École Polytechnique de l'Université de Tours
64, Avenue Jean Portalis
37200 TOURS, FRANCE
Tél. +33 (0)2 47 36 14 14
www.polytech.univ-tours.fr

Département Informatique
3e année
2011 - 2012

Projet Système d'Exploitation

Interaction utilisateur PC-webcam

Encadrant
Mathieu DELALANDRE
mathieu.delalandre@univ-tours.fr

Université François-Rabelais, Tours

Étudiants
Clément TESSIER
clement.tessier@etu.univ-tours.fr
Raphaël ROGER
raphael.roger@etu.univ-tours.fr

DI3 2011 - 2012

Version du 9 juin 2012

Table des matières

Table des figures

Introduction

Dans le cadre des projets de 3ème année, une liste de projets ayant pour thème la matière « Système d'exploitation »nous a été proposée. Dans cette liste de nombreux projets intéressants ont retenu notre attention et plus particulièrement ce projet : Interaction utilisateur PC webcam. Les principales raisons de ce choix sont en premier notre intérêt pour le cours de « Traitement d'images ». Ensuite, pour notre curiosité commune à la réalité augmentée : découvrir plus en profondeur le traitement de l'image en passant par son acquisition nous permettrait de nous conforter ou non dans ce domaine, dans l'éventualité du PFE.

L'objectif principal de ce projet, fixé par notre encadrant, était de récupérer un flux d'informations en provenance de la webcam. Nous avons donc pensé à utiliser une librairie telle que OpenCv, librairie la plus utilisée dans la recherche scientifique portant sur le traitement d'images. Suite à une mauvaise compréhension de notre part, nous avons compris que l'objectif principal était en fait de récupérer les flux d'informations d'une webcam en passant uniquement par les drivers, sans librairies tiers. Le projet étant avant tout porté sur le Système d'exploitation et non la conception d'algorithmes avec OpenCv.

Système d'Exploitation

2.1 Fonctionnement d'une webcam

2.1.1 Intérets du traitement d'images

Notre projet porte donc en partie sur un périphérique de type webcam. De nos jours tous les ordinateurs ou presque ont une webcam intégrée. Les commerçants vendent ces périphériques pour des sommes de plus en plus minimes (webcam de petite qualité pour environ 10 euros) et même nos téléphones portables s'en équipent ! Le traitement d'images est donc devenu un point primordial de cette évolution. Que ce soit la détection de sourire sur un appareil photo, la détection de personnes marchant sur la route dans le noir : système installé par exemple dans les voitures de la marque Audi sous le label Audi Lab, vision nocturne avec détection de piétons. Le traitement d'images et les petites caméras sont partout autour de nous et ont de nombreuses fonctions : sécurité(détection de piétons ou conducteur en train de s'endormir), loisir(déformations de corps/visage, ajout d'objet sur l'image dans le cas de certains jeux vidéos), informatif(réalité augmentée, ajout d'informations sur des bâtiments au travers de lunettes équipées d'une caméra et d'écrans). En résumé, ce domaine est devenu incontournable.

Une webcam, autant dans sa partie matérielle que logicielle, est composée de plusieurs éléments. Ils permettent l'acquisition des photos et leur transfert vers une application ou un autre périphérique.

2.1.2 Composition d'une webcam

D'un point de vue uniquement matériel, une webcam est principalement composée d'un système optique et d'un capteur CCD.

Le système optique

Le système optique a pour objectif de concentrer la lumière sur le capteur CCD mais pas uniquement. Il est en vrai composé d'une lentille placée sur un support amovible, ce qui permet d'approcher ou d'éloigner celle-ci du capteur. C'est avec ce procédé que la mise au point sera réalisée. Ce support peut-être une simple vis ou un système plus complexe permettant un auto-focus.

Le capteur CCD

Le capteur CCD (Charge-Coupled Device, ou dispositif à transfert de charge) est un capteur composé d'un grand nombre de photosites réparti en colonnes et en lignes. Lorsqu'un photon (particule de lumière) traversera un photosite, celui-ci se chargera négativement. L'image sera donc stockée sous forme de charges négatives dans le capteur CCD (dans les photosites).

Pour récupérer une image, l'appareil utilisera le front d'horloge pour étudier chaque photosite et envoyer l'information vers un convertisseur analogique (CAN). Après conversion l'image sera ré-assemblée.

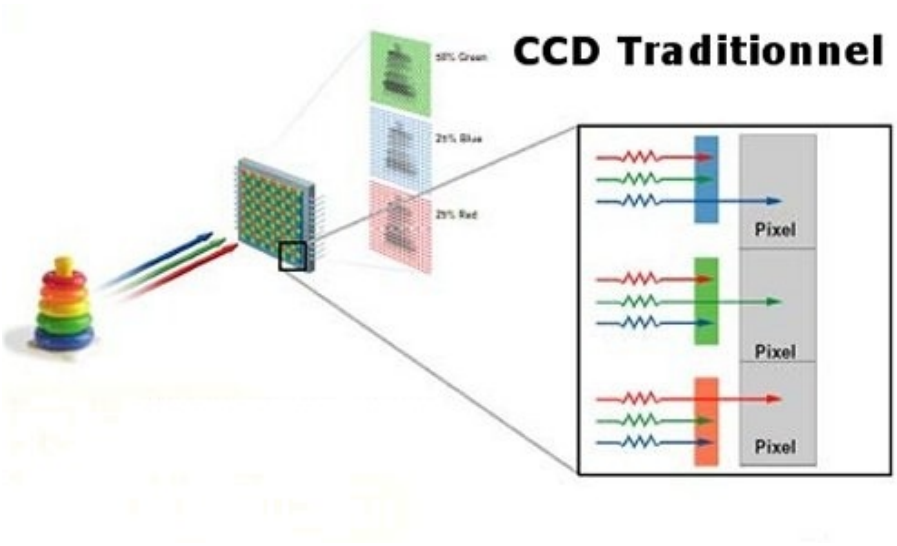

FIGURE 2.1 – Principe de capture des couleurs par un capteur CCD

2.2 Fonctionnement des buffers

Dans le cas de notre webcam, ce périphérique a son propre buffer. Il le remplit à chaque capture en écrasant les données précédentes s'il n'y a plus de mémoire disponible. Pour récupérer le flux d'une webcam, donc le contenu de ce buffer, il existe principalement 3 solutions que nous détaillerons.

2.2.1 Copie de la mémoire directement avec le processeur

Cette façon de faire est la plus mauvaise des trois. Elle a pour but de demander au processeur de copier la mémoire contenue dans le buffer de la webcam (par l'intermédiaire du contrôleur du périphérique) vers la mémoire vive de notre ordinateur (un buffer dans notre application logicielle par exemple). Le principal inconvénient est que le processeur sera inutilement solicité à ne faire que ça. Il faudra vérifier en continu si le buffer a changé, si oui recopier ces informations vers la mémoire vive de l'ordinateur, et continuer cela en boucle. Cette méthode est à proscrir.

