UCLouvain

École polytechnique de Louvain

Système de suivi du mouvement des lèvres en temps réel par stéréovision

LMTS 2.0

Auteurs: **Gauthier De Coninck, Guillaume Derycke**
Promoteurs: **Benoît Herman, Christophe De Vleeshouwer**
Lecteurs: **Martin Edwards, Benoît Macq**
Année académique 2019–2020
Master [120] : ingénieur civil électromécanicien

Remerciements

Nous souhaitons remercier nos deux promotteurs, Benoit Herman et Christophe De Vleeschouwer, pour leur engagement, leur disponibilité et l'aide qu'ils nous ont apportée tout au long de cette année académique.

Nous remercions également Martin Edwards d'avoir financé l'achat de la caméra stéréo de notre dispositif.

Nous remercions particulièrement Antoinette Hoppe qui a utilisé les versions successives de notre application "LMTS Studio 2.0" et nous a fourni un feedback régulier permettant d'y apporter de nombreuses améliorations.

Enfin, nous remercions toutes les personnes qui nous ont aidés dans cette aventure, qui ont accepté de se faire maquiller pour des tests et qui nous ont aidés à relire le présent rapport.

Abstract

Suivre le mouvement des lèvres est utile dans divers domaines dont la logopédie, où une observation objective de ces mouvements aiderait au diagnostic d'un patient. Ce mémoire développe un système complet "low-cost" destiné avant tout à la recherche. Il permet le suivi en temps réel des mouvements labiaux dans les trois dimensions spatiales grâce à un traitement d'images.

La partie hardware du dispositif se compose d'un casque, qui se place sur la tête du patient, sur lequel une caméra stéréo industrielle filme la bouche. Des marqueurs colorés (points de maquillage ou gommettes autocollantes) sont appliqués au niveau des points d'intérêt du contour des lèvres.

Les images obtenues sont traitées dans une application Windows nommée « LMTS Studio 2.0 ». Son interface graphique est facile d'utilisation et permet d'initialiser manuellement les marqueurs à suivre. Une segmentation de couleur par seuillage évolutif est réalisée, au choix, dans l'espace de couleur RGB, HSV, Lab ou YUV. Pour atteindre un fonctionnement en temps réel (minimum 40 images traitées par seconde), le traitement d'images se fait sur des régions d'intérêt (ROI) tout en faisant un usage intensif du multithreading.

Le taux de réussite de détection de marqueurs est en moyenne de 99.7% sur 10h de vidéos réalisées en conditions réelles avec des marqueurs dessinés au maquillage. La mesure de profondeur par stéréovision montre quant à elle une erreur de moins de 1.0%.

Table des matières

1 Introduction **1**

2 Traitement d'images **4**
2.1 État de l'art de la détection labiale par traitement d'images 4
2.1.1 Introduction 4
2.1.2 Sans marqueurs 4
2.1.3 Avec marqueurs 7
2.2 Algorithme 11
2.2.1 Outils de traitement d'images : 12
2.2.2 Algorithme de suivi des marqueurs : 16
2.2.3 Tests de colorimétrie et de marqueurs 21
2.3 Tests de l'algorithme 25
2.3.1 Données Réelles 25
2.3.2 Données synthétiques 28
2.3.3 Taux de réussite du suivi de points en conditions réelles . . . 31
2.4 Impact des paramètres de traitement 35
2.5 Conclusions sur l'algorithme développé 36

3 Dispositif d'acquisition des images **37**
3.1 Modélisation et calibration d'une caméra 38
3.1.1 Le modèle sténopé 38
3.1.2 Distorsion de l'image 40
3.1.3 Calibration 41
3.2 La stéréovision 42
3.2.1 Principe de base 42
3.2.2 Stéréovision à miroirs 44
3.2.3 Étude de faisabilité 50
3.3 Choix de la caméra 52
3.3.1 Deux caméras indépendantes 53
3.3.2 Caméra stéréo pré-montées 53

