Ecole Polytechnique de l'Université François Rabelais de Tours
Département Informatique
64 avenue Jean Portalis
37200 Tours, France
Tél. +33 (0)2 47 36 14 14
www.polytech.univ-tours.fr

Projet Recherche & Développement
2015-2016

Estimation de la qualité perçue d'une vidéo

Tuteurs académiques
Donatello Conte

Étudiants
Anthony Demarcy (DI5)

26 septembre 2016

Liste des intervenants

Nom	Mail	Qualité
Anthony Demarcy	anthony.demarcy@etu.univ-tours.fr	Étudiant DI5
Donatello Conte	donatello.conte@univ-tours.fr	Tuteur académique, Département infomatique

Avertissement

Ce document a été rédigé par Anthony Demarcy susnommé les auteurs.

L'école polytechnique de l'université François Rabelais de Tours est représentée par Donatello Conte susnommé les tuteurs académiques.

Par l'utilisation de ce modèle de document, l'ensemble des intervenants du projet acceptent les conditions définies ci-après.

Les auteurs reconnaissent assumer l'entière responsabilité du contenu du document ainsi que toutes suites judiciaires qui pourraient en découler du fait du non respects des lois ou des droits d'auteur.

Les auteurs attestent que les propos du document sont sincères et assument l'entière responsabilité de la véracité des propos.

Les auteurs attestent ne pas s'approprier le travail d'autrui et que le document ne contient aucun plagiat.

Les auteurs attestent que le document ne contient aucun propos diffamatoire ou condamnable devant la loi.

Les auteurs reconnaissent qu'ils ne peuvent diffuser ce document en partie ou en intégralité sous quelque forme que ce soit sans l'accord préalable des tuteurs académiques.

Les auteurs autorisent l'école polytechnique de l'université François Rabelais de Tours à diffuser tout ou partie de ce document, sous quelque forme que ce soit, y compris après transformation en citant la source. Cette diffusion devra se faire gracieusement et être accompagnée du présent avertissement.

Pour citer ce document :
Anthony Demarcy, *Estimation de la qualité perçue d'une vidéo:* , Projet Recherche & Développement, Ecole Polytechnique de l'Université François Rabelais de Tours, Tours, France, 2015-2016.

```
@mastersthesis{
    author={Demarcy, Anthony},
    title={Estimation de la qualité perçue d'une vidéo: },
    type={Projet Recherche & Développement},
    school={Ecole Polytechnique de l'Université François Rabelais de Tours},
    address={Tours, France},
    year={2015-2016}
}
```

Document mis en forme sous LaTeX
grâce à la classe de document `polytech.cls` V1.0 (2015/11/18)

Table des matières

Table des figures

7 Campagne de tests

Liste des tableaux

Introduction

L'estimation de la qualité d'une vidéo est une problématique épineuse. Le principe peut sembler simple, mais la notion même de "qualité" est floue : parle-t-on de ***qualité objective***, basée sur des critères purement mathématiques, ou bien d'une ***qualité subjective*** qui tenterait de retranscrire la qualité de la vidéo telle qu'elle est perçue par un oeil humain ?

Sachant que les vidéos sont destinées dans la très grande majorité des cas à être visionnées par des utilisateurs humains, la qualité subjective, autrement appelée **qualité perçue**, est intéressante à étudier.

L'objectif de ce projet est de **réaliser un outil** qui, pour une vidéo donnée en entrée, permettrait d'en estimer la qualité perçue dans des délais raisonnables. Il ne fait pas suite à un autre projet sur la même thématique, ce qui implique qu'aucun existant n'a dû être pris en considération.

Ce projet a été proposé et encadré par **Donatello Conte** de **l'équipe RFAI** (Reconnaissance de Formes et Analyse d'Images) **du laboratoire d'informatique de Polytech Tours**. Cette équipe est composée d'une vingtaine de membres permanents (maîtres de conférences et professeurs) et d'une dizaine de membres non-permanents (doctorants et postdoc), s'investissant tous dans la recherche l'analyse d'images, l'auto-apprentissage et d'autres domaines connexes.

Figure 1 – *Logos du laboratoire d'informatique et de l'équipe RFAI*

Ce rapport expose dans un premier temps une synthèses des recherches réalisées autour de la problématique d'estimation de la qualité perçue d'une vidéo, pour ensuite proposer une conception appropriée pour l'outil qui doit être réalisé. Une dernière partie reviendra sur la méthodologie de gestion de projet employée afin de mener ce dernier à bien.

Première partie

Recherche

1 Notions générales

1 Définition de la qualité perçue

La **qualité perçue** (ou *qualité perceptuelle*) peut être définie comme "***la qualité d'une vidéo telle qu'elle est (en moyenne) perçue par un oeil humain***".

Il s'agit d'une notion purement **subjective** qui fait appel à un jugement personnel sur un élément concret et persistent. Par conséquent, plusieurs personnes peuvent attribuer une qualité différente à une même vidéo, selon leur acuité visuelle, leur habitude de ce genre de vidéos ou bien d'autres facteurs, quelquefois connexes au domaine de la psychologie (e.g. le degré d'exigence). Il est même possible d'envisager qu'une même personne puisse donner des scores de qualité différents à une même vidéo si les deux tests sont réalisés à des instants éloignés dans le temps.

Dans l'idéal, la qualité perçue s'applique à la *forme* de la vidéo étudiée, et non au *fond*. Ainsi, ce n'est pas la qualité du contenu de la vidéo que l'estimateur juge, mais la **qualité des images** qu'elle comporte.

Un algorithme d'estimation de la qualité perçue doit tenter de s'approcher au mieux de la "vérité terrain", c'est-à-dire du score de qualité qu'auraient attribué des visionneurs humains en chair et en os à la même vidéo. Avant d'étudier des pistes pour réaliser des algorithmes parvenant à estimer la qualité perçue, il serait donc intéressant de se pencher sur les protocoles existants pour évaluer la qualité perçue qui permettent d'obtenir cette fameuse "vérité terrain".

1.1 Protocole d'évaluation de la qualité perçue

Le protocole de test subjectif défini ci-dessous est celui basé sur les recommandations ITU-R BT.500-11 et ITU-T P.910[2].

Pour réaliser un tel test, il est nécessaire de réunir un peu plus d'une vingtaine de personnes (qu'on appelle "**sujets**") qui regardent un ensemble de vidéos et donnent une note à la qualité de chacune d'elles. La moyenne des notes données par chacun des sujets à une même vidéo est effectuée et produit un ***Mean Opinion Score*** (score d'opinion moyen, abrégé MOS), qui représente un échantillon de référence en terme de qualité subjective pour cette vidéo.

Il ne faut pas oublier que ces tests demeurent subjectifs et peuvent varier selon les sujets et les conditions de visionnage. C'est d'ailleurs pour atténuer ces variations que l'on demande à un nombre de sujets relativement grand de réaliser l'expérience en même temps, dans des environnements prévus pour de telles expériences et avec des conditions de visionnage optimales, pour obtenir des résultats de fiabilité

maximale. Même en prenant ces précautions, le score de qualité obtenu n'est pas à prendre comme un nombre exact, mais davantage comme un intervalle (voire même une distribution statistique).

Il est possible d'avoir recours à différentes procédures de visionnage, selon des ressources dont on dispose et du type de qualité que l'on souhaite évaluer. Par exemple, une procédure simple pourrait structurer le visionnage de la vidéo en deux parties : un premier visionnage en "qualité parfaite", puis un second avec des détériorations de qualité.

Parmi les procédures couramment utilisées, on peut citer *Double Stimulus Continuous Quality Scale* (**DSCQS**), *Stimulus Impairment Scale* (**DSIS**), *Single Stimulus Continuous Quality Evaluation* (**SSCQE**) ou encore *Absolute Category Rating* (**ACR**).

Ces tests sont très coûteux, demandent du personnel et du temps à réaliser. Il n'est donc pas réaliste de considérer ces tests subjectifs dans de nombreux cas : par exemple, dans le cadre d'un système temps réel, les résultats doivent être obtenus dans des fractions de secondes, là où un test subjectif met plusieurs heures à être réalisé. Pour des raisons économiques et de limitations de ressources, la nécessité de réaliser des systèmes d'estimation automatisés (comme celui qui va être conçu pour ce projet) devient évidente.

1.2 Composante spatiale et temporelle

Ignorer le *fond* de la vidéo (la nature de son contenu) est utopique dans l'étude de la qualité perceptuelle. En effet, l'œil humain ne perçoit précisément les détails que dans une zone assez réduite où la densité des cellules rétiniennes est très élevée (la **fovéa**), et celle-ci est naturellement attirée par les points d'intérêt de l'image qui sont la plupart du temps liés au contenu et à son contexte (et a fortiori au *fond* de la vidéo).

Par exemple, pour une vidéo montrant un dialogue entre deux protagonistes, le visionneur va naturellement focaliser sa vision sur les interlocuteurs et beaucoup moins sur le fond. Tout algorithme tentant d'approximer la qualité perceptuelle doit donc dans l'idéal pondérer les imperfections qu'il trouve en fonction de leur probabilité qu'un cerveau humain les remarque. Cet aspect de positionnement dans l'image est la **composante spatiale** de la vidéo.

Estimer la qualité d'une image fixe et d'une vidéo sont deux exercices bien différents. En effet, la qualité des images d'une vidéo peut varier au fil du temps : il se peut qu'une vidéo soit globalement de bonne qualité mais comporte quelques instants où la qualité est moins bonne. Cet aspect met en avant la **composante temporelle** inhérente à toute vidéo, qu'il faut prendre en compte lorsque l'on réalise un estimateur de qualité.

Enfin, on parle de **composante spatio-temporelle** lorsque l'on prend en compte les deux composantes.

2 Vocabulaire lié aux vidéos

Cette partie recense quelques termes de base qui seront utilisés tout au long de ce rapport et qui sont utilisés de manière récurrente dans le domaine de l'analyse de vidéos.

Frame : Une image à un instant donné de la vidéo.

Framerate / Taux d'échantillonnage : Nombre d'images présentes pour chaque seconde de vidéo. On parle quelquefois de ***frames per second*** (abrégé *FPS*). Une valeur plus élevée signifie une vidéo plus fluide. Dans le cinéma, les normes oscillent entre 24 et 30 images par seconde selon les pays. Pour ce qui est des vidéos nécessitant un rendu plus "naturel", elles ont souvent un taux d'échantillonnage de 60 images par secondes (dû au fait qu'une majorité d'écrans soient cadencés à 60Hz).

Résolution : La taille, en pixels, des *frames* de la vidéo (au format $largeur \times hauteur$). Une résolution plus élevée donne une image plus fine où plus de détails sont perceptibles. En contrepartie, le moindre défaut

s'aperçoit de façon très nette (là où les basses résolutions ont tendance à en partie les masquer par leur rendu approximatif et "baveux"). On trouve différents standards en terme de résolution :

- Les normes **basse définition** EDTV :
  - 480p (720x480)
  - 576p (720x576)
- Les normes **haute définition** HDTV :
  - 780p (1280x720)
  - *Full HD* - 1080p (1920x1080)
- Les normes **ultra-haute définition** UHDTV (encore peu répandues) :
  - *4K UHD* - 2160p (3840x2160)
  - *8K UHD* - 4320p (7680x4320)

3 Usages concrets

Voici quelques cas d'usages concrets de la recherche et de l'industrie de l'estimation de qualité perçue.

3.1 Compression vidéo

De nos jours, la **compression des vidéos** est une problématique extrêmement commune. Le nombre de plateformes en ligne de diffusion de vidéos ne cesse de croître, et de plus en plus d'internautes utilisent ce genre de services avec des applications toujours plus variées (divertissement, éducation...).