2.2.2 Utilisation du DMA (Direct Memory Access)

Le principe du DMA est très intéressant dans notre cas. Il faut bien se rendre compte que le CPU peut faire des requêtes à un contrôleur E/S, mais seulement d'un byte par unité de temps, un important gâchis de temps de calcul. Le DMA ne peut fonctionner uniquement si un contrôleur DMA existe dans le hardware, ce qui est le cas pour beaucoup de systèmes. Il peut aussi être intégré au contrôleur du périphérique. Peu importe son emplacement, le DMA a un accès au bus de données et contient plusieurs registres susceptibles d'être écrits ou lus par le CPU. Ces registres sont composés : d'une adresse de mémoire de registre, d'un compteur et d'un (ou plusieurs) registre(s) de contrôle. Le registre de contrôle sert à spécifier le sens des transferts vers le périphérique (lecture ou écriture), l'unité de transfert (byte/time ou word/time), ainsi que le nombre de byte à transférer en une salve.

2.2.3 Fonctionnement du DMA (fly-by MODE)

Le fonctionnement du DMA est assez simple, le CPU configure le contrôleur du DMA en précisant ce qu'il doit copier (buffer d'un périphérique), à quel emplacement de la mémoire copier ces informations, et sous quelles quantités et vitesse de transfert. Le DMA une fois paramétré demande au contrôleur du

périphérique de copier les informations de son buffer vers l'adresse mémoire précisée par le CPU au paramétrage. À chaque « copie »dans la mémoire, l'adresse est incrémentée et le compteur décrémenté. Une fois le compteur à zéro, le DMA réalise une interruption pour prévenir le processeur que le transfert est terminé, ainsi le CPU peut réaliser d'autres opérations pendant le transfert.

On pourrait résumer son fonctionnement de la manière suivante :

1. Le CPU paramètre le contrôleur DMA (adresse, compteur de mots, etc)
2. le DMA réalise une requête pour transférer la mémoire
3. La mémoire est transférée selon les paramètres
4. Le DMA met à jour l'adresse et le compteur de mots (incrémentation et décrémentation)
5. étapes 2, 3 et 4 jusqu'à ce que le compteur arrive à zéro
6. le DMA réalise une interruption pour dire au CPU que le transfert est terminé

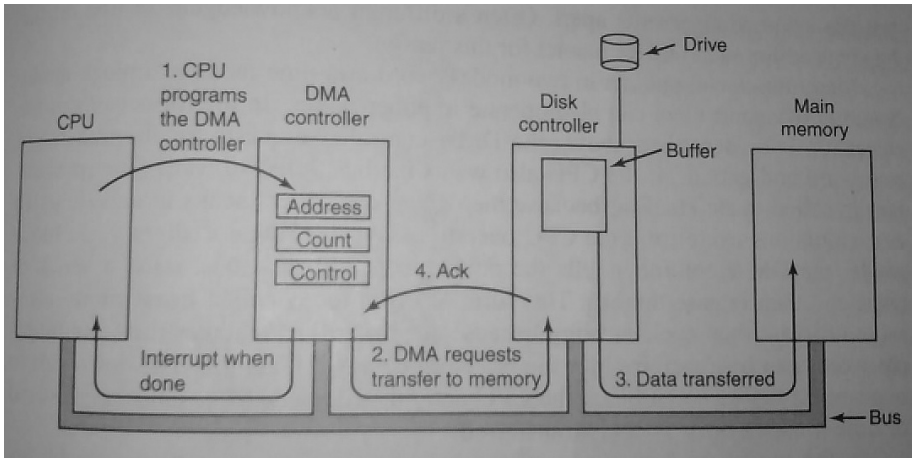

FIGURE 2.2 – Principe de fonctionnement du DMA

Malheureusement, tous les ordinateurs n'utilisent pas le DMA. La raison est que le CPU principal est souvent de loin plus rapide que le contrôleur DMA et peut ainsi effectuer le travail bien plus rapidement (à condition que la vitesse des périphériques puisse tenir le rythme). De plus, la mise en place d'un contrôleur DMA sur des technologies de type embarquées (webcam par exemple) augmente les frais. Il est donc courant de ne pas mettre de contrôleur DMA pour économiser de l'argent.

2.2.4 Fonctionnement du MMAP (Mémory Mapping)

Dans le cadre de notre projet nous n'avons pas mis en place un système DMA par contrainte matérielle. Mais le système mis en place reste aussi performant. Selon les concepteurs de cette méthode, elle surpasserait le DMA. N'ayant pas pu mettre en place un système DMA, nous reportons uniquement leurs conclusions et nous ne pouvons pas certifier cette information.

Le mémory mapping a pour principe de ne pas copier la mémoire en provenance du contrôleur du périphérique vers la mémoire vive de notre ordinateur. Mais uniquement d'échanger des pointeurs entre les « deux »buffers. Seulement, et c'est certainement sur cet aspect que le MMAP permet peut-être de meilleurs résultats que le DMA, le Memory Mapping permet de relier plusieurs buffers (de l'application logicielle) au buffer du contrôleur de périphérique (dans notre projet nous en utiliserons 4 en moyenne). Ainsi, quand le CPU traite un buffer de l'application logicielle, le MMAP se charge de mettre à jour l'un des autres buffers.

Comme pour le DMA, le processeur se charge uniquement de paramétrer le MMAP, on augmente ainsi largement le « temps libre »de calcul CPU. Son initialisation d'un point de vue programmation sera étudiée plus loin.

2.3 Codage de l'information

Lorsque nous avons réussi à initialiser une « connexion »entre le périphérique et notre logiciel, un premier problème est apparu : une absence totale de documentation sur l'encodage des données dans le buffer du contrôleur. En nous remémorant nos cours sur le langage assembleur et la mémoire de l'ordinateur a son plus bas niveau, nous avons suivi différentes pistes. La première fut de lire ce flux d'informations comme un flux binaire, ce qui nous paraissait logique et qui se révéla payant. Le problème étant de savoir sur combien d'octets était codée l'information ? Les principaux types au plus bas niveau sont le byte, le word et le double word. Les images étant codées« en général »sur plusieurs canaux (RGB, lab, etc), nous avons tenté de lire ce flux octet par octet (composantes R G B sur 8 bits par exemple). Une redondance de données est rapidement apparue. Pour vérifier que nous étudions bien les résultats d'une capture nous avons essayé de réaliser une capture dans le noir, et une avec une importante source de lumière dirigée vers le capteur. Les résultats étaient encourageants.

