we do not track motion as in Lagrangian methods [Liu et al. 2005; Wang et al. 2006]. In this section, we show how temporal processing produces motion magnification using an analysis that relies on the first-order Taylor series expansions common in optical flow analyses [Lucas and Kanade 1981; Horn and Schunck 1981].

3.1 First-order motion

To explain the relationship between temporal processing and motion magnification, we consider the simple case of a 1D signal undergoing translational motion. This analysis generalizes directly to locally-translational motion in 2D.

Let $I(x, t)$ denote the image intensity at position x and time t . Since the image undergoes translational motion, we can express the observed intensities with respect to a displacement function $\delta(t)$, such that $I(x, t) = f(x + \delta(t))$ and $I(x, 0) = f(x)$. The goal of motion magnification is to synthesize the signal

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

for some amplification factor α .

Assuming the image can be approximated by a first-order Taylor series expansion, we write the image at time t , $f(x + \delta(t))$ in a first-order Taylor expansion about x , as

$$I(x, t) \approx f(x) + \delta(t) \frac{\partial f(x)}{\partial x}. \quad (2)$$

Let $B(x, t)$ be the result of applying a broadband temporal bandpass filter to $I(x, t)$ at every position x (picking out everything except $f(x)$ in Eq. 2). For now, let us assume the motion signal, $\delta(t)$, is within the passband of the temporal bandpass filter (we will relax that assumption later). Then we have

$$B(x, t) = \delta(t) \frac{\partial f(x)}{\partial x}. \quad (3)$$

In our process, we then amplify that bandpass signal by α and add it back to $I(x, t)$, resulting in the processed signal

$$\tilde{I}(x, t) = I(x, t) + \alpha B(x, t). \quad (4)$$

Combining Eqs. 2, 3, and 4, we have

$$\tilde{I}(x, t) \approx f(x) + (1 + \alpha)\delta(t) \frac{\partial f(x)}{\partial x}. \quad (5)$$

Assuming the first-order Taylor expansion holds for the amplified larger perturbation, $(1 + \alpha)\delta(t)$, we can relate the amplification of the temporally bandpassed signal to motion magnification. The processed output is simply

$$\tilde{I}(x, t) \approx f(x + (1 + \alpha)\delta(t)). \quad (6)$$

This shows that the processing magnifies motions—the spatial displacement $\delta(t)$ of the local image $f(x)$ at time t , has been amplified to a magnitude of $(1 + \alpha)$.

This process is illustrated for a single sinusoid in Figure 3. For a low frequency cosine wave and a relatively small displacement,

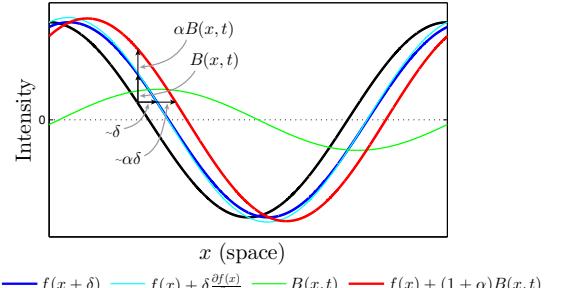


Figure 3: Temporal filtering can approximate spatial translation. This effect is demonstrated here on a 1D signal, but equally applies to 2D. The input signal is shown at two time instants: $\tilde{I}(x, t) = f(x)$ at time t and $\tilde{I}(x, t + 1) = f(x + \delta)$ at time $t + 1$. The first-order Taylor series expansion of $\tilde{I}(x, t + 1)$ about x approximates well the translated signal. The temporal bandpass is amplified and added to the original signal to generate a larger translation. In this example $\alpha = 1$, magnifying the motion by 100%, and the temporal filter is a finite difference filter, subtracting the two curves.

$\delta(t)$, the first-order Taylor series expansion serves as a good approximation for the translated signal at time $t + 1$. When boosting the temporal signal by α and adding it back to $I(x, t)$, we approximate that wave translated by $(1 + \alpha)\delta$.

For completeness, let us return to the more general case where $\delta(t)$ is not entirely within the passband of the temporal filter. In this case, let $\delta_k(t)$, indexed by k , represent the different temporal spectral components of $\delta(t)$. Each $\delta_k(t)$ will be attenuated by the temporal filtering by a factor γ_k . This results in a bandpassed signal,

$$B(x, t) = \sum_k \gamma_k \delta_k(t) \frac{\partial f(x)}{\partial x} \quad (7)$$

(compare with Eq. 3). Because of the multiplication in Eq. 4, this temporal frequency dependent attenuation can equivalently be interpreted as a frequency-dependent motion magnification factor, $\alpha_k = \gamma_k \alpha$, resulting in a motion magnified output,

$$\tilde{I}(x, t) \approx f(x + \sum_k (1 + \alpha_k) \delta_k(t)) \quad (8)$$

The result is as would be expected for a linear analysis: the modulation of the spectral components of the motion signal becomes the modulation factor in the motion amplification factor, α_k , for each temporal subband, δ_k , of the motion signal.

3.2 Bounds

In practice, the assumptions in Sect. 3.1 hold for smooth images and small motions. For quickly changing image functions (i.e., high spatial frequencies), $f(x)$, the first-order Taylor series approximations becomes inaccurate for large values of the perturbation, $1 + \alpha\delta(t)$, which increases both with larger magnification α and motion $\delta(t)$. Figures 4 and 5 demonstrate the effect of higher frequencies, larger amplification factors and larger motions on the motion-amplified signal of a sinusoid.

As a function of spatial frequency, ω , we can derive a guide for how large the motion amplification factor, α , can be, given the observed motion $\delta(t)$. For the processed signal, $\tilde{I}(x, t)$ to be approximately equal to the true magnified motion, $\hat{I}(x, t)$, we seek the conditions under which

$$\begin{aligned} \tilde{I}(x, t) &\approx \hat{I}(x, t) \\ \Rightarrow f(x) + (1 + \alpha)\delta(t) \frac{\partial f(x)}{\partial x} &\approx f(x + (1 + \alpha)\delta(t)) \end{aligned} \quad (9)$$