Cependant, les vidéos sont des fichiers volumineux, car les composantes de chaque pixel à chaque instant de la vidéo doivent pouvoir être connues du système de lecture qui restitue le fichier sous forme d'image à l'utilisateur. Les volumes de données brutes deviennent très vite excessifs : par exemple, une vidéo non compressée durant 1 minute avec une résolution basse définition 480p et un taux d'échantillonnage classique de 24 images par seconde représente **plus de 1,4 Go de données**. En 1080p, on monte à plus de 8,5 Go de données pour seulement une minute de vidéo.

Il est dès lors évident que dans un contexte où ces vidéos doivent être envoyées via des connexions internet aux débits descendants de l'ordre du Mo/s, la compression s'est imposée comme une nécessité. Les techniques de compression vidéo les plus efficaces **déteriorent volontairement l'image** afin de diminuer drastiquement le volume de données présent en contrepartie d'une perte de qualité maîtrisée. Cette perte de qualité doit être raisonnablement légère afin de ne pas trop nuire à l'expérience de visionnage de l'utilisateur final.

C'est ici que les méthodes d'**estimation de qualité vidéo** jouent un rôle prépondérant. En effet, sachant que le but final est que **la compression soit la meilleure possible tout en nuisant le moins possible à la qualité perçue** par l'utilisateur final, les méthodes d'estimation de la qualité perceptuelle s'avèrent être les plus adaptées pour estimer l'impact de la compression. Cependant, comme déjà évoqué auparavant, réaliser ces tests de nombreuses fois pour évaluer la qualité d'un algorithme de compression serait extrêmement coûteux. Avoir un outil automatisé permet donc d'à la fois faciliter le processus, de l'accélérer et d'en éliminer le coût financier.

3.2 Evaluation de Quality of Experience (QoE)

De nombreux services de diffusion vidéo existent (que ce soit via Internet ou pour la télévision numérique) et doivent s'assurer de la qualité de l'expérience utilisateur finale. Pour cela, les fournisseurs de ces services utilisent des sytèmes temps-réel évaluant en permanence la qualité de leur flux vidéo afin de s'assurer que la Quality of Experience (QoE) de leur service reste maximale.[2]

4 Cycle de vie d'une vidéo et artefacts

Une vidéo, de son acquisition par une caméra à son affichage sur l'écran de l'utilisateur, traverse de nombreuses étapes durant lesquelles ses données sont altérées d'une manière ou d'une autre : c'est son **cycle de vie**. Voici les cinq étapes auxquelles une vidéo est communément soumise :

1. L'**acquisition de l'image** au moyen d'une caméra
2. L'**encodage du fichier** afin de diminuer significativement sa taille et de faciliter son transport
3. Le **transfert** vers l'ordinateur de l'utilisateur
4. Le **décodage**
5. L'**affichage** sur l'écran de l'utilisateur

Chacune de ces étapes peut introduire son lot d'imperfections à la vidéo, diminuant de façon effective la qualité perçue par l'utilisateur au bout de la chaîne. On appelle ces défauts des **artefacts**.

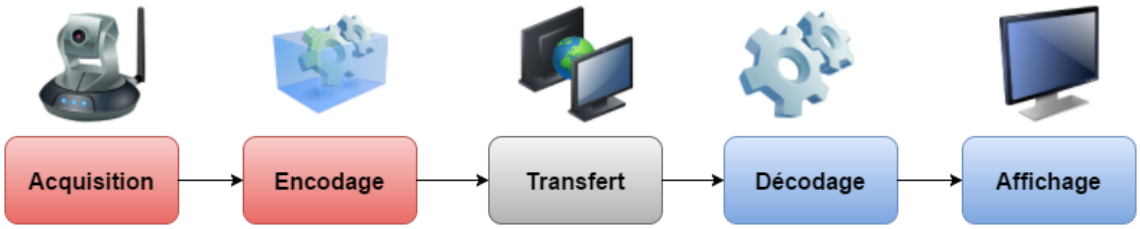


Figure 1 – *Cycle de vie classique d'une vidéo*

Lors de l'acquisition, on peut observer l'apparition d'un **flou** (qui peut être focal, de mouvement...) ou de **bruit** sous certaines conditions, qui vont de l'appareil de capture jusqu'aux conditions de prise de la scène (luminosité, exposition, mouvement...). L'objectif n'est pas de rentrer en détail dans des considérations poussées sur le domaine de la photographie, mais il faut savoir que certains artefacts s'introduisent dans une vidéo avant même toute opération de manipulation de données.

Lors de l'encodage, de nombreuses déteriorations d'image volontaires peuvent avoir lieu afin de réduire le volume des données (principe de **compression**).

Lors du transfert du fichier, le type de transfert est important. En effet, un téléchargement direct est vérifié une fois terminé, afin de s'assurer que le fichier obtenu chez le client est bien le même que celui initialement envoyé par le serveur. Cependant, dans le cas des transferts en flux (le ***streaming*** en est un exemple), il est possible que des paquets se perdent sur le réseau, entraînant inévitablement l'apparition d'artefacts.

Il est intéressant de noter que beaucoup de méthodes d'encodage modernes proposent des mécanismes de **récupération d'erreurs** (autrement appelés *stratégies de masquage* ou *concealment strategies*) afin de limiter l'impact de ce genre d'événements sur l'image finale. Pour une image manquante, on peut par exemple tenter de la reconstituer en réalisant une interpolation avec l'image précédente et l'image suivante.

Enfin, le type d'artefacts les plus gênants sont ceux survenant à l'affichage de la vidéo, car ils ne proviennent pas de la vidéo en elle-même mais de l'écran servant à l'afficher aux yeux de l'utilisateur. Là où une bonne configuration du moniteur peut cacher certains artefacts mineurs, une mauvaise configuration peut à l'inverse en introduire en faussant le rendu de l'image. Le grand paradoxe est que ces artefacts peuvent grandement affecter la qualité perçue par l'utilisateur mais ne peuvent pas être repérés par des outils d'analyse de vidéo, puisqu'ils sont propres au dispositif d'affichage de chaque utilisateur. Nous n'avons donc pas d'autre choix que de les ignorer.

5 Différents types d'estimateurs

Dans le cadre de l'estimation de qualité vidéo perçue, de nombreux **estimateurs** (ou *métriques*) ont vu le jour au fil des années[5], donnant chacun une approche différente pour l'évaluer et ayant pour certains un champ d'application bien précis. Ces métriques peuvent être classées en fonction de plusieurs paramètres.

5.1 Notion de référence

Les estimateurs peuvent être séparés en trois catégories distinctes selon la **référence** sur lesquels ils travaillent :

— Les **estimateurs "*Full Reference*" (FR)** qui comparent les séquences de la vidéo compressée / dégradée avec celles de la vidéo originale. C'est le cas pour un algorithme de compression, qui peut comparer la vidéo avant et après la compression. Ce genre d'estimateurs a le défaut de comparer davantage les dissimilarités entre la vidéo originale et la vidéo après traitement plutôt que de réellement tenter de trouver des défauts de qualité. Un traitement sur une vidéo peut avoir pour objectif d'améliorer la qualité de l'image, et les différences seront considérées comme des pertes de qualité là où elles sont en réalité tout l'inverse.

— Les **estimateurs "*Reduced Reference*" (RR)** qui effectuent des mesures ciblées sur la vidéo originale, et qui utilisent ces mesures pour analyser la vidéo compressée plutôt que de la comparer à la version originale. Ces estimateurs sont principalement utilisés pour du monitoring temps réel de la qualité, grâce à leur coût de calcul réduit par rapport à une analyse FR.

— Les **estimateurs "*No Reference*" (NR)** qui analysent la vidéo compressée sans avoir aucune connaissance de la vidéo d'origine. De tels estimateurs peuvent trouver leur utilité dans l'estimation de la **Quality of Experience** (qualité de l'expérience utilisateur, souvent abrégée QoE) ; ou encore dans des systèmes d'amélioration de la qualité (*video enhancements*) qui doivent juger au préalable si, oui ou non, la vidéo requiert des améliorations.

Il est intuitivement plus facile d'estimer la qualité en disposant de la vidéo d'origine (en *Full Reference*) qu'en ne possédant que la vidéo dégradée (en *No Reference*).

Dans le cadre de notre projet, nous allons nous focaliser sur les estimateurs NR car ce sont ceux pouvant être utilisés dans le plus grand nombre de cas.

5.2 Types de métriques

Selon les types de données qu'elles traitent, les métriques peuvent être classées en différentes catégories :

— Les **métriques de données** qui traitent mathématiquement le signal de la vidéo, sans se soucier du rendu des images en lui-même. La qualité perçue est donc estimée en jugeant la qualité du signal, sans réellement prendre en compte l'impact d'anomalies du signal sur l'image (et a fortiori, la qualité). Ces métriques sont longtemps restées très populaires par leur simplicité, mais leurs résultats ne sont pas au niveau de celles des autres catégories. Quelques exemples de métriques de ce type sont la ***Mean Squared Error*** (MSE) ou le ***Peak to Signal Noise Ratio*** (PSNR).

— Les **métriques d'image** qui traitent l'image en elle-même, à la recherche d'indices sur sa qualité. On distingue deux approches majeures pour les métriques d'image :

1. L'approche de **modélisation de la vision**, qui utilise des caractéristiques connues du système de vision humain en se basant sur des données provenant d'expériences psychophysiques. La qualité est ainsi déduite de caractéristiques comme la sensibilité au contraste ou la perception des couleurs.

2. L'approche **ingénieur** (*engineering approach*), qui se base sur les principes d'extraction de caractéristiques et de détection d'artefacts pour estimer la qualité des images contenues dans la vidéo.

— Les **métriques réseau**, dédiées à l'évaluation de l'impact de la transmission d'une vidéo par Internet sur la qualité perçue par l'utilisateur final. Ce type de métriques ont vu le jour en parallèle avec l'explosion des services de vidéo à la demande et de streaming vidéo qui pullulent désormais sur Internet ; et elles permettent par exemple d'estimer la perte en qualité perçue en fonction du taux de perte de paquets.

— Les **métriques hybrides**, combinant tout ou partie des autres types de métriques présentés ci-dessus. On peut ainsi à la fois prendre en compte la qualité du signal, effectuer de la détection d'artefacts et du monitoring réseau afin de tenter d'évaluer au mieux la qualité. Ces métriques sont potentiellement les plus puissantes, mais sont difficiles à concevoir car la pondération de chaque élément dans la qualité finale estimée est très difficile à juger, et cela requiert beaucoup d'essais et d'affinage pour arriver à un résultat concluant.

Dans le cadre de ce projet, nous nous sommes principalement orientés vers l'**approche ingénieur** des métriques d'image, sans pour autant négliger les caractéristiques des autres métriques qui pourraient être ajoutées pour former une métrique hybride. Les détails de la métrique conçue seront abordés plus loin dans ce rapport.

6 Quelques considérations sur la détection d'artefacts

Dans l'optique de réaliser un système de détection d'artefacts visuels, il faut prendre certains éléments en considération : leur **visibilité** et leur **pertinence**.

6.1 Visibilité d'un artefact

Un artefact n'est pas toujours visible par l'œil humain même s'il est bel et bien présent dans une vidéo. Cela peut-être causé par :

— une présence **extrêmement courte** : l'artefact n'apparaît que pendant quelques *frames* seulement, à la manière d'images subliminales. L'image est alors inconsciemment perçue par le cerveau humain, mais ses détails visuels ne le sont pas et n'influenceront donc pas sa qualité perçue. C'est la composante temporelle de la vidéo qui est mise en jeu.

— une **présence très discrète** : l'artefact n'altère pas suffisamment l'image pour que cela soit perceptible pour l'utilisateur ou réellement impactant sur la qualité perçue. C'est la composante spatiale de la vidéo qui est mise en jeu.