Dans le cas de la capture dans le noir (réciproquement avec beaucoup de lumière), un octet sur deux était en moyenne à 0 (réciproquement à 255). Notre surprise fut de trouver des nombres se répétant de manière plus ou moins aléatoire, par exemple l'une des premières séries :
254 34 255 89 255 35 253 91 255 33 252 88
Les redondances entre 34, 35 et 33 ainsi que 89, 91 et 88 ne pouvaient pas être le fruit du hasard. De plus la caméra ne supportait pas les captures au format RGB. Nous avons donc cherché quel format choisir ? Cette série est donc issue d'une capture initialisée au format YUV.

2.4 Un modèle de représentation de la couleur : le format YUV (YCbCr)

Le format YUV nous a posé de nombreux soucis car il comporte en réalité de nombreux dérivés (yuv 422, yuv 444, yuv 411, etc), soit environ une quinzaine. En parcourant une ancienne documentation classée comme obsolète, nous avons compris que notre format ne pouvait être soit le format YUV 422, soit le format YUV 420. D'après nos précédentes études, le format YUV 422 était le plus probant.

Il faut savoir que le format YUV est le format de base de nombreuses webcams, ainsi que d'une grande majorité des appareils de capture video. Ce format est né aux environs de 1959, rattaché aux standards de codage PAL SECAM. Son principal intérêt est que le format YUV est composé d'une luminance Y, d'une chrominance en rouge Cr et d'une chrominance en bleu Cb. Ainsi, la composante Y compose l'image en niveaux de gris, la chrominance bleu et rouge se mélangent pour recréer les couleurs. Ce format permettra le passage de la télé en noir et blanc à la télé en couleurs sans aucun problème. Les anciens téléviseurs ne connecteront que le câble correspondant au canal Y, pour garder une image en noir et blanc, tandis que les téléviseurs couleurs seront raccordés par les trois câbles correspondant à la luminance, la chrominance rouge et la chrominance bleu. Cette information sera cruciale pour notre projet car la conversion YUV en niveaux de gris sera finalement aisée.

2.4.1 Les différents formats YUV

Un format YUV est caractérisé par les trois nombres qui le suivent : YUV 422, YUV 420, YUV 411, etc. Ces formats sont caractérisés par le nombre d'octet pour la luminance et pour les chrominances.

YUV422

L'encodage YUV422 signifie 2 octets de luminance, 1 octet de chrominance rouge, 1 octet de chrominance bleu, le tout pour 2 pixels. Cela veut dire que chaque chrominance affectera deux pixels collés tandis que la luminance n'affectera qu'un seul pixel à chaque fois.
Cela donne l'encodage suivant :

Y0 Cb0 Y1 Cr0 Y2 Cb1 Y3 Cr1

Le pixel 1 aura pour composition : Y0, Cb0, Cr0
Le pixel 2 aura pour composition : Y1, Cb0, Cr0
Le pixel 3 aura pour composition : Y2, Cb1, Cr1
Le pixel 4 aura pour composition : Y3, Cb1, Cr1

YUV420

L'encodage YUV420 signifie 4 octets de luminance, 1 octet de chrominance rouge, 1 octet de chrominance bleu, le tout pour 4 pixels. Cela veut dire que chaque chrominance affectera un groupe (carré) de quatre pixels tandis que la luminance affectera individuellement chaque pixel. La seconde particularité de cet encodage est que toutes les informations sont regroupées : il y aura d'abord toutes les luminances, suivies des chrominances bleu et finalement les chrominances rouge. Cet encodage faciliterait la compression de données.
Soit l'encodage suivant :

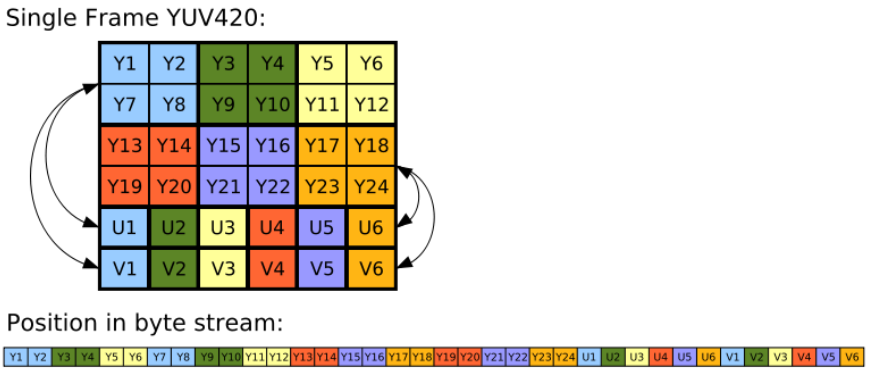

FIGURE 2.3 – Encodage YUV420

YUVXYZ

Il existe de nombreux autres formats mais il n'y a que peu d'intérêts à les recenser dans ce rapport. Pour plus d'informations, il faut se reporter à la bibliographie en fin de rapport.

2.5 Le format d'image RAW

Ce n'est pas un format standard, mais plutôt la désignation d'un certain type de fichier créé par des dispositifs tels que les appareils photo numériques ou les scanneurs et caractérisé par le fait de n'avoir subi que peu de traitement informatique. Ce fut notre première méthode de sortie. Ce format est utilisé par de nombreuses marques, notamment : Panasonic, Sony et Canon. Ce format est un format brut, il ne

contient ni en-tête pour spécifier comment sont codées les informations, ni indications pour reconstituer l'image. Le problème étant que chaque constructeur a son propre encodage pour le fichier RAW. Certaines « règles »existent, mais notre fichier de format raw n'était d'aucune marque précise. Après plusieurs recherches, il s'est avéré que l'encodage était le plus simple possible : il correspondait« simplement »à l'enregistrement des octets les uns après les autres sous l'encodage YUV, à savoir le contenu de notre fichier était : Y0 U0 Y1 V0 Y2 U1 Y3 V1 Y4 U2 Y5 V2 ... Les logiciels ne reconnaissant pas notre format de fichier, l'aperçu de l'image fut à la fois encourageant et démotivant. Notre « premier résultat visuel »fut les têtes de notre binôme, coupées en 3, avec des diagonales fluorescentes et des déformations de couleur importantes. Un effet de zoom intense était aussi présent. Ce format de fichier nous a permis de diviser notre projet, sur la partie acquisition d'images, en deux parties. En premier lieu, la création de fichiers RAW issus de la webcam, et ensuite une fonction permettant de recoder ce fichier RAW en une matrice correspondant à une image en niveaux de gris (sur 8bits).