3.4 Choix des optiques . . . 54
3.5 Conception du casque . . . 56
3.5.1 Casque du LMTS originel . . . 56
3.5.2 Cahier des charges pour le nouveau casque . . . 57
3.5.3 Premier prototype du casque . . . 57
3.5.4 Version finale du casque . . . 60
3.6 Mise en pratique de la stéréoscopie . . . 62
3.6.1 Transformation des coordonnées "caméra" vers des coordonnées 3D . . . 62
3.6.2 Tests de stéréoscopie . . . 65
3.7 Conclusions sur le dispositif final . . . 69

4 Intégration : LMTS Studio 2.0 71
4.1 Langage et bibliothèques utilisées : . . . 71
4.2 Multithreading : . . . 72
4.2.1 Outils de communication entre threads . . . 74
4.2.2 Implémentation et challenges . . . 81
4.3 Interface caméra . . . 86
4.4 Enregistrement du son . . . 88
4.5 Interface graphique . . . 89
4.6 Conclusions sur l'application complète . . . 91

5 Discussion et Conclusions 92

A Espaces colorimétriques 99

B Algorithmes du traitement d'image 104

C Histogrammes de couleur : 108

D Stéréoscopie avec des caméras parallèles 112

E Spécifications de la caméra stéréo 115

F Résultat de la calibration des caméras stéréos 117

G Pièces 3D du premier prototype du casque du LMTS 2.0 119

H Listing des pièces composant le casque du LMTS 2.0 121

I Structure du code 123

J Mode d'emploi du LMTS Studio 2.0 126

J.1 Installation 126
J.2 Description de l'interface graphique 129

Notations utilisées

LMTS	Lip Motion Tracking System
fps	(Frame Per Second) Nombre d'images par seconde.
FPGA	Field Programmable Gate Arrays
ROI	*Region of interest*, Sous-image traitée
RGB	Espace de couleur *Red - Green - Blue*
HSV	Espace de couleur *Hue - Saturation - Value*
Lab	Espace de couleur *Luminance - color A - color B*
YCbCr ou YUV	Espace de couleur *Luminance - Blue difference - Red difference*

Chapitre 1

Introduction

Les lèvres constituent un organe important dans la parole. Celles-ci ont à la fois un rôle dans l'appareil phonatoire, pour l'articulation de certains sons, et dans le langage non verbal. Le suivi du mouvement des lèvres est donc utile dans différentes applications. Ces dernières vont du déplacement des lèvres de personnages de films d'animation à des applications médicales, en passant par de la reconnaissance de visages et des applications de sécurité.

L'objectif de ce mémoire est de fournir un nouvel outil destiné à des fins de recherche et à une utilisation en logopédie. En effet, les troubles dysarthriques, c'est-à-dire liés à l'articulation et aux troubles du langage [1], peuvent prendre différentes formes en fonction de leurs origines. Un dispositif de diagnostic est dès lors utile pour déterminer et différencier les types de dysarthries, mais aussi pour effectuer un suivi objectif des patients. En effet, il est difficile d'observer de manière quantitative la manière dont une personne articule, et il est encore plus complexe de pouvoir apprécier l'évolution de celle-ci séance après séance.

L'étude de la position et du mouvement de points particuliers du contour des lèvres est pour cela utile. En fonction de l'expérience, un nombre plus ou moins important de points peux être utilisé. Le suivi de cinq points tel qu'illustré Figure 1.1 permet de déterminer :

- l'ouverture et la vitesse d'ouverture de la bouche sur base des points supérieurs et inférieurs,
- la largeur de la bouche sur base notamment des points aux commissures,
- la non-symétrie d'une ouverture de la bouche grâce aux deux points supérieurs,
- le périmètre de la bouche sur base des cinq points.



FIGURE 1.1 – Poins d'intérêts sur le contour d'une bouche.

En 2015, Lise Colmant a conçu un dispositif répondant à ce besoin [2]. Sur base d'une GOPRO (caméra d'action) attachée à un casque de vélo, le suivi des lèvres est fait par traitement d'images. Pour ce faire, des gommettes autocollantes bleues sont apposées sur la peau aux points d'intérêt autour de la bouche. La GOPRO enregistre le flux vidéo sur sa carte mémoire et la vidéo est traitée une fois l'acquisition terminée.

Par la suite, le traitement d'image a été revu par Pierre Venet et une application nommée LMTS studio a été créée pour permettre le post-traitement des vidéos obtenues avec la GOPRO.