Dans les deux cas, des mesures de filtrage doivent être prises pour prendre en compte ces cas particuliers. Comme nous le verrons par la suite, cela se réalise au moyen de techniques de **seuillage**.

6.2 Pertinence d'un artefact

Quelquefois, il arrive que certains "défauts" d'image soit volontaires : il peut s'agir d'effets visuels volontairement introduits dans la vidéo lors de sa conception. Par exemple, on peut observer du **flou de mouvement** dans des films d'action ou du **blocking** dans une vidéo contenant une image subdivisée en carrés réguliers. On dit alors que ces artefacts ne sont pas **pertinents**, et aucun ne sera considéré comme tel par un visionneur humain dans son jugement de la qualité perçue.

La différence majeure entre le point de vue humain et celui de l'algorithme est la compréhension du **contexte de la vidéo** qui justifie la présence de ces artefacts. Un algorithme de détection, lui, se fie

seulement aux images qui lui sont confiées (sans compréhension du contenu), et il est dès lors difficile d'évaluer la pertinence d'un artefact.

Dans le cadre d'estimateurs Full Reference (FR), on dispose de la vidéo initiale pour vérifier si ces effets étaient déjà présents avant toute déformation : il est donc relativement facile de déterminer s'ils sont intentionnels ou non. Dans notre cas, nous souhaitons réaliser des estimations No Reference (NR) et n'avons donc pas cette facilité.

7 Recensement des artefacts

Il est important de bien comprendre les éléments qui dégradent la qualité d'une vidéo afin de savoir le genre d'imperfections qu'un algorithme d'estimation doit repérer.

Le catalogue catalogue qui suit, bien qu'il soit loin d'être exhaustif, présente les artefacts les plus communément rencontrés.

7.1 Mosquito noise

Le "***Mosquito noise***" est un type de bruit particulier et reconnaissable survenant lors de l'**encodage**. Cet effet, typique de compressions basées sur la technique **DCT** (*Transformée en cosinus discrète*), fait apparaître de petits points au niveau des bordures des objets. Il se retrouve également dans les vidéos avec un léger déplacement et scintillement de ces points au fil des *frames*, ce qui peut s'apparenter à des moustiques tournant autour des objets (d'où le nom donné à cet artefact).

Figure 2 – *Exemple de "Mosquito noise"*

Cet effet est dû à une perte d'informations lors de l'encodage, qui oblige à effectuer des approximations lors du décodage en inversant la transformation. La technique DCT étant utilisée dans l'ensemble des encodages de la famille MPEG ainsi que dans la compression d'images JPEG, il s'agit d'un artefact couramment rencontré.

7.2 Blocking

Le **blocking** est une détérioration majeure de l'image qui devient criblée de blocs pendant une courte durée. Ce genre d'artefact est l'un des rares pouvant survenir lors du **décodage**. En effet, lors de l'utilisation d'un encodage à base de ***compensation de mouvement*** (ou *Motion compensation*), les *frames* sont déduites à partir des *frames* précédentes et de vecteurs de transformation. Cela est bien plus léger que de stocker chaque *frame* indépendamment, et a donc été massivement utilisé dans de nombreuses méthodes de compression telles que MPEG-1, MPEG-2 et MPEG-4.

Il arrive cependant quelquefois que des paquets se perdent lors du transfert de la vidéo (notamment en téléchargement par flux, comme du streaming), omettant certains vecteurs de transformation et

perturbant ainsi les *frames* à venir pendant plusieurs secondes. Les méthodes d'encodage proposent lors du décodage des **systèmes de prédiction de la compensation de mouvement** afin de tenter de "deviner" les parties de la vidéo qui ne sont pas arrivées à bon port. Cela a tendance à limiter les dégâts, mais n'empêche pas cet effet de *blocking* de se produire, comme on peut le voir sur l'image qui suit :

Figure 3 – *Exemple de blocking*

7.3 Ringing

L'effet de ***ringing*** est une formation d'artefacts qui se manifestent le plus souvent par la formation d'un halo ou de défauts de couleur autour des formes aux bords nets. De la même manière que pour le *Mosquito Noise*, cet artefact est typique des transformations basées sur la transformée en cosinus discrète (ou DCT). Cela est dû au fait que la DCT engendre des pertes de précision sur les éléments de haute fréquence qui sont les bordures nettes dans le cas des images.

Figure 4 – *Exemple de ringing sur une image*

7.4 Aliasing

L'**aliasing** est un effet "d'escaliers" qui apparaît lors de l'**acquisition** si la résolution de capture de la vidéo est trop faible. Les contours arrondis ou obliques peuvent alors apparaître légèrement "hachurés" à l'œil nu, comme sur le bord des lunettes dans l'image suivante.

Ce problème, également rencontré dans le domaine des rendus 3D (notamment dans les jeux-vidéos), peut être en partie corrigé par des méthodes d'***anti-aliasing*** qui lissent ces bordures en créant des "pixels de transition" autour de celles-ci, faisant apparaître la forme comme plus naturelle pour l'œil humain (mais toujours moins nette que si la résolution initiale avait été plus élevée).

Figure 5 – *Exemple d'aliasing*

7.5 Rémanence / Ghosting

La **rémanence** (ou ***ghosting***) est un artefact survenant à l'**affichage**, principalement sur des écrans LCD (qui sont majoritaires de nos jours). Il se manifeste par des "traînées" derrière les éléments en mouvement, et s'explique par une semi-persistance (en transparence) des *frames* précédentes par dessus les *frames* actuelles. Il ne peut pas être détecté par un quelconque algorithme d'estimation puisqu'il dépend de l'écran de l'utilisateur.

Figure 6 – *Exemple de ghosting sur un écran LCD*

7.6 Autres artefacts

Comme précisé précédemment, ce document n'a pas pour vocation à proposer une liste exhaustive des artefacts existants, seulement les plus courants. Quelques autres types d'artefacts peuvent être occasionnellement rencontrés : parmi eux, on peut compter le **flou de mouvement** (ou motion blur), le ***tearing*** ("déchirure" de l'image à l'affichage lorsque la fréquence d'affichage est trop élevée), ou encore une **mauvaise restitution des couleurs et/ou de la luminosité** à l'écran.

8 Standards de la qualité vidéo et VQEG

Pour que les nombreux estimateurs existants puissent être comparés sur un même pied d'égalité, il s'est avéré nécessaire d'introduire une forme de **standardisation** autour des études de la qualité vidéo et sur

ce qui est considéré comme une bonne estimation ou une mauvaise estimation.

Cette lourde tâche revient au ***Video Quality Experts Group*** (VQEG) [WWW6] qui a réalisé un ensemble de vidéos de test de contenu et de forme variés ainsi que des protocoles de tests rigoureux permettant de formaliser le processus et d'assurer la qualité des résultats d'éventuelles études comparatives. Ce groupe a été fondé en 1997 en tant qu'organisation indépendante non lucrative, et il est composé d'experts du milieu de la recherche en qualité vidéo (comme son nom l'indique).

Le VQEG s'occupe de récolter les besoins des industriels en terme d'analyse de la qualité vidéo afin d'adapter ses standards et d'aiguiller la recherche vers ces besoins. Les recherches sont en majorité menées par des équipes académiques de recherche réparties dans le monde entier, d'où un besoin évident de standardisation afin de coordonner les efforts scientifiques pour voir le domaine progresser plus rapidement.

Figure 7 – *Logo du Video Quality Experts Group*

Le groupe est structuré en deux grands ensembles :

— Les **groupes de travail** (*work groups*), effectuant des recherches et des développements sur divers sujets liés à la qualité vidéo.
— Les **groupes de soutien** (*support groups*), réalisant du contenu, des tests et des analyses statistiques pour appuyer le travail des *work groups*.

Le groupe de travail produisant les résultats les plus intéressants en rapport avec ce projet est le ***Video and Image Models for consumer content Evaluation*** (VIME), qui oeuvre à développer de nouvelles méthodes NR pour estimer la qualité perceptuelle des vidéos. Ce groupe de travail est très récent, puisque sa création remonte à juillet 2014. Cela montre que ce domaine est en pleine expansion et que de réels besoins industriels existent en rapport avec cette problématique.

9 Impact de la qualité audio

Il ne faut pas oublier qu'une vidéo n'est pas exclusivement composée d'images, mais également de son. Que ce soit des musiques en fond ou les paroles d'une discussion, la qualité de l'audio perçu par l'utilisateur a son importance. En effet, un son grésillant ou des voix désynchronisées avec l'image peuvent très nettement dégrader l'expérience de ce dernier, et cela ne serait pourtant pas perçu par un algorithme qui se focalise uniquement sur la qualité de l'image.

Peu de métriques prennent en compte l'audio dans leur analyse, et elles ont même leur propre classification empirique d'**estimateurs audio-visuels**. Pour souligner leur aspect expérimental, il est intéressant de noter que le VQEG a un groupe de travail entièrement dédié à ces estimateurs : l'***Audiovisual High Definition*** (AVHD).

Le plus souvent, ces métriques se basent à la fois sur la **synchronisation entre l'image et le son** (principe de synchronisation labiale ou *lip-sync*) et sur la qualité du signal audio en lui-même (recherche d'artéfacts, de la même manière que pour les images).

2 Framework généraliste pour l'estimation de qualité

1 Vue d'ensemble

Le framework qui suit a été proposé dans le papier de Hemami et Riebman intitulé "**No-reference image and video quality estimation : Applications and human-motivated design**"[1]. Il préconise une base de travail fixe dans l'estimation de qualité perceptuelle NR qui se subdivise en trois grandes phases :

1. Une phase de **measuring**
2. Une phase de **pooling**
3. Une phase de **mapping to quality** (classification)

Nous allons voir en détail chacune de ces phases, ainsi que les différents algorithmes qu'il est possible d'implémenter pour chacune d'entre elles.

2 Measuring

Durant la phase de *measuring*, l'estimateur effectue des **mesures** sur la vidéo donnée en entrée : ce sont les **caractéristiques** du flux vidéo. Les différentes caractéristiques que l'outil tente d'extraire dépendent du but final de l'estimateur (les éléments de qualité que l'on veut tout particulièrement mettre en avant, par exemple).

L'extraction de caractéristiques depuis une vidéo est une opération extrêmement volumineuse en mémoire si l'on tente de tout mesurer sur l'ensemble d'une longue vidéo.
En effet, sur de grandes images, les caractéristiques étant extraites pour chaque pixel de l'image, on obtient très vite des données de l'ordre du méga-octet. Si l'on transpose cela à chaque *frame* pour une longue vidéo en haute résolution et pour chaque caractéristique, il ne s'agit qu'une question de temps avant d'atteindre l'ordre du giga-octet, voire même du tera-octet.

C'est pourquoi il est essentiel de choisir judicieusement la façon dont ces mesures sont effectuées. On pourra par exemple effectuer des mesures à des intervalles réguliers (par exemple, 5 frames toutes les 50 frames de vidéo), ce qui permet de garder l'essentiel de l'information à propos de la qualité tout en divisant par 10 les quantités de données obtenues.

2.1 Gradient

Dans le domaine du traitement de l'image, le **gradient** est la dérivée de l'image étudiée selon un axe donné [WWW2]. Par exemple, il est possible d'effectuer un gradient horizontal afin de comparer les intensités des pixels horizontalement voisins, permettant de mettre en évidence l'évolution de l'intensité au fil de l'axe X. Les gradients sont très souvent utilisés pour effectuer des détections de contours, et constituent une des pierres angulaires des mesures en traitement de l'image.

Dans notre cas, en traitement vidéo, l'axe **temporel** (t) s'ajoute aux axes **horizontal** (X) et **vertical** (Y), permettant de mesurer l'évolution des intensités au fil des *frames* de la vidéo.