Ce format ainsi que la création de fichier RAW sera abandonné une fois la fonction de conversion créée et optimisée pour traiter les flux vidéo en temps continu et non en passant par des fichiers.

2.6 Les conversions à partir du format YUV

Les images dont la couleur a été encodée au format YUV ont la possibilité d'être converties vers des formats couleur plus conventionnels ou communs. Nous verrons dans ce rapport les deux conversions les plus intéressantes dans notre cas.

2.6.1 Conversion du format YUV au format niveaux de gris

Comme décrit précédemment, le format YUV est composé d'un paramètre nommé luminance qui contient toutes les informations nécessaires pour reformer notre capture en niveaux de gris. Il suffira de récupérer un octet sur deux et de les stocker à la suite dans une matrice (tableau deux dimensions dans notre cas) pour ainsi supprimer les chrominances rouge et bleu et ne garder que les indices de luminances.

2.6.2 Conversion du format YUV au format RGB

La conversion RGB a été réalisée dans ce projet, mais dans le cas du traitement d'images, nous avons préféré le format niveaux de gris au format RGB. Cette préférence est simplement liée au fait que les principales méthodes vues en cours de traitement d'images sont appliquées sur des images en niveaux de gris (même si l'adaptation de ces méthodes peut être réalisée pour des images au format RGB) et dans une approche d'optimisation de la fluidité.
Les conversions YUV -> RGB et RGB -> YUV se réalisent par la multiplication des composantes YUV (réciproquement RGB) par une matrice dont les coéfficients sont issus de recherches scientifiques portant sur la perception humaine des couleurs. Ils existent plusieurs matrices de conversion, toutes plus ou moins proches.
Dans le cas de notre projet nous avons utilisé les deux matrices de conversion suivantes.

$$\begin{bmatrix} Y' \\ U \\ V \end{bmatrix} = \begin{bmatrix} 0.2126 & 0.7152 & 0.0722 \\ -0.09991 & -0.33609 & 0.436 \\ 0.615 & -0.55861 & -0.05639 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

FIGURE 2.4 – Matrice de conversion du format RGB au format YUV

$$\begin{bmatrix} R \\ G \\ B \end{bmatrix} = \begin{bmatrix} 1 & 0 & 1.28033 \\ 1 & -0.21482 & -0.38059 \\ 1 & 2.12798 & 0 \end{bmatrix} \begin{bmatrix} Y' \\ U \\ V \end{bmatrix}$$

FIGURE 2.5 – Matrice de conversion du format YUV au format RGB

Développement Logiciel

3.1 Vidéo For Linux 2

Pour réaliser notre logiciel ainsi que l'acquisition de données, nous avons utilisé la bibliothèque Vidéo For Linux 2, nommée V4L2. Cette bibliothèque n'est en aucun point comparable à des bibliothèques comme OpenCv. Celle-ci ne permet aucun traitement de l'image et ne comporte d'ailleurs qu'une seule fonction : l'envoi de requête sur le périphérique. Cette « librairie »permet aux développeurs de programmer en utilisant directement les drivers du périphérique car V4L2 utilise et installe pour la webcam ses propres drivers génériques. Ainsi le périphérique devient 100% compatible avec la bibliothèque Video For Linux 2.

3.2 Fonctionnement de V4L2

Cette bibliothèque n'incorpore qu'une seule réelle fonction : une fonction qui permet uniquement d'envoyer une requête et de récupérer un statut sur son envoi. Dans cette fonction, et c'est à ce moment là que ça se complique, les arguments nécessaires à la requête sont peu nombreux : les deux premiers sont obligatoires ; fd et argument, ainsi qu'un troisième paramètre optionnel selon le cas. Le paramètre fd est le lien vers le « périphérique », et non le flux de la webcam. Concrètement, ce lien est /dev/video0 par défaut sur les système UNIX/LINUX. Le paramètre argument va renseigner le périphérique sur l'objet de la requête. Cet argument sera un flag qui correspondra donc à notre demande. Il y a un flag pour tout type de demande. Les principaux flag que nous utiliserons seront les suivants.

3.2.1 VIDIOC_QBUF

VIDIOC_QBUF, on demande au driver de placer le buffer, donné en paramètre optionnel, dans la queue d'attente du driver.

3.2.2 VIDIOC_STREAMON

VIDIOC_STREAMON (réciproquement VIDIOC_STREAMOFF), on demande au driver d'initialiser la capture, ou plus précisément de produire un flux vidéo envoyé vers le buffer du driver.

3.2.3 VIDIOC_REQBUFS

VIDIOC_REQBUFS, on demande si le périphérique est compatible avec la mise en place d'un système buffer, on lui envoie en paramètre le nombre de buffers que l'on veut mettre en place (4 dans notre cas) , le type de système buffer (V4L2_MEMORY_MMAP dans notre cas, un flag qui permet d'indiquer que le système de buffer voulu est de type Memory Mapping) et bien sûr le type d'informations stockées dans le buffer (V4L2_BUF_TYPE_VIDEO_CAPTURE dans notre cas, un flag qui signifie que l'on veut enregistrer un flux vidéo). Dans ce premier cas on demande uniquement si notre matériel est compatible avec cette requête, si c'est le cas, on relance une requête avec des paramètres sensiblement différents et on donne ainsi nos buffers logiciel qui devront être utilisés pour le MMAP.

3.2.4 VIDIOC_QUERYBUF

VIDIOC_QUERYBUF, on demande l'état du buffer dont l'adresse est placée en paramètre de notre requête (le pointeur). Les réponses possibles sont mapped, enqueued, full ou empty. Cette requête nous permet de nous renseigner rapidement et facilement sur l'état de nos différents buffer.

3.2.5 VIDIOC_CROPCAP

VIDIOC_CROPCAP, on demande les limites de cadrage (résolution) et de profondeur des couleurs (8bits par exemple) pour une capture (V4L2_BUF_TYPE_VIDEO_CAPTURE).

3.2.6 VIDIOC_S_CROP

VIDIOC_S_CROP, on exécute cette requête avec les paramètres que l'on souhaite (cadrage 640x480 : flag defrect) et quelle partie de la capture on souhaite récupérer (par défaut la totalité, mais on peut préciser que l'on souhaite uniquement récupérer un carré de 10x10 pixels, pour tester par exemple si une pièce est plongée ou non dans le noir).

3.2.7 VIDIOC_S_FMT

VIDIOC_S_FMT, on peut utiliser ce flag pour 3 requêtes : la première pour tester si le format des données que l'on souhaite est compatible avec notre webcam. La seconde pour définir ses paramètres et modifier le format des données. La troisième permet de récupérer le format des données si elles ont été correctement paramétrées précédemment. Notre format de données sera principalement composé : d'une largeur exprimée en pixel (640), d'une hauteur exprimée en pixel (480) et d'un format d'encodage des couleurs (V4L2_PIX_FMT_YUYV, flag correspondant au format YUV422) . Par défaut le format YUV422 est en 8 bits.

3.2.8 VIDIOC_QUERYCAP

VIDIOC_QUERYCAP, on demande avec ce flag les capacités matérielles, pour vérifier sa compatibilité avec les drivers V4L2. Si ce matériel ne l'est pas, nous ne pourrons pas utiliser les drivers génériques V4L2 ou bien ils ne seront pas pleinement compatibles.

3.2.9 VIDIOC_DQBUF

VIDIOC_DQBUF, ce flag permet de vérifier l'état de nos buffers comme VIDIOC_QUERYBUF, à une exception près, si un buffer est rempli, le driver le sort de la queue d'attente du MMAP et nous renvoie le numéro du buffer. Nous utilisons ce flag uniquement lorsque l'on veut étudier un buffer.

3.3 Logiciel de capture et de traitement d'images

Comme dit précédemment, notre logiciel est basé sur V4L2, une API de base intégrée au noyau linux. Cette librairie ne met à disposition du développeur qu'une unique fonction. Cette dernière nous permettra d'intérroger le périphérique via le driver, de lui commander certaines actions et de s'informer sur son état. Ce projet n'utilisant aucune librairie de traitement d'images ou de gestion de flux vidéos (comme OpenCv par exemple), tous les systèmes de connexion, de gestion de buffer ou de décodage/encodage d'images restent à notre charge. Cette partie classée comme bas-niveau (à comprendre ici qu'il n'y a aucune couche d'abstraction entre l'utilisateur et les drivers), il n'existe pas de fonction *ouvrir_camera*, *enregistrer_image_jpg* comme avec OpenCv, ici tout est à faire. On rajoutera qu'OpenCv possède déjà tous les outils de base pour réaliser les actions suivantes :

- capture simple et complexe d'un flux vidéo
- détection de contours
- détection de visages et de sourires
- détection d'objets et de formes
- déformations de base
- création de panorama
- créer une interface de réalité augmentée
- et beaucoup d'autres ...

3.3.1 Presentation de notre logiciel

Notre logiciel se divise en plusieurs fonctions qui traitent différentes tâches telles que la liaison avec la webcam ou encore la détection de contours en temps continu.

Main

La fonction main appelle dans le bon ordre les différentes fonctions de notre programme. Elle permet de :

- ouvrir une connexion au périphérique
- initialiser le périphérique et le mmap
- lancer une requête de capture
- utilisation du mmap pour récupérer les captures executées par la webcam
- récupération des buffers et enregistrement de l'image dans une matrice
- traiter les images selon différentes méthodes (Laplacian par exemple)
- finalisation de la capture
- fermeture du mmap et liberation de la mémoire
- coupure du lien entre le periphérique et l'application via les drivers

open_device

Cette fonction vérifie si le fichier « /dev/video0 »existe, s'il est répertorié comme périphérique, et si c'est le cas, on ouvre ce périphérique (comme un fopen() en langage C) et on stocke le résultat dans fd, pointeur utilisé fréquemment dans le programme.

init_device

Cette fonction vérifie si le périphérique est compatible avec les drivers V4L2 (Video for linux 2). Si c'est le cas on vérifie si la capture vidéo est supportée. Si la webcam passe cette série de test, on informe le driver que nous utiliserons la webcam pour réaliser une capture. A ce stade, le périphérique sait uniquement que l'on veut réaliser une capture, rien de plus. Si la webcam nous confirme que notre information est bien prise en compte, nous lui envoyons le format de capture souhaité : largeur de 640px, hauteur de 480px, encodage des couleurs au format YUV (YUV 422). Le périphérique nous répond s'il est capable de réaliser une telle capture, mais la capture n'est toujours pas initialisée, le driver connait à ce moment uniquement le format souhaité.

init_mmap

Cette fonction enregistre tout d'abord les paramètres que l'on souhaite dans une structure, à savoir le nombre de buffers souhaité, le type de buffers et le type de système de buffers. Dans notre cas nous souhaitons 4 buffers, des buffers de type « capture vidéo »et un système de buffer « Memory Mapping ». Le driver nous répond s'il est capable de réaliser un tel système de buffer. Si c'est le cas, nous réalions la

requête pour ce système. Le driver peut changer certains paramètres si au moment précis de notre requête il n'a plus de mémoire disponible ou qu'un notre logiciel devait réaliser une requête similaire en parallèle. Nous vérifions donc que le nombre de buffers alloué est supérieur à deux, sinon le système MMAP ne sera pas efficace. Si tous les paramètres nous correspondent, on crée une structure « buffers »qui nous permettra de gérer les 4 buffers tout au long du programme. On initialise ensuite chaque buffer contenu dans la structure buffers avec les paramètres suivant :

- type de buffer : buffer de capture vidéo
- type de système buffer : Memory Mapping
- index du buffer : numéro du buffer (de 1 à 4)

On récupère ensuite pour chaque buffer son adresse dans le driver et on les ajoute au système de Memory Mapping, car le driver peut très bien avoir plusieurs type de buffers.

start_capturing

Cette fonction ajoute nos buffers de type capture vidéo et de système MMAP dans la queue d'exploitation du Memory Mapping dans le driver. On demande ensuite au driver de démarrer le flux vidéo de type capture vidéo.

main_loop

Cette fonction vérifie dans un premier temps qu'il n'y a pas d'interruptions dans le flux vidéo. Si le flux vidéo n'envoie pas d'informations pendant 2 secondes, on considère que la connexion est rompue et on arrête le logiciel. Si le flux n'est pas rompu, on étudie ce flux dans une première fonction nommée read_frame. On affichera le nombre de frames par seconde (FPS) dans cette fonction.

read_frame

Cette fonction demande au driver de sortir un buffer de la queue d'attente du MMAP et on récupère l'adresse de ce buffer. Ce buffer contient les informations selon l'encodage des couleurs au format YUV comme énoncé dans la première partie de ce rapport. On envoie donc ce buffer vers la fonction process_image qui se chargera de décomposer le buffer pour reconstruire l'image. Une fois la fonction process_image executée, on demande au driver de rajouter l'actuel buffer dans la queue d'attente du MMAP.

process_image

Cette fonction lit le buffer du début jusqu'à la fin et inscrit les luminances dans une matrice qui représentera l'image en niveaux de gris. Une fois cette tâche terminée, elle appelle une fonction de traitement d'images qui créera une nouvelle image (une copie de celle qu'on vient de décoder) et qui lui appliquera une transformation parmis celles disponibles (detection de contours : Kirsh, Sobel, Laplacian ; flou gaussien, filtre Nagao). Après la transformation, la fonction écrit pixel par pixel l'image originale à gauche et l'image modifiée à droite pour pouvoir les comparer.

uninit_device

Cette fonction réalise surtout le dé-mappage de la mémoire. Il supprime les liens entre nos buffers et le MMAP du driver, puis libère de la mémoire nos buffers. Il libère ensuite la structure qui nous permettait de gérer les buffers.

close_device

Cette fonction réalise l'équivalent d'un fclose en langage C. Il ferme le périphérique « /dev/video0 »et stocke le résultat dans fd. Ce résultat nous permet de vérifier que la fermeture s'est bien déroulée, que notre webcam n'est plus reliée à notre logiciel et surtout qu'elle sera désormais disponible pour les autres logiciels.

fonctions de traitement d'images

Ces fonctions réalisent les opérations vues en cours de traitement d'images en appliquant des masques sauf pour Nagao qui est plus complexe.