Mathématiquement parlant, le gradient est un **opérateur de convolution** suivant la formule suivante :

$$\nabla f = \begin{bmatrix} g_x \\ g_y \end{bmatrix} = \begin{bmatrix} \frac{\partial f}{\partial x} \\ \frac{\partial f}{\partial y} \end{bmatrix}$$

où $\frac{\partial f}{\partial x}$ est le gradient horizontal et $\frac{\partial f}{\partial y}$ est le gradient vertical.

Le filtre de convolution le plus simple utilisable pour le réaliser est le suivant : $\begin{bmatrix} -1 \\ 0 \\ +1 \end{bmatrix}$

2.2 Flot optique

Le **flot optique** est une mesure consistant en l'**estimation du mouvement apparent des objets** entre deux frames consécutives. Ce mouvement peut aussi bien être causé par un mouvement de l'objet que par un mouvement de l'observateur (en l'occurence, de la caméra) [WWW4].

Il s'exprime comme un champ de vecteurs à deux dimensions où chaque vecteur représente le déplacement des points depuis la frame de référence vers la suivante. Pour estimer le flot optique, on part du principe que les pixels voisins aux pixels traités ont un mouvement similaire, et que l'intensité des pixels ne varie pas significativement entre les deux frames (pas de clignotement brusque, par exemple).

On parle d'*estimation* du flot optique car l'équation du flot optique proprement dit présente trop d'inconnues et est donc insoluble. Différentes méthodes ont donc été conçues pour pallier à ce problème. L'une d'entre elles, la **méthode de Lucas-Kanade** [3], est particulièrement répandue pour traiter un nombre limité de points. Cette méthode utilise des carrés de 3x3 pixels pour lesquels on assume que tous les points suivent le même mouvement.

Obtenir le flot optique $(\mathrm{V}_x, \mathrm{V}_y)$ revient alors à résoudre le système d'équations suivant :

$$\begin{bmatrix} \mathrm{V}_x \\ \mathrm{V}_y \end{bmatrix} = \begin{bmatrix} \sum_i w_i \mathrm{I}_x(q_i)^2 & \sum_i w_i \mathrm{I}_x(q_i)\mathrm{I}_y(q_i) \\ \sum_i w_i \mathrm{I}_x(q_i)\mathrm{I}_y(q_i) & \sum_i w_i \mathrm{I}_y(q_i)^2 \end{bmatrix}^{-1} \begin{bmatrix} -\sum_i w_i \mathrm{I}_x(q_i)\mathrm{I}_t(q_i) \\ -\sum_i w_i \mathrm{I}_y(q_i)\mathrm{I}_t(q_i) \end{bmatrix}$$

où les q_i sont les pixels du carré, et $\mathrm{I}_x(q_i), \mathrm{I}_y(q_i), \mathrm{I}_t(q_i)$ les dérivées partielles de l'image I par rapport à la position x et y et au temps t évaluées au point q_i pour le moment donné.

Cette méthode a beau être efficace pour quelques points et pour des mouvements relativement lents, elle perd en efficacité lorsque des mouvements rapides surviennent (à cause de son fonctionnement local) ou lorsque l'on doit effectuer le calcul pour l'ensemble des points d'une *frame* donnée. Pour cela, d'autres algorithmes existent, dits de "**flot optique dense**" qui permettent de calculer ce dernier pour l'ensemble des pixels d'une frame, avec des méthodes plus adaptées. Cette vision d'ensemble permet de détecter des mouvements plus grands, et de calculer le champ de vecteurs complet en une seule fois [WWW3]. Une méthode efficace pour le calcul de flots optiques denses est **l'algorithme de Farneback** (détaillé dans le papier "Two-Frame Motion Estimation Based on Polynomial Expansion" de Gunnar Farnebäck, 2003).

3 Pooling

Une fois les mesures effectuées, il est nécessaire de les "synthétiser" durant une phase de ***pooling*** pour les rendre exploitable au sein d'un algorithme de classification. La phase de *pooling* consiste donc en un **rassemblement** des mesures effectuées lors de l'étape précédente [WWW1]. Le principe est simple : on agrège les caractéristiques d'une région de l'image en suivant une **stratégie de pooling** (qui peut être une moyenne, un maximum...) afin de former des blocs pour lesquels on obtient une unique valeur en rapport avec la caractéristique. Cela permet d'alléger drastiquement les quantités de données manipulées et n'engendre pas de perte d'information significative si les régions sont intelligemment choisies.

Le *pooling* est une étape nécessaire, car la vidéo qui est traitée doit être intégrée d'une manière ou d'une autre à la base d'apprentissage une fois la qualité déduite. Or, intégrer des données aussi précises (avec autant de vecteurs) nuirait à moyen et long terme sur la capacité de généralisation du système : on parle également de **sur-apprentissage**.

Dans un cas de sur-apprentissage, le système apprend "par cœur" plutôt que d'essayer de comprendre les *patterns* (motifs) qui sont révélateurs d'une bonne ou d'une mauvaise qualité. Si on lui donne une autre vidéo par la suite, il regardera davantage si la nouvelle vidéo étudiée ressemble (au sens strict) à une vidéo qu'il a déjà étudiée pour pouvoir lui attribuer la même qualité, plutôt que de tenter de reconnaître des *patterns* qu'il a déjà observé pour en déduire la qualité.

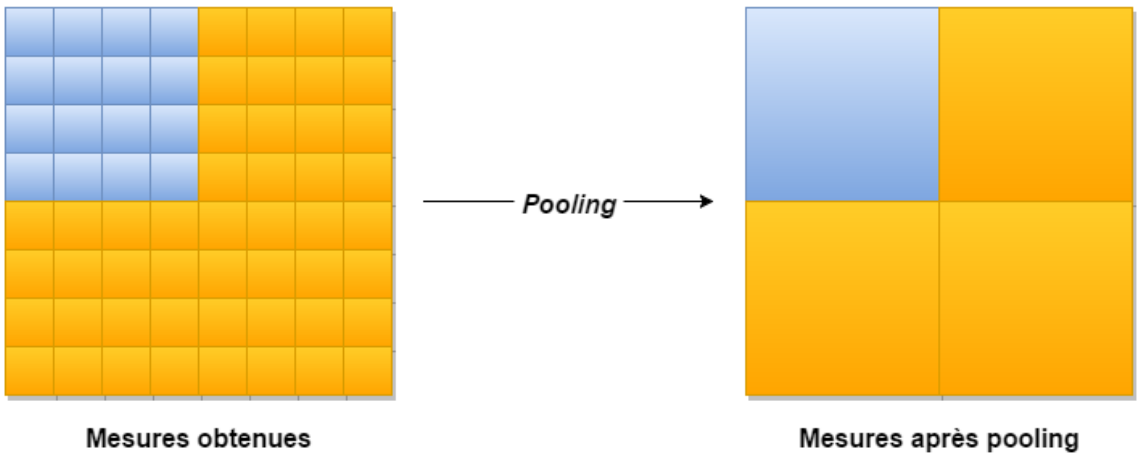


Figure 1 – *Schéma représentant la phase de pooling*

Dans le cadre d'une vidéo, de nombreux **axes de pooling** existent pour choisir en fonction de quelle notion on souhaite regrouper les données. Il est possible d'effectuer du pooling sur des axes pouvant combiner ou non les notions d'**espace**, de **temps**, d'**orientation** et de **fréquence spatiale**. En voici quelques exemples :

- Le pooling **spatial**, regroupant les données sous formes de régions d'une seule *frame* de la vidéo.
- Le pooling **temporel**, regroupant les données pour un même pixel sur différentes *frames* consécutives de la vidéo.
- Le pooling **spatio-temporel**, regroupant les données pour une région de pixels sur différentes *frames* de la vidéo.

Nous allons désormais voir différentes stratégies de pooling, utilisables sur n'importe lequel des axes ci-dessus.

3.1 Maximum / Minimum

Un pooling intuitif (et relativement inefficace) est l'application d'un maximum ou d'un minimum sur l'ensemble des données *poolées*. Selon la mesure sur laquelle on applique cette stratégie, l'un ou l'autre peut se réveler plus adapté. On a donc :

$$f_{max} = \max x_i (\forall i \in [1, \mathrm{N}])$$
$$f_{min} = \min x_i (\forall i \in [1, \mathrm{N}])$$

où les x_i sont les valeurs *poolées* et N est leur nombre.

3.2 Pooling de Minkowski

Une stratégie de pooling légèrement plus élaborée est celle de Minkowski, consistant à effectuer une moyenne des valeurs élevées à une puissance p [4]. Ainsi, pour $p = 1$, on a une moyenne classique sur les valeurs, tandis que pour $p = 2$, on effectue une moyenne des valeurs élevées au carré, ce qui permet d'amplifier les écarts de valeur pour une mesure particulière. La fonction du pooling de Minkowski est la suivante :

$$f_{Minkowski} = \frac{1}{\mathrm{N}} \sum_{i=1}^{\mathrm{N}} x_i^p$$

3.3 Pooling pondéré

Plusieurs stratégies de pooling utilisent un système de pondération, où des poids différents sont attribués à chacune des valeurs poolées en fonction de différents facteurs, selon ce que l'on veut mettre en avant [4]. Par exemple, il est possible d'évaluer la saillance (à quel point le pixel "ressort" parmi ses voisins) afin de connaître les points d'intérêts en terme de contenu et de valoriser ces points d'intérêts, qu'on suppose davantage remarqués par celui qui visionne la vidéo que les autres pixels.

D'un point de vue mathématique, le pooling pondéré peut s'exprimer comme suit :

$$f_{Weighted} = \frac{\sum_{i=1}^{\mathrm{N}} w_i x_i}{\sum_{i=1}^{\mathrm{N}} w_i}$$

où les w_i sont les poids attribués aux valeurs x_i correspondantes.

4 Mapping to Quality

La dernière phase dite de "*Mapping to quality*" consiste en l'association de la vidéo étudiée à un niveau de qualité. Ce niveau est obtenu par une **classification** en utilisant des **données d'apprentissage**. Par exemple, on peut imaginer qu'on fonctionne avec 10 classes représentant 10 niveaux de qualité différents, contenant chacune des données caractéristiques de la qualité correspondante.

Lors de l'étape de *Mapping to Quality*, les caractéristiques issues du *pooling* sont donc comparées avec celles contenues dans la base d'apprentissage pour en déduire la classe à laquelle cette vidéo devrait appartenir. Une grande variété d'algorithmes de classification existent et sont utilisables ici, tous ayant des performances différentes dans des cas différents.

En voici quelques exemples :

- La méthodes des **K-moyennes** (K-means)
- Les **Support Vector Machines** (SVM)
- Les algorithmes de **réseau de neurones** (Perceptron multi-couches...)

3 Démarche de conception de l'application

1 Choix du langage : C++

Avant de débuter le développement, une phase de conception assez sommaire a été nécessaire afin d'être sûrs de produire un logiciel maintenable, compréhensible, et à la fois utilisable et modifiable dans le cadre de futurs travaux de recherches.

Le choix du langage et des outils s'est fait en premier et tout naturellement : pour pouvoir développer rapidement, l'usage de la **bibliothèque OpenCV** s'est imposé comme évident, et nous avons choisi le **langage C++** pour sa robustesse, ses performances et son paradigme objet.

2 Outils de reconnaissance de formes : OpenCV

OpenCV (pour "Open Computer Vision") est une bibliothèque open-source disponible sous licence BSD servant à la vision par ordinateur (et dans notre cas au traitement de l'image et à la reconnaissance de formes). Elle est multi-plateformes et présente des implémentations en C, C++, Java et Python. Elle est utilisée dans le monde entier par de très grandes entreprises et universités, et peut donc être considérée comme très fiable à ce titre [WWW5].