Traitement d'Images

La partie « Traitement d'images »est une partie mineure dans ce projet étant donné que la partie la plus compliquée fut la gestion de la capture. Une fois l'image stockée sous forme de matrice, le traitement d'images est une application directe du cours. Seulement il y a tout un travail d'optimisation à réaliser en arrière-plan pour espérer dépasser les 10 images par seconde.

Pour ce projet, nous avons décidé de mettre en place différentes détections de contours pour tester dans un premier temps si notre logiciel pouvait réaliser du traitement d'images en temps continu, puis dans un second temps pour avoir une application concrète de notre cours. Nous avons ensuite décidé de tester différents masques moyenneurs (moyenneur classique, moyenneur pondéré et gaussien) ainsi que le filtre Nagao, qui est notre projet de traitement d'images.

4.1 Détection de contours

Pour toutes nos détections de contours, nous avons appliqué deux fois le masque, une fois avec une rotation, pour pouvoir détecter plus efficacement les contours. Etant donné que cette partie est explicitement basée sur le cours de traitement d'Images, nous nous contenterons de montrer ici nos résultats.

4.1.1 Sobel

FIGURE 4.1 – Détection de contours par Sobel

4.1.2 Kirsh

4.1.3 Laplacian

4.2 Flou gaussien

Le flou gaussien mis en place dans notre logiciel est un masque de 3x3 et le flou reste donc discret.

FIGURE 4.2 – Détection de contours par Kirsh

FIGURE 4.3 – Détection de contours par Laplacian

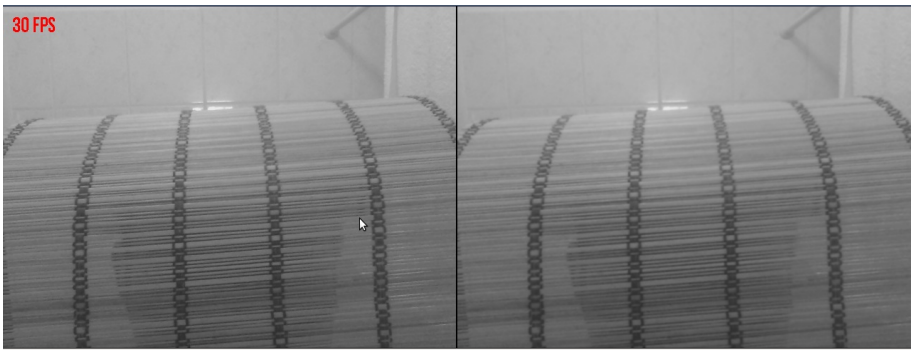

FIGURE 4.4 – Flou gaussien

Améliorations pensées et si possible réalisées

5.1 Système d'Exploitation

5.1.1 Mise en place d'un système de buffer

Notre première idée était de mettre en place un système DMA, présenté par notre encadrant lors de notre première réunion. Après avoir étudié les différentes documentations trouvées sur l'internet et dans le livre « Operating System »de Andrew Tanenbaum, avoir assimilé ce système, nous avons recherché les différentes possibilités liés aux webcams. Première déconvenue dans ce projet : les webcams ne gèrent pas en général le DMA, et encore moins souvent celles sous le driver V4L2 (ainsi que V4L1, première version des drivers). Le problème étant que nous ne pouvons pas nous passer d'un système automatisé pour la gestion de la mémoire : le traitement d'images requière le processeur à un rythme plutôt soutenu. Il est donc primordial de ne pas « gâcher »du temps de calcul inutilement en copie de mémoire. Le système Memory Mapping s'est avéré être un bon choix. De plus il était compatible avec nos drivers de webcam, la solution optimale dans ce cas précis.

5.1.2 Réécriture des drivers

Une solution envisagée mais irréalisable à notre niveau (cours sur l'écriture de drivers en 5^{e} année) serait la réécriture d'un driver pour une excellente raison : quand nous traitons le flux vidéo, nous négligeons en moyenne 50% de l'information. Il serait intéressant de ne pas enregistrer la chrominance rouge et bleu mais uniquement la luminance. Les buffers seraient réduits de moitié dans le cas du format YUV 422 (33% de réduction dans le cas du YUV 420, mais ce mode n'était pas compatible avec notre webcam). Cette hypothèse pourra donc être testée dans deux ans.

5.1.3 Changer le capteur CCD, ou la webcam plus simplement

Après de nombreuses optimisations nous sommes passés de 5 images par seconde au début du projet, où on affichait uniquement l'image d'origine, à environs 33 images par seconde en affichant l'image d'origine et l'image transformée. Malgré de nouvelles optimisations nous plafonnons à ce débit d'affichage de 33 images par seconde (débit plutôt correct comparé aux applications android qui peinent à dépasser les 20 images par seconde). Et cette limite ne vient pas d'un problème de vitesse de calcul car notre processeur ne dépasse pas les 60% d'utilisation et il nous est possible de mettre « en pause »notre logiciel durant quelques millisecondes sans passer en dessous des 33 images par seconde. Il semblerait donc que notre webcam ne soit pas capable de réaliser plus de 33 captures par seconde et son changement par une caméra à haute vitesse (par exemple le Casio Exilim EX-F1 qui peut filmer jusqu'à 300 images par secondes dans une résolution de 512x384, ou 60 images par seconde dans une résolution haute définition 1920 x 1080) nous permettrait de réaliser les limites de notre application.

5.2 Developpement Logiciel

5.2.1 Ecriture des pixels en temps réel

La première écriture de l'image à l'écran se faisait après le calcul total de l'image : lorsque nous étudions le buffer, nous remplissions la matrice équivalente à l'image. Une fois cette matrice complète, on réalisait une double boucle permettant de parcourir toute la matrice, et pour chaque pixel de l'afficher à l'écran.

Pour optimiser l'affichage de l'image nous écrivons à l'écran les pixels lors du calcul de l'image transformée. Pour chaque pixel calculé, on affiche à l'écran le pixel original d'un côté, et le pixel transformé de l'autre. Ainsi on évite des boucles inutiles ou redondantes.

5.2.2 Traiter les buffer en temps « réel »

L'idée que nous avons eu était de traiter les buffers au fur et à mesure qu'ils se remplissent, dans l'optique de ne pas devoir attendre un buffer complet à chaque fois. Le problème rencontré était que la vitesse du driver était sensiblement variable, ainsi il nous arrivait souvent d'accéder à une partie non remplie ne nous étant donc pas encore destinée. Un système d'exclusions mutuelles (MUTEX) devrait pouvoir régler ce souci, pour pouvoir réguler les accès aux données. On peut donc se poser la question : ce système peut-il être intéressant une fois le mutex mis en place ? Ne revient-il pas à segmenter les buffers au final ? De plus le driver ne facilite pas la tâche, il ne propose pas nativement de fonctions permettant de récupérer un buffer à moitié rempli, mais comme nous avions les pointeurs des buffers nous avons quand même eu un accès aux buffers en cours de remplissage. La réécriture des drivers pourrait faciliter cette intégration.

5.3 Traitement d'Images

5.3.1 Implémenter de nouvelles fonctions

Etant donné que la capture est totalement opérationnelle, qu'elle est optimisée et donne de très bons résultats, il serait intéressant de mettre en place de nouveaux processus image. La détection de forme simple, de visage ou d'objet permettrait de comparer notre logiciel aux résultats créés par la librairie OpenCv.

5.4 Comparaison avec OpenCv

La librairie OpenCv est une librairie de capture d'images et plus principalement de traitement d'images. Cette librairie est utilisée par une grande quantitée de laboratoire de traitement d'images dans le monde (première librairie utilisée dans le traitement d'images). Celle-ci utilise les drivers par défaut de la machine, mais on a aussi la possibilité d'utiliser les drivers V4L2 (ou V4L). Pour comparer notre logiciel et une application générée avec la bibliothèque de développement OpenCv, nous avons fait réaliser aux deux logiciels la même tâche.

La librairie OpenCv possède ses propres outils de fenêtrage et d'affichage, en théorie optimisés, et nous les utiliserons donc dans le logiciel généré avec OpenCv. Dans le cas de notre application, nous privilégerons la librairie SDL (Simple DirectMedia Layer, librairie utilisée durant nos deux années dans le parcours PEIP) qui a su nous convaincre dans le passé.

5.4.1 Affichage simple du flux vidéo

Ce premier test consistait simplement à afficher les images capturées par notre webcam sur l'écran. La version OpenCv tourne en moyenne à 19 images par secondes avec un pic à 20. Notre logiciel affiche en moyenne 32 images par secondes avec un pic à 33. C'est une victoire intéressante de notre logiciel.

5.4.2 Affichage simple de détection de contours sur le flux vidéo

Ce second test consistait à afficher une détection de contours en temps continu, sans afficher l'original. La version OpenCv tourne en moyenne à 18 images par secondes avec un pic à 19. Notre logiciel affiche en moyenne 32 images par secondes avec un pic à 33. C'est une nouvelle victoire pour notre logiciel.

5.4.3 Affichage simultané du flux vidéo et de de la détection de contours sur le flux vidéo

Cet ultime test vise en particulier les outils de fenêtrage et d'écriture sur l'écran. L'affichage est composé d'une part du flux vidéo original, et d'une autre part de la détection de contours. La version OpenCv tourne en moyenne à 16 images par secondes avec un pic à 18. Notre logiciel affiche en moyenne 32 images par secondes avec un pic à 33. C'est une victoire pour notre logiciel, ainsi que pour la SDL qui ne ralentie nullement l'écriture des images à l'écran.

5.4.4 Conclusion sur ce comparatif

Même si ce comparatif reste limité et peu complet, il met en avant une faiblesse de la librairie OpenCv : ses multitudes conversions de format qui ralentissent énormément le système. OpenCv utilise des formats particuliers pour stocker les images en couleurs, en niveaux de gris ou pour certaines transformations telles que la détection de contours. Ces conversions prennent du temps et ont un impact direct sur la fluidité des traitements. En passant directement par les drivers et en optimisant toutes les procédures, de la capture à l'affichage, notre démarche se voit récompensée d'une productivité largement supérieure dans les conditions actuelles. Pour aller plus loin, la detection de visage en temps continu (avec OpenCv) dans son mode par défaut ne dépasse pas les 6 images par secondes en moyenne (un utilisateur parle même de 3 images par secondes, nos tests sont plus autours de 6). Une fois optimisée, on peut espérer atteindre les 15 images par secondes. Ces résultats soulèvent de nombreuses interrogations : si nous pouvons détecter des contours, afficher l'image originale et l'image modifiée, le tout avec un rythme de 32 images par secondes en moyenne, peut-on réaliser une détection de visage plus efficace que OpenCv ?