La majorité des algorithmes et notions de traitement de l'image et de reconnaissance de formes évoquées dans la première partie de ce rapport sont directement implémentées sous OpenCV, permettant de créer des prototypes d'outils efficaces rapidement. Cela permet de voir plus vite si les idées mises en pratiques s'avèrent efficaces ou non dans notre but (ici, l'estimation de la qualité perceptuelle).

Figure 1 – *Logo de la bibliothèque OpenCV*

3 Outil en console ou interface graphique ?

Une question pertinente est la présence ou non d'une interface graphique pour simplifier l'usage pour l'utilisateur finale. Sachant que l'outil est davantage un système automatisé (on donne une vidéo en entrée et on applique une procédure donnant un score de qualité) qu'un système interactif, l'ajout d'une interface graphique est loin d'être indispensable et coûterait un temps de développement précieux qui pourrait être utilisé à l'amélioration de l'efficacité de l'outil en lui-même.

Le choix d'**outil en console** semble donc être le plus judicieux.

L'application sera de toute manière réalisée de façon modulaire, ce qui permet l'ajout ultérieur d'une interface graphique si le besoin s'en fait ressentir.

4 Conception de l'architecture de l'application

Une fois que des réponses ont été apportées à ces questions basiques, il a été possible de se pencher sur la conception de l'application en elle-même. Cela s'est réalisé en plusieurs étapes.

4.1 Clarification du besoin

Avant de bondir dans la conception de l'application à proprement parler, il est important de redéfinir le besoin afin de s'assurer que rien ne manque et que rien n'est fait en trop.

Dans le cadre de notre outil, voici les fonctionnalités requises :

- On doit pouvoir choisir une vidéo présente sur le disque
- On doit pouvoir choisir parmi plusieurs méthodes de mesure
- On doit pouvoir choisir parmi plusieurs méthodes de pooling
- On doit pouvoir choisir parmi plusieurs méthodes de classification
- L'application doit effectuer les mesures, le pooling et la comparaison par rapport à une base d'apprentissage et donner le score de qualité

On observe dès lors que de nombreuses méthodes de mesure, de pooling et de classification seront accessibles à l'utilisateur. Pour qu'il soit simple d'ajouter de nouvelles méthodes à l'avenir et que le code soit facilement maintenable, il faudrait une structure générique permettant de substituer une méthode à une autre de façon totalement naturelle et sans impacter le reste du code. La manière la plus adaptée de réaliser cela en C++ passe par l'utilisation de **classes abstraites**.

En effet, il nous suffit de définir des composants génériques de mesure, de pooling et de classification (tous les 3 abstraits) contenant une fonction virtuelle pure qui effectue le travail requis (respectivement *measure*, *pool* et *mapToQuality*). Ainsi, à chaque nouvelle méthode que l'on souhaite ajouter au programme, il suffit de créer une nouvelle classe implémentant la classe abstraite voulue et dont l'implémentation de la fonction virtuelle effectue le travail de manière appropriée.

4.2 Classes déduites du besoin

Voici une liste de classes déduites du besoin exprimé ci-dessus :

- une classe **QualityEstimationTool** qui sert de "chef d'orchestre" à l'application, et instancie l'ensemble des autres éléments nécessaires à l'estimation de la qualité perçue

- des classes **Video** et **VideoFrame** représentant respectivement une vidéo et une frame de vidéo, permettant de faciliter leur manipulation par les autres classes

— des classes abstraites **MeasuringComponent**, **PoolingComponent** et **MappingToQualityComponent** servant de base à tous les composants de mesure, de pooling et de classification réalisés par la suite

— une classe **LearningDatabase** qui gèrera la base d'apprentissage. Pour l'instant, la gestion de cette dernière reste floue et sera éclaircie plus tard dans la phase de développement

4.3 Interactions entre classes

Un diagramme de classes (respectant de façon vague les normes UML) a été réalisé afin de mettre en évidence les relations entre classes et les attributs et méthodes fondamentales. Celui-ci est loin d'être exhaustif, et sert davantage à comprendre la structure de l'application :

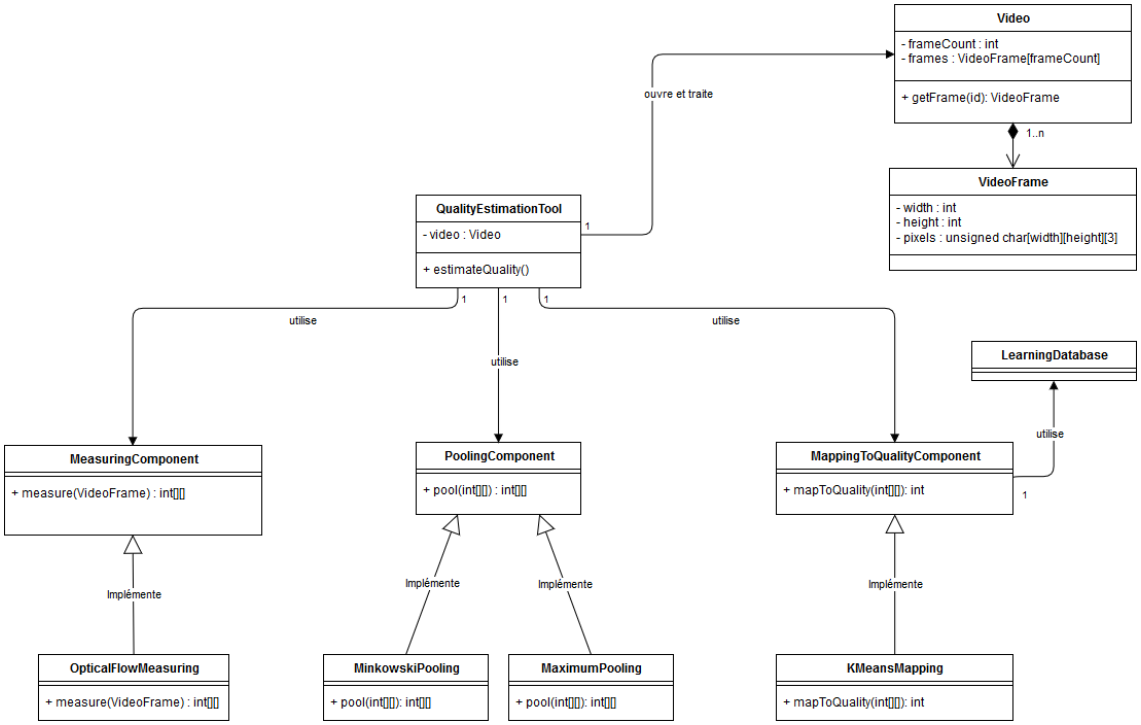


Figure 2 – *Diagramme de classes simplifié de l'application*

4 Planification prévisionnelle du projet

1 Méthodologie

Bien que ce projet soit davantage un projet de recherche scientifique qu'un ouvrage nécessitant une planification pointilleuse, un planning a été réalisé afin d'essayer de dégager des tâches à réaliser et d'avoir par conséquent une meilleure vision de ce qui a été fait et de ce qu'il reste à faire tout au long du projet.

La première étape pour parvenir à réaliser ce planning a été de subdiviser le projet en taches.

2 Tâches du projet

Voici les tâches qui ont pu être dégagées de l'analyse réalisée :

1. État de l'art (partie recherche)
   - **Description** : Recherche étendue sur la problématique de la qualité perçue, tout particulièrement sur l'état actuel de la communauté scientifique sur ce domaine. Acquisition et restitution des notions requises, citation des sources.
   - **Livrable** : Rapport de recherche synthétisant l'état de l'art réalisé

2. Développement de l'architecture de base
   - **Description** : Réalisation de l'architecture de base de l'application (création de l'ossature des classes abstraites, des classes servant de support aux vidéos...). Cette phase inclut la mise en place de l'environnement de développement, la création du projet et la configuration de tout ce qui doit l'être pour pouvoir développer sans encombres par la suite.
3. Chargement d'une vidéo et traitement simple
   - **Description** : Implémentation du module de chargement vidéo, et réalisation du traitement le plus simple possible afin de voir l'ensemble de la *processing chain* en action.

4. Gestion de l'auto-apprentissage
   - **Description** : Ajout du système d'apprentissage par le logiciel (où chaque élément traité est incorporé au sein d'une base d'apprentissage qui sert de référence pour les traitements ultérieurs). Nécessité d'établir une base d'apprentissage de départ.
   - **Livrable** : Outil rudimentaire mais fonctionnel, bien que peu performant à ce stade

5. Évaluation de l'outil
   — **Description** : Réalisation et application d'un protocole d'évaluation des performances de l'outil. Comme aucune ressource publique n'a pu être trouvée concernant ce domaine, des vidéos standards pourront être utilisée, et le protocole devra être établi.
   — **Livrable** : Un tableau de résultats sur un ensemble de vidéos bien définies.

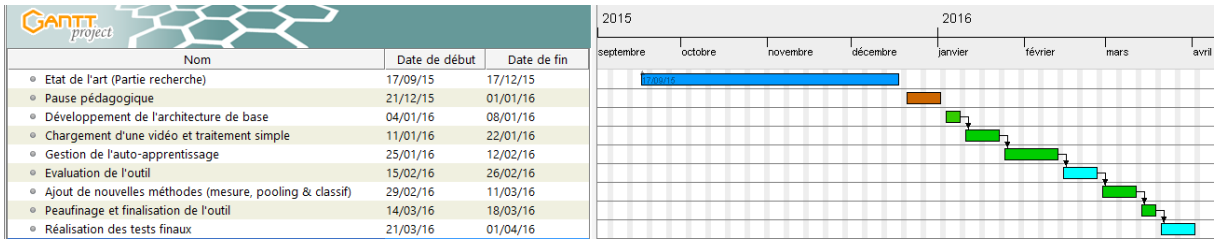
6. Ajout de nouvelles méthodes (mesure, pooling & classif)
   — **Description** : Ajout de multiples méthodes, dont celles vues dans l'état de l'art, et si le temps le permet d'autres plus poussées.

7. Peaufinage et finalisation de l'outil
   — **Description** : Finalisation du code, de manière à ce que tout soit documenté, commenté et compréhensible pour une éventuelle reprise ultérieure.
   — **Livrable** : Un code propre, avec éventuellement un manuel de reprise si cela est nécessaire.

8. Réalisation des tests finaux
   — **Description** : Réalisation des tests finaux en suivant le protocole établi à la première séance de tests. Comparaison des résultats.
   — **Livrable** : Tableau de résultats finaux.

3 Planning prévisionnel

De cette subdivision en tâche, il a été possible d'estimer des durées et de former un planning prévisionnel sous la forme d'un diagramme de Gantt. Voici la version finale de ce diagramme :



Nom	Date de début	Date de fin
Etat de l'art (Partie recherche)	17/09/15	17/12/15
Pause pédagogique	21/12/15	01/01/16
Développement de l'architecture de base	04/01/16	08/01/16
Chargement d'une vidéo et traitement simple	11/01/16	22/01/16
Gestion de l'auto-apprentissage	25/01/16	12/02/16
Evaluation de l'outil	15/02/16	26/02/16
Ajout de nouvelles méthodes (mesure, pooling & classif)	29/02/16	11/03/16
Peaufinage et finalisation de l'outil	14/03/16	18/03/16
Réalisation des tests finaux	21/03/16	01/04/16

Figure 1 – *Planning prévisionnel réalisé sous GanttProject*

4 Fiabilité du planning

La fiabilité de ce planning est limitée, n'ayant pas une grande expérience en évaluation de la durée des tâches de développement. Cependant, il représente une sorte "d'idéal" à respecter, et les durées des différentes tâches ont été intentionnellement revues à la hausse afin de donner du temps supplémentaire si un imprévu survenait (tâche non prédite, tâche plus longue que prévue...). Ce temps excédentaire pourra malgré tout être utilisé en fin de projet si tout se passe sans encombres.

Deuxième partie

Développement

5 Méthodologie de suivi et de gestion du projet

1 Contact avec l'encadrant

Ce projet n'a fait intervenir qu'un nombre très limité d'acteurs : l'encadrant (Donatello CONTE) et l'étudiant (Anthony DEMARCY). Il s'agissait d'un projet de recherche scientifique sur un sujet relativement novateur, et non pas d'un ouvrage sur des éléments parfaitement maîtrisés.

M. Conte a été très coopératif tout au long du projet et venait toutes les semaines prendre compte de l'avancement des choses, en apportant des réponses aux questions et en donnant des pistes d'exploration utiles. Les rares semaines où il n'a pas pu venir, je me suis efforcé de l'informer par mail de l'avancement des choses et d'aller le voir dans son bureau dans les très rares cas où j'étais bloqué au niveau de mes recherches. Le contact était donc direct et informel, ce qui a permis au projet d'avancer vite et aux informations de circuler facilement.

Comme nous nous voyions chaque semaine pour faire le point, le besoin de faire des comptes-rendus hebdomadaires par mail ne se faisait pas ressentir, mais j'ai pris le soin de noter les tâches réalisées afin de pouvoir compléter les comptes-rendus présents à la fin de ce rapport.

2 Organisation durant la partie recherche

Pour être sûr de ne rien oublier, je me suis servi de l'application web **Trello** en tant que simple aide mémoire. Celle-ci permet de poser des "post-it" virtuels afin de s'organiser dans ses tâches et de pouvoir prendre des notes. La quantité de tâches parallèles diminuant au fur et à mesure de l'avancement dans le projet, j'ai fini par délaisser son utilisation (n'en éprouvant plus le besoin).

Figure 1 – *Logo de l'application Trello*

3 Outil de versioning

Dans le cadre du développement de ce projet, un outil de versioning a été utilisé afin de s'assurer de pouvoir rétablir tout changement néfaste au code et de garder une trace des modifications effectuées.

L'outil utilisé est **Git**, un SCM (Source Control Manager) très répandu dans la communauté Linux et plutôt simple d'utilisation. Le code était régulièrement déposé sur deux dépôts privés : un pour le présent rapport de projet, et un second pour le code source de l'outil développé.

Figure 2 – *Logo de l'application Git*

4 Convention de documentation utilisée

Pour maximiser la réutilisabilité du projet (ce qui est clairement l'objectif visé dans un projet mettant en place une architecture générique comme celui-ci), il est essentiel d'utiliser une convention de documentation bien établie et de bien documenter les différentes classes réalisées.

A ces fins, la convention **Doxygen** a été employée afin de pouvoir générer automatiquement la documentation avec l'outil du même nom.

Figure 3 – *Logo de l'application Doxygen*

Celle-ci repose, un peu à la manière de la convention *Javadoc*, sur l'utilisation de mots-clés particuliers (*brief*, *param*, *return*...) pour détailler le fonctionnement des fonctions, classes et modules définis directement dans le code.

5 Mise en place d'une assurance qualité

La qualité du code réalisé a été assurée au moyen de **tests unitaires** automatisés, ainsi que de **tests de performances** réalisés manuellement sur les différents composants de l'application de façon individuelle. Le cahier de tests du projet est détaillé plus loin dans ce rapport.

Nous allons désormais voir en détail la mise en œuvre du projet.

6 Mise en œuvre

1 Fonctionnalités implémentées

Comme il en a été amplement question dans la première partie de ce rapport, l'objectif de ce projet en terme de développement était de créer la base d'une architecture souple et générique qui permettrait d'estimer la qualité d'une vidéo en fonction d'une base d'apprentissage fournie. Cette base se veut modulaire et pourrait ainsi par la suite être étendue dans le cadre d'autres projets ou recherches.

L'implémentation réalisée comprend donc cette architecture qui s'articule autour d'une classe principale **VideoQualityEstimator**. Celle-ci utilise trois composants de types différents pour effectuer respectivement les mesures, le pooling et la classification. En créant un objet de type VideoQualityEstimator, on lui passe donc le chemin vers la vidéo à étudier, ainsi que trois références vers des objets de type **MeasuringComponent**, **PoolingComponent** et **MappingToQualityComponent** afin de définir quels types de mesures, de pooling et de classification vont être effectués.

Cette classe centrale gère ensuite tout l'assemblage des "briques" entre elles : elle passe la vidéo au composant de mesure, en sauvegarde la sortie pour la passer au composant de pooling, pour ensuite enchaîner et envoyer le tout au composant de classification. Cette partie "d'assemblage" existe avec son lot de paramètres (notamment au niveau de l'agrégation des mesures en vue du pooling : pooling spatial ou temporel...), tous configurables depuis l'extérieur de la classe au moyen de méthodes *setters* pratiques.

Quelques composants ont été implémentés afin de tester l'architecture et de pouvoir servir d'exemple pour la suite :

- **MeasuringGradientX**, un composant de mesure calculant le gradient horizontal pour une frame de la vidéo au moyen de filtres de Sobel.
- **MeasuringOpticalFlow**, un composant de mesure calculant le flot optique dense pour un enchaînement de deux frames successives par l'algorithme de Färneback.
- **PoolingMax**, un composant de pooling effectuant un maximum basique pour chaque bloc de données

Ces composants sont combinables de façon libre, et il est possible d'imaginer une pléthore d'autres qui pourraient être implémentés (dont toutes les méthodes de mesures, de pooling et de classification citées dans la partie "Recherche" de ce rapport).

Pour des raisons de délais, on remarque qu'aucun composant de classification ne fait partie de cette liste. Cet imprévu est justifié plus loin dans ce rapport, et malgré cela, le composant générique pour la classification est bel et bien défini et intégré à l'ossature du VideoQualityEstimator.

2 Fonctionnement des types génériques de la bibliothèque OpenCV

Le système générique créé étant une sorte de "chaîne d'assemblage" avec chaque module ayant ses propres entrées et sorties, il a fallu s'assurer de la généricité des entrées et sorties de chacun de ces modules afin de pouvoir tous les combiner sans encombres. Il s'avère que d'un point de vue purement programmatique, la bibliothèque OpenCV fournit des types génériques pour manipuler les données, comme par exemple la classe **cv : :Mat** pour gérer une matrice.

Cette généricité entraîne bien évidemment d'autres complications, puisque ces matrices peuvent contenir n'importe quel type de données. OpenCV gère ce problème en permettant de récupérer un "code" indiquant le type des données présentes dans la matrice, via l'accesseur **Mat : :type()**. On peut ensuite récupérer les données en elles-mêmes via la méthode **Mat : :at<type>(coordonnees)**, où "type" doit être le type correspondant au code renvoyé par l'accesseur de type.

Par exemple, si l'accesseur de type renvoie la constante "CV_ 32FC1", cela veut dire que la matrice contient des **floats** (F) mesurant **32 bits** (32) et à une seule composante (C1). On doit donc utiliser le type "float" en appelant la méthode **Mat : :at**.

Il s'avère que les composants de pooling doivent récupérer les valeurs des matrices pour pouvoir effectuer les différentes logiques de pooling proposées. La méthode adoptée pour garder l'architecture générique a donc été d'utiliser un mécanisme de **templates** sur les composants de Pooling, servant à préciser le type des mesures qui doivent être poolées.
Il suffit alors de documenter les différents composants de mesures pour préciser quels types de données ils génèrent en sortie, afin que le type soit donné au composant de pooling lors de sa création.

Comprendre ces mécanismes (qui ne sont pas documentés de façon claire et accessible) et adapter la conception initialement réalisée à ceux-ci a pris un temps non négligeable au démarrage du projet, causant un léger retard qui s'est répercuté sur l'ensemble du planning.

3 Estimation de la complexité et de la fiabilité

Dans le cadre de ce projet, il n'est pas possible d'estimer une quelconque complexité globale. Il serait cependant tout à fait possible d'estimer la complexité de l'algorithme employé par chacun des composants implémentés dans le cadre de l'architecture dessinée pour ce projet, mais ceux implémentés jusqu'alors utilisent pour la plupart des fonctions d'OpenCV qui ont des complexités connues, établies et documentées.
Aucune estimation de complexité n'a donc été réalisée, car elle n'aurait pas réellement de sens dans notre cas.

De la même manière, notre programme est un programme procédural à utilisation ponctuelle. Il n'a pas pour vocation à tourner en continu, et n'est pas soumis aux aléas que pourrait subir, par exemple, une application serveur. Ici, l'entrée utilisateur est très limitée, et il n'y a pas de raison que la procédure fonctionne une fois et plante une autre fois.
On peut alors dire que, dans la mesure où le programme a été testé dans sa globalité, celui-ci est "très fiable", mais encore une fois, cela n'a pas de réel sens ici.

7 Campagne de tests

1 Cahier de tests fonctionnels

Table 1 – *Tableau des tests fonctionnels réalisés*

ID	Fonctionnalité	Conditions de test	Résultats
Classe `Video`			
V1	Constructeur	Conditions normales	Créé l'objet et l'initialise
V2	Constructeur	Chemin vers la vidéo invalide	Message d'erreur
V3	getFrame()	Conditions normales	Renvoie la frame demandée
V4	getFrame()	ID de frame invalide	Message d'erreur
Classe `VideoQualityEstimator`			
VQE1	Constructeur	Conditions normales	Créé l'objet et l'initialise
VQE2	setPoolingOptions()	Conditions normales	Modifie les attributs
VQE3	setPoolingOptions()	blockSize == 0	Message d'erreur
VQE4	setPoolingOptions()	blockSize > largeur de la vidéo	Message d'erreur
VQE5	setPoolingOptions()	blockSize > hauteur de la vidéo	Message d'erreur
VQE6	setPoolingOptions()	framesPerPooling == 0	Message d'erreur
VQE7	estimateQuality()	Pooling spatial	Réalise mesures et pooling
VQE8	estimateQuality()	Pooling temporel	Réalise mesures et pooling
VQE9	estimateQuality()	Pooling spatio-temporel	Réalise mesures et pooling

Table 2 – *Tableau des tests fonctionnels réalisés (suite)*

ID	Fonctionnalité	Conditions de test	Résultats
Classe `MeasuringGradientX`			
MGX1	measure()	Conditions normales	Renvoie le gradient horizontal de la frame
Classe `MeasuringOpticalFlow`			
MOF1	measure()	Conditions normales	Renvoie le flot optique des frames i et i+1
Classe `PoolingMax`			
PM1	pool()	Conditions normales	Pooling selon le maximum
Classe `MappingKPPV`			
MK1	mapToQuality()	Conditions normales	Non implémenté

Le tableau ci-dessus répertorie l'ensemble des tests fonctionnels réalisés sur les différentes classes du programme, ainsi que leur résultat. Dans le but de fournir l'architecture la plus épurée et la plus libre de dépendances possible, ces tests n'ont pas été automatisés au moyen d'une bibliothèque externe (telle que **CppUnit**), et ont donc été réalisés manuellement (la faible taille de l'architecture a rendu cela possible).

2 Tests de performance

Des tests de performance ont pu être réalisés sur la chaîne "Mesure et Pooling" (la classification ne pouvant pas y être ajoutée puisque non implémentée à l'heure actuelle), sur un ordinateur possédant 8 Go de RAM et avec un processeur Intel Core i5-3570K cadencé à 3.4GHz.

Comme il est possible de s'y attendre, les performances varient selon les composants utilisés dans le processus d'estimation, et selon les paramètres donnés à la classe `VideoQualityEstimator`.