Conclusion

Le projet ne s'est pas déroulé comme nous le pensions à l'initial. Dans un premier temps nous avons pensé utiliser OpenCv et réaliser différents algorithmes pour détecter des personnes, des visages etc. Mais nous avons compris alors que ce n'était pas du tout l'objectif principal de ce projet. Une fois les objectifs clairement compris par notre binôme, nos recherches ont débuté.
Les recherches sont une partie très importante de ce projet. La documentation sur V4L et V4L2 a posé de nombreux problèmes. Les documents traitant de V4L, particulièrements incomplets, qualifient souvent cette bibliothèque comme obsolète et conseillent aux developpeurs de passer à la version 2. Sur les conseils de M.Aupetit, nous avons décidé d'étudier la version 2. Notre première surprise fut de remarquer que tous les « flags »et toutes les fonctions n'étaient pas rétro-compatibles avec la version 1. Nos recherches ne servaient donc plus à rien : pire, les noms ont tous changés de manière significative.
La documentation sur Vidéo For Linux 2 est sensiblement plus complète et uniquement en anglais (notre TOEIC ne s'en portera que mieux). Malheureusement, les explications sont plutôt techniques, non vulgarisées et il y a encore quelques redirections vers la documentation de V4L première version, notée comme obsolète. Les forums ne traitent quasiment pas de V4L2, en général les utilisateurs conseillent plutôt de passer par une bibliothèque comme OpenCv, pour faciliter les traitements et le developpement. Ainsi, les seuls forums où nous avons pu trouvé un problème similaire aux notres se sont souvent soldés par l'utilisation d'OpenCv.
Concernant le déroulement du projet, il s'est surtout effectué par « tâtonnement ». Nous avancions par expérience, étude de la mémoire et intuition principalement. Notre première avancée fut simple mais réellement encourageante : la led de la webcam qui s'allume ! On savait alors déclencher une capture, mais on avait encore aucune idée de comment récupérer les images. De fil en aiguille nous avons réussi à récuperer des données dans des fichiers RAW. Nos connaissances sur matlab nous ont permis alors d'étudier cette première « image ». Nous avons cherché des redondances d'informations, en prenant une capture dans le noir et une capture avec forte luminosité (lampe dirigée vers le capteur). Lorsque qu'une redondance de 0 ou de 255 apparaissait, nous savions que nous touchions au but. Notre projet a avancé jusqu'au point de réussir la récupération totale de l'image. A ce moment tout s'est accéléré et nous avions enfin le fruit de notre travail.
Ce projet nous a donné la possibilité de mettre en application de nombreux cours tels que : Système d'Exploitation, Programmation Scientifique, Traitement d'Images, Traitement du signal et Langage d'Assemblage. Il nous a aussi conforté dans notre intéret pour le traitement d'Images et l'éventuel choix d'un PFE portant sur ce thème. Ce projet nous a aidé à assimiler des principes de cours que nous pensions maitriser et surtout développer nos connaissances dans ces divers domaines.
Nous espérons pouvoir améliorer notre logiciel avec les différents cours de 4^{e} année et de 5^{e} année.
Nous terminons ce rapport en remerciant notre encadrant M.Delalandre pour ses explications, sa compréhension et ses nombreux encouragements.

Bibliographie

[1] livre, *Operating System*.
Andrew Tanenbaum

[2] forum, *le Site du Zéro : utiliser le périphérique /dev/video0 comme point d'entrée*.
http ://www.siteduzero.com/forum-83-748326-p1-capture-de-flux-video-sous-linux.html

[3] site web, *kraxel : projet xawtv, open source, application TV utilisant V4L, projet abandonné*.
http ://www.kraxel.org/blog/linux/xawtv/

[4] forum, *Altera : créer un lien avec la webcam en connectant /dev/video0 à un pointeur de type fichier*.
http ://www.alteraforum.com/forum/showthread.php ?p=67632

[5] site web, *Wikipédia : page de présentation de Video For Linux, liste de projets utilisants V4L*.
http ://fr.wikipedia.org/wiki/Video4Linux

[6] site web, *exploits.org/v4l : page de présentation de projets utilisants V4L*.
http ://www.exploits.org/v4l/

[7] site web, *antonym : page de présentation de projets TV V4L*.
http ://antonym.org/libfg/

[8] site web, *V4L2spec : specifications sommaires de V4L2*.
http ://v4l2spec.bytesex.org/spec/

[9] site web, *linuxtv : spécifications techniques de V4L2*.
http ://www.linuxtv.org/downloads/legacy/video4linux/API/V4L2_API/spec-single/v4l2.html

[10] forum, *le Site du Zéro : problème de capture vidéo, ressource webcam busy*.
http ://www.siteduzero.com/forum-83-735538-p1-probleme-v4l-opencv-libunicap-et-dazzle-dvd-recorder.html

[11] forum, *Ubuntu-fr : installation de drivers V4L2 pour une webcam*.
http ://forum.ubuntu-fr.org/viewtopic.php ?id=221645

[12] site web, *V4L2spec : spécifications V4L2 sur les buffers*.
http ://v4l2spec.bytesex.org/spec/x5953.htm#V4L2-BUFFER

[13] site web, *Thedirks : spécifications sur les capacités webcam V4L2*.
http ://www.thedirks.org/v4l2/v4l2cap.htm

[14] site web, *Abul : installation de drivers V4L et V4L2*.
http ://www.abul.org/Piloter-une-webcam.html

[15] forum, *Ubuntu-fr : problème de rétro-compatibilité entre V4L et V4L2*.
http ://forum.ubuntu-fr.org/viewtopic.php ?id=314977

[16] site web, *Alumnos : présentation technique de V4L2*.
http ://alumnos.elo.utfsm.cl/ỹanez/video-for-linux-2-sample-programs/

[17] site web, *Nikonpassion : informations sur le format RAW*.
http ://www.nikonpassion.com/quest-ce-que-format-raw/

[18] site web, *LinuxForDevice : introduction à V4L2*.
http ://www.linuxfordevices.com/c/a/Linux-For-Devices-Articles/Intro-to-V4L2/

[19] site web, *univ-lyon : cours de traitement d'images, informations sur le Thresholding et OpenCv*.
http ://www710.univ-lyon1.fr/b̃ouakaz/OpenCV-0.9.5/docs/ref/OpenCVRef_ImageProcessing.htm#decl_cvThres

[20] site web, *cs.iit.edu : introduction à la programmation OpenCv*.
http ://www.cs.iit.edu/ agam/cs512/lect-notes/opencv-intro/opencv-intro.html

[21] forum, *le Site du Zéro : incompatibilité entre OpenCv et certains drivers de webcam*.
http ://www.siteduzero.com/forum-83-753347-p1-opencv-divers-problemes.html

[22] site web, *V4L2spec : spécification la plus complète sur V4L2, notre BIBLE lors du projet*.
http ://v4l2spec.bytesex.org/spec-single/v4l2.html

[23] site web, *V4L2spec : spécification sur les encodages de couleurs, particulièrement le format YUV*.
http ://v4l2spec.bytesex.org/spec/r4339.htm

[24] forum, *Developpez : problème pour décoder des images au format YUV*.
http ://www.developpez.net/forums/d1078894/c-cpp/cpp/format-image-yuyv-image-verte-lors-rendu/

[25] forum, *le Site du Zéro : performance de OpenCv sur la detection de visage*.
http ://www.siteduzero.com/forum-83-494611-p1-opencv-performance-detection-de-visage.html

Interaction utilisateur PC-webcam

Département Informatique
3e année
2011 - 2012

Projet Système d'Exploitation

Résumé : Ce projet a pour but de capturer des images à partir d'une webcam et de les traiter informatiquement ensuite.

Mots clefs : webcam, traitement d'images, accès direct à la mémoire, mappage de la mémoire

Abstract: This project aims to capture images from a webcam and then process them by computer.

Keywords: webcam, image processing, Direct Memory Access, Memory Mapping

---

Encadrant
Mathieu DELALANDRE
mathieu.delalandre@univ-tours.fr

Université François-Rabelais, Tours

Étudiants
Clément TESSIER
clement.tessier@etu.univ-tours.fr
Raphaël ROGER
raphael.roger@etu.univ-tours.fr

DI3 2011 - 2012