Afin de proposer un support de test stable et comparable, les composants utilisés ont été fixés : nous utilisons ainsi la **mesure de gradient horizontal** et un **pooling spatial par maximum** et nous faisons varier les deux paramètres majeurs qui sont les suivants :

— **La taille des blocs de pooling** : plus celle-ci est élevée, moins il y a de blocs, donc plus le traitement est rapide. Une taille de 1 équivaut au traitement de l'ensemble des pixels individuellement (donc pas de composante spatiale au pooling), tandis qu'une taille supérieure définit la taille en pixels du "curseur" qui va être utilisé pour effectuer le pooling.

— **L'intervalle de frames** : il s'agit du nombre de frames ignorées entre deux itérations de mesure / pooling. Ainsi, si l'intervalle est de 100, la frame # 0 sera mesurée puis poolée, puis la frame # 101... Agrandir cet intervalle réduit le nombre global d'itérations, et réduit donc le temps global d'exécution.

Ces tests ont été réalisés sur une vidéo de 1583 frames ayant une résolution de 768×576.

2.1 Variation de la taille des blocs de pooling

On fixe l'intervalle de frames à 10, ce qui nous donne un processus de mesures et pooling en 143 itérations. En faisant varier la taille des blocs de pooling, on observe un changement considérable du

temps d'éxecution :

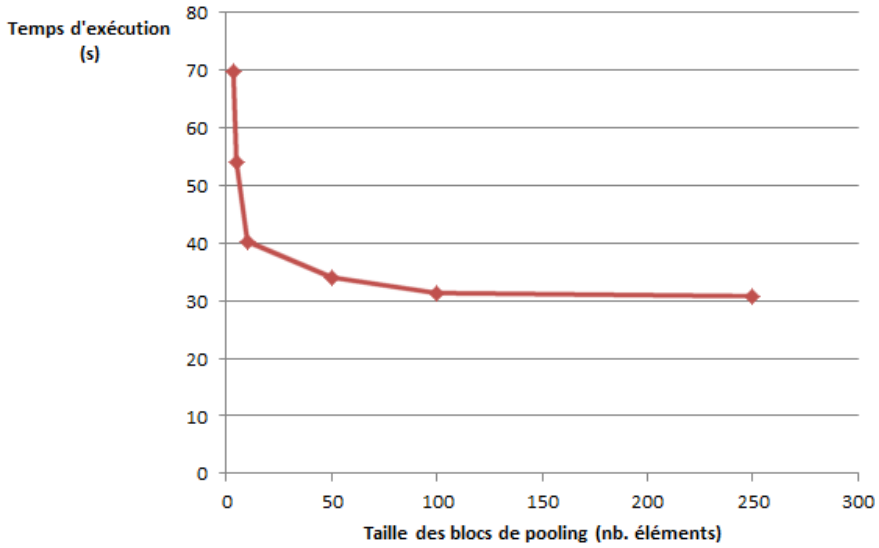


Figure 1 – *Courbe d'évolution du temps en fonction de la taille des blocs de pooling*

On observe une augmentation exponentielle du temps d'exécution en réduisant la taille des blocs de pooling. L'idéal est donc de prendre une taille de blocs de pooling autour de 20 éléments, afin d'avoir une réduction drastique du temps de calcul sans trop perdre du point de vue de la précision des mesures.

2.2 Variation de l'intervalle de frames

On fixe la taille des blocs à 20 et on fait varier l'intervalle de frames (et donc le nombre global d'itérations) :

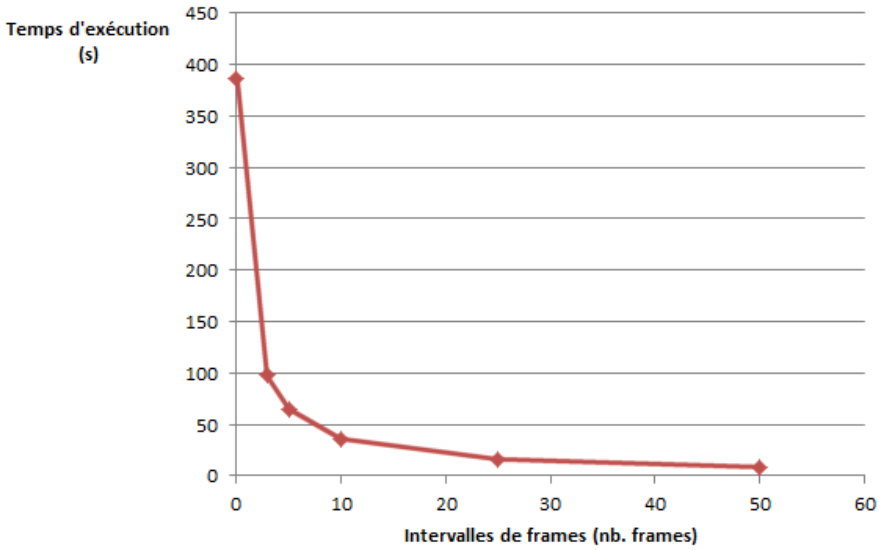


Figure 2 – *Courbe d'évolution du temps en fonction de la taille de l'intervalle de frames*

On observe encore une fois un motif exponentiel où, en diminuant la taille de l'intervalle entre deux itérations, on augmente de manière exponentielle le temps de traitement (ce qui était prévisible).

2.3 Synthèse et remarques

Dans leur globalité, ces tests démontrent que les temps d'exécution restent raisonnables pour peu que de légers paramétrages soient réalisés en amont.

La phase de classification, une fois un composant implémenté, rajouterait un temps supplémentaire en bout de chaîne qui devrait, lui aussi, rester dans des temps raisonnables, permettant à l'outil de remplir son objectif de façon concrète.

En cas de problèmes de performance, il serait tout à fait envisageable de paralléliser le code existant, que ce soit sur CPU ou GPU, afin d'accélerer drastiquement les algorithmes des différents composants.

3 Création d'une base de vidéos de test

Bien que la fonctionnalité de base d'apprentissage n'ait pas pu être développée dans les temps, un ensemble de **3 vidéos de test** a été réalisé, représentant la même vidéo à des niveaux de qualité différents.

Cela a été réalisé en téléchargeant une vidéo quelconque sur la plateforme en ligne YouTube (en l'occurrence, une scène où des gens se déplacent vue de dessus), puis en la soumettant à un même algorithme de compression avec différents niveaux de qualité sous le logiciel d'édition vidéo libre **Virtual Dub**.

Les 3 vidéos sont les suivantes :

— `video_quality_bad.avi` : la scène en mauvaise qualité
— `video_quality_average.avi` : la scène en qualité moyenne
— `video_quality_good.avi` : la scène en bonne qualité

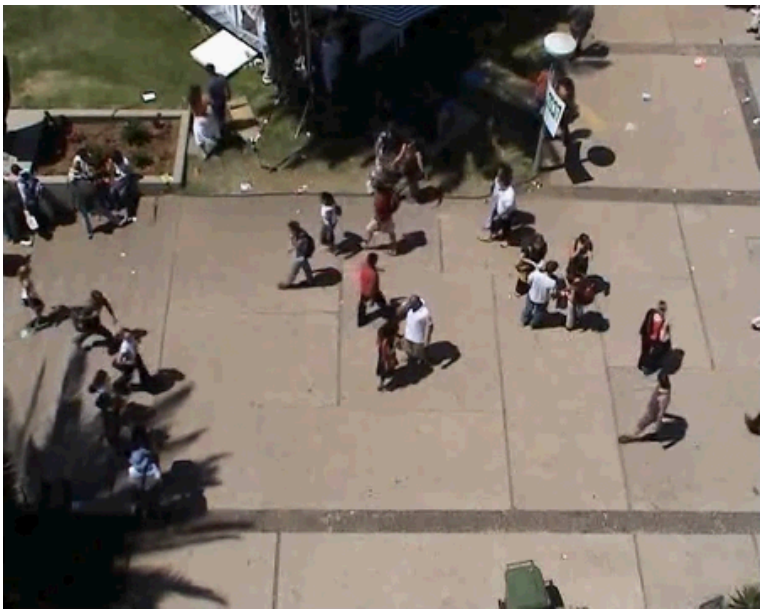


Figure 3 – *Aperçu de la scène des vidéos de test*

8 Reproductibilité

1 Installation d'OpenCV sous Visual Studio 2015

Le code tel qu'il a été rendu à la fin de ce projet comporte une solution ouvrable sous Visual Studio 2015, mais il est tout à fait possible de l'ouvrir dans des versions ultérieures.
Le projet compile pour des **architectures de type 64 bits** (x64), et nécessite l'installation d'OpenCV au préalable au sein de l'environnement de développement.

En voici le procédé, détaillé pas à pas :

1. Installez **Visual Studio 2015** ou plus récent (la version gratuite "Community Edition" fonctionne parfaitement) en le téléchargeant depuis le site de Microsoft.

2. Téléchargez la dernière version d'**OpenCV** sur le site officiel d'OpenCV (la version utilisée initialement avec le projet est la **3.1.0**). A l'heure où ce rapport est écrit, il s'agit d'un auto-extracteur exécutable.

3. Extrayez les fichiers d'OpenCV dans un dossier à part en lançant l'exécutable téléchargé.

4. Allez dans le **répertoire d'installation de Visual Studio** (si vous n'avez pas spécifié de chemin particulier à l'installation : `C:\Program Files (x86)\Microsoft Visual Studio 14.0\`) et allez dans le répertoire dédié au compilateur Visual C++ dénommé "*VC*".

5. Dans le répertoire nouvellement ouvert se trouve un dossier "***include***". A l'intérieur de celui-ci, créez un dossier "***opencv2***". Copier dans ce nouveau dossier l'intégralité des fichiers d'OpenCV présents dans le dossier `opencv/sources/include/opencv2` extraits auparavant.

6. Dans le répertoire "*VC*" se trouve un dossier "***lib***". Dans ce répertoire devrait se trouver un autre dossier appelé "***amd64***". Copiez dans ce répertoire l'ensemble des bibliothèques issues du dossier `opencv/build/x64/vc14/lib`. Ici, "vc14" fait référence à la version de Visual Studio installée.

Vous pouvez désormais lancer Visual Studio, ouvrir le fichier .sln du projet et essayer de compiler le projet. La compilation devrait se passer sans encombres.

2 Utilisation du programme

Le programme rendu étant davantage une architecture générique qu'un outil fini, il est difficile de parler en l'état de "manuel d'utilisation du programme".

Le seul point modulable à l'utilisation à l'heure actuelle (qui, a priori, restera la même pour l'outil fini) est le choix des composants utilisés. Pour cela, il suffit de modifier dans le fichier "`main.cpp`" les types de composants instanciés qui sont ensuite transmis au **VideoQualityEstimator**.

3 Enrichissement de programme existant

L'architecture développée peut être facilement enrichie en développant de nouveaux modules de mesure, de pooling ou de classification qui peuvent être "emboîtés" avec les anciens comme avec de nouveaux.

Pour cela, il suffit de créer de nouvelles classes qui héritent des classes de composants abstraits définies pour l'occasion :

- **MeasuringComponent** pour créer un module de mesures
- **PoolingComponent** pour créer un module de pooling
- **MappingToQualityComponent** pour créer un module de classification

Chacune de ces classes abstraites possède des méthodes virtuelles pures à implémenter de façon à représenter une technique de mesure, de pooling ou de classification. Il est tout à fait possible d'ajouter des attributs de paramétrage supplémentaires au module.

Les différentes classes génériques fournies dans le code sont entièrement documentées afin d'expliquer le fonctionnement des différents éléments, et plusieurs classes d'exemple ont été développées.

4 Perspectives d'amélioration

De nombreuses perspectives d'amélioration existent, au delà de l'ajout de nouveaux composants.

Par exemple, il serait intéressant de pouvoir **pooler des vecteurs**. A l'heure actuelle, le module de pooling n'est capable, de façon générique, d'effectuer la stratégie de pooling que sur des **valeurs discrètes**. Dans le cas du flot optique, par exemple, on obtient avec la méthode de Färneback un **champ de vecteurs** qui indiquent le "mouvement des pixels" entre deux frames. Il pourrait être intéressant de trouver un moyen de permettre de façon transparente de pooler des valeurs discrètes comme des objets à multiples composantes (vecteurs).

De la même manière, une **base de vidéos de test** est fournie avec le programme rendu, mais aucun test de bout en bout n'a pu être réalisé car le temps a manqué pour **créer une structure générique pour la base d'apprentissage**. En effet, les problématiques sont nombreuses : comment classifier des vidéos de taille différentes entre elles ? Comment stocker les données d'apprentissage et y accéder de manière simple et optimale, sachant que les méthodes de mesures et de pooling peuvent changer d'une exécution à une autre ? Ces questions n'ont pas trouvé de réponse dans le temps imparti, et c'est pourquoi une amélioration importante pour rendre l'outil utilisable à des fins scientifiques serait de créer ce système de base d'apprentissage et d'y injecter les vidéos fournies, pour ensuite effectuer de la classification.

Enfin, si l'architecture devient un outil d'estimation de bout en bout, il pourrait être intéressant d'introduire une gestion des **paramètres de ligne de commande** (via la variable `argv`), avec un dictionnaire

de composants qui associe des noms sous forme de chaîne de caractères ("GradientX", "OpticalFlow"...) aux composants en eux-mêmes, ce qui permettrait de ne plus avoir à modifier le code pour changer la *pipeline* d'exécution.

Conclusion

En conclusion, la partie recherche de ce projet a été intéressante et riche en découvertes et enseignements. Elle m'a permis d'acquérir la **vision d'ensemble** nécessaire pour pouvoir faire un choix parmi plusieurs solutions et aborder la phase de développement avec plus de confiance.

Cette seconde phase ne s'est pas exécutée tout à fait comme espéré, avec une accumulation de retards majoritairement liés à la mauvaise estimation du temps requis pour prendre en main OpenCV. Ceux-ci m'ont éloigné du planning prévisionnel, mais le résultat final est tout de même convaincant, puisqu'il propose une architecture générique fonctionnelle pour l'estimation de la qualité perçue d'une vidéo et implémente plusieurs algorithmes de mesures et de pooling.

La structure modulaire de l'application permettra de rajouter de nouvelles fonctionnalités de façon naturelle par la suite, ce qui assure sa pérennité sur le long terme dans le cas où elle trouverait un intérêt concret au sein de l'école ou dans la communauté scientifique.

Dans son ensemble, ce projet a été très enrichissant car il m'a confronté à la gestion de projet et à ses aléas, à des sujets de recherche pointus, ainsi qu'à la conception et au développement d'un outil générique qui se veut extensible par nature.

Comptes rendus hebdomadaires

Compte rendu n°1 du 17/09/2015

- Découverte des notions fondamentales
- Démarrage des recherches de documents en rapport avec le sujet
- Première approche des modèles d'estimation paramétriques

Compte rendu n°2 du 24/09/2015

- Étude d'un framework générique d'analyse de qualité vidéo qui sert de base de travail pour la suite des recherches
- Listing et étude d'un grand nombre d'artéfacts générés par différents types de compressions vidéo

Compte rendu n°3 du 01/10/2015

- Début d'écriture du rapport de projet
- Approfondissement des notions liées au framework étudié la semaine précédente
- Recherche parmi les références d'informations complémentaires
- Recherche de nouveaux papiers à étudier connexes au sujet

Compte rendu n°4 du 08/10/2015

- Progression massive du rapport de projet
- Début d'étude du papier de Winkler couvrant de nombreuses notions en rapport avec le sujet, et permettant de préciser certains points déjà vus dans les papiers étudiés auparavant

Compte rendu n°5 du 15/10/2015

- Etude en profondeur du papier de Winkler, permettant de faire progresser les recherches et avancer le rapport
- Mise à jour du modèle LaTeX du rapport de projet

Compte rendu n°6 du 22/10/2015

- Recherches sur des notions connexes au framework généraliste (notamment le pooling)

— Réflexions et recherches sur l'aspect "qualité audio"
— Absence le jeudi après-midi en raison d'un entretien de stage

Compte rendu n°7 du 05/11/2015

— Semaine majoritairement dédiée à l'écriture du rapport (nombreuses reformulations, restructurations et réorganisation des parties entre elles)
— Recherches sur le Video Quality Experts Group
— Recherche en particulier sur le groupe de travail VIME.

Compte rendu n°8 du 12/11/2015

— Présentation de mon travail jusqu'ici à M. Conte avec le diaporama réalisé la semaine précédente
— Aiguillage vers de nouveaux axes de recherche
— Recherches sur des algorithmes de mesures.

Compte rendu n°9 du 19/11/2015

— Poursuite des recherches sur des algorithmes de mesures
— Recherches sur des algorithmes de pooling

Compte rendu n°10 du 26/11/2015

— Recherche sur des algorithmes de classification efficaces
— Une seule journée de PRD cette semaine à cause du forum des entreprises

Compte rendu n°11 du 3/12/2015

— Recherche de jeux de données existants (entre autres du côté du VQEG VIME) afin de pouvoir évaluer l'efficacité de l'outil qui sera produit en partie développement
— Aucune base de données publiques ne semble exister

Compte rendu n°12 du 10/12/2015

— Réalisation du planning prévisionnel pour la phase de développement
— Réalisation des diapositives pour la soutenance
— Mise à jour du modèle LaTeX pour le rapport

Compte rendu n°13 du 17/12/2015

— Réalisation du poster et finition du rapport

Compte rendu n°14 du 07/01/2016

— **DÉBUT DE LA PARTIE DÉVELOPPEMENT**
— Installation d'OpenCV sous Visual Studio (quelques encombres...)
— Initiation à OpenCV.

Compte rendu n°15 du 14/01/2016

— Développement de l'ossature de l'architecture de base
— Création de la classe Vidéo pour manipuler les vidéos

Compte rendu n°16 du 21/01/2016

— Création d'un composant de mesures : "MeasuringGradientX"
— Problèmes rencontrés avec les types de données dans les matrices OpenCV
— Recherches vis à vis du fonctionnement interne des conteneurs de données OpenCV

Compte rendu n°17 du 28/01/2016

— Ajout d'une fonctionnalité de pooling spatial : "PoolingMax"
— Réflexions sur comment rendre le pooling temporel

Compte rendu n°18 du 04/02/2016

- Création d'une classe "chef d'orchestre" : "VideoQualityEstimator"
- Intégration de paramètres à cette nouvelle classe pour gérer le pooling temporel
- Modification de "PoolingMax" pour gérer le pooling spatial et temporel

Compte rendu n°19 du 11/02/2016

- Réflexions sur la compatibilité des types de données entre mesures et pooling
- Introduction du mécanisme de template pour pallier à ce problème de compatibilité

Compte rendu n°20 du 25/02/2016

- Écriture du rapport de développement
- Documentation du code existant

Compte rendu n°21 du 03/03/2016

- Ajout d'un composant de mesures du flot optique "MeasuringOpticalFlow"
- Rédaction du cahier de tests

Compte rendu n°22 du 10/03/2016

- Réalisation des tests
- Consignation des résultats
- Création d'une base de vidéos de tests avec Virtual Dub

Compte rendu n°23 du 17/03/2016

- Dernières améliorations au code
- Finition du rapport
- Paquetage des livrables

Webographie

[WWW1] *Devblog Nvidia sur les concepts fondamentaux du "Machine Learning"*. URL : http://devblogs.nvidia.com/parallelforall/deep-learning-nutshell-core-concepts/#pooling.

[WWW2] *"Gradient" sur Wikipedia*. URL : https://en.wikipedia.org/wiki/Gradient.

[WWW3] *"Optical Flow" sur le site d'OpenCV*. URL : http://docs.opencv.org/master/d7/d8b/tutorial_py_lucas_kanade.html.

[WWW4] *"Optical Flow" sur Wikipedia*. URL : https://en.wikipedia.org/wiki/Optical_flow.

[WWW5] *Site officiel d'OpenCV*. URL : http://opencv.org/.

[WWW6] *Site officiel du VQEG*. URL : http://www.its.bldrdoc.gov/vqeg/vqeg-home.aspx.

Bibliographie

[1] Sheila S. Hemami et Amy R. Reibman. « No-reference image and video quality estimation : Applications and human-motivated design ». In : *Signal Processing : Image Communication* 25 (août 2010), p. 469–481.

[2] Jose Joskowicz et J. Carlos Lopez Ardao. « A General Parametric Model for Perceptual Video Quality Estimation ». In : *IEEE Internation Workshop Technical Committee on Communications Quality and Reliability* (2010).

[3] Bruce D. Lucas et Takeo Kanade. « An Iterative Image Registration Technique with an Application to Stereo Vision ». In : *Proceedings of Imaging Understanding Workshop* (1981), p. 121–130.

[4] Zhou Wang et Xinli Shang. « Spatial Pooling Strategies for Perceptual Image Quality Assessment ». In : *IEEE International Conference : Image Processing, Atlanta, GA* (oct. 2006).

[5] Stefan Winkler et P. Mohandas. « The Evolution of Video Quality Measurement : From PSNR to Hybrid Metrics ». In : *Broadcasting, IEEE Transactions* 54 (août 2008), p. 660–668.

Estimation de la qualité perçue d'une vidéo :

Anthony Demarcy

Encadrement : Donatello Conte

Objectifs

Créer un **programme** qui donne une note évaluant la **qualité des images d'une vidéo**. Cette note doit se rapprocher au mieux de la note qu'un humain aurait donné après l'avoir regardée. On ne juge pas le contenu de la vidéo, mais bien l'absence de défauts dans les images. Réaliser ce genre de tests avec de vraies personnes serait très coûteux en temps et en argent.

Une machine peut-elle voir comme l'être humain ?

Utilité concrète

De nombreux services de diffusion de vidéo (par exemple, de streaming sur Internet) doivent **s'assurer de leur qualité** en permanence et en temps réel. Pour cela, il faut une méthode très rapide et peu coûteuse : un outil automatisé. Bien d'autres utilités existent...

*Un défaut évident : le **blocking***

Technique employée

On effectue dans un premier temps des mesures sur les pixels des images, qu'on regroupe ensuite en "paquets". On compare finalement ces paquets à ceux que l'on a repéré dans les vidéos étudiées avant celle-ci pour juger de la qualité de la vidéo : il s'agit de l'**auto-apprentissage**.

*Un défaut plus subtil : le **mosquito noise***

Ecole Polytechnique de l'Université François Rabelais de Tours
Département Informatique
64 avenue Jean Portalis, 37200 Tours, France
www.polytech.univ-tours.fr

POLYTECH TOURS
Informatique

Estimation de la qualité perçue d'une vidéo

Résumé

Ce document présente la synthèse d'un état de l'art sur l'estimation de qualité vidéo perçue, en restituant les notions nécessaires à sa compréhension. Elle présente également les enjeux et la méthodologie derrière un projet visant à créer un outil d'estimation en utilisant les connaissance acquises dans cet état de l'art.

Mots-clés

traitement du signal, traitement de l'image, qualité vidéo, auto-apprentissage

Abstract

This document presents the synthesis of a state of the art on perceptual video quality estimation, giving the background knowledge required to understand it. It also exposes the goals and methodology behind a project aiming to create a quality estimation tool, using the previously acquired knowledge.

Keywords

signal processing, image processing, video quality, machine learning

Tuteurs académiques
Donatello Conte

Étudiants
Anthony Demarcy (DI5)