Le système ainsi développé a été utilisé dans différents mémoires pendant et après sa création [3, 4, 5, 6, 7, 8]. Les tests effectués dans [2, 6, 8] valident le dispositif, notamment en le comparant à un dispositif de référence dans le suivi du mouvement (Codamotion).

En 2016, une tentative peu concluante de remplacer la GOPRO par un Raspberry Pi et son module caméra a été menée par Abdelilah Sellami [9].

Notre objectif est de fournir un outil plus adapté à une utilisation courante dans un cabinet de logopédie et offrant davantage de possibilités. Le choix a été fait de maintenir l'utilisation d'un traitement d'image pour effectuer le suivi. Parmi les alternatives explorées, celle-ci est la moins contraignante pour le patient et est également la moins onéreuse.

Deux points forment notre objectif primaire. Tout d'abord, suivre le mouvement des lèvres dans l'espace et plus seulement dans un plan : ce qui offre à l'utilisateur des données supplémentaires à analyser. Ensuite, obtenir un fonctionnement en temps réel. De plus, le dispositif fini doit rester "low-cost" pour permettre à des cabinets de logopédie de l'acquérir.

Le fonctionnement en temps réel consiste à au moins avoir un feedback sur le bon fonctionnement du traitement pendant un test sur patient. L'intérêt de ce feedback est de garantir l'exploitabilité d'un fichier vidéo. Celle-ci pouvant être compromise par un mauvais placement de la caméra ou de mauvaises conditions

d'éclairage. L'intérêt est également de réduire les étapes manuelles nécessaires entre l'acquisition des données et l'obtention des résultats de position.

Le choix a été fait de reprendre un principe de fonctionnement semblable à celui du LMTS de Lise Colmant : un dispositif fixé sur la tête du patient mais utilisant cette fois-ci une caméra stéréo pour filmer la bouche sous deux angles de vue différents. Des points de couleur (généralement bleue) sont placés sur le bord des lèvres pour servir de marqueurs. Ce sont les mouvements de ces marqueurs et non pas celui des lèvres en tant que tel qui sont suivis. Suite aux désagréments rencontrés avec l'utilisation de gommettes auto-collantes, nous avons proposé de les remplacer par des marques de couleurs faites au maquillage. L'utilisation de celles-ci, non-obligatoire [1], permet de placer les marqueurs directement sur les lèvres sans gène pour le patient.

Enfin, une interface utilisateur permet d'initialiser le suivi de couleur adaptatif et d'obtenir les positions successives des points d'intérêts dans le repère image (les repères images dans le cas stéréo), mais aussi dans l'espace.

Ce mémoire sera séparé en trois chapitres rendus les plus indépendants possibles et ayant pour sujet :

- l'algorithme de traitement d'images utilisé,
- le dispositif d'acquisition d'images stéréo,
- l'intégration de l'algorithme et de l'utilisation du dispositif d'acquisition via une application Windows.

Le chapitre 2 présentera donc le choix et la façon de faire une segmentation de couleur pour suivre la position des marqueurs dans des sous-images.

Le chapitre 3 abordera tous les aspects liées à la caméra, à la stéréovision et au casque placé sur la tête du patient.

Enfin, le chapitre 4 traitera du développement de toute l'interface graphique et de l'utilisation du multithreading dans une situation peu favorable afin de rendre le traitement le plus rapide possible.

1. Le traitement implémenté fonctionne tout aussi bien avec les gommettes auto-collantes qu'avec des points de maquillage.

Chapitre 2

Traitement d'images

Avant d'expliquer l'algorithme que nous avons utilisé et afin d'expliquer les choix qui ont été faits, un état de l'art des différentes solutions qui s'offraient à nous est nécessaire. Le choix d'utiliser des marqueurs ajoutés sur le visage du patient sera essentiellement justifié par la vitesse du traitement qui en découle et la nécessité pour les logopèdes de connaître la position de cinq points particuliers. La segmentation de couleur dans des sous-images (ROI) sera ensuite expliquée conceptuellement. Sa mise en pratique sera quant à elle abordée dans le chapitre 4.

2.1 État de l'art de la détection labiale par traitement d'images

2.1.1 Introduction

De manière générale, le suivi d'objets par traitement d'images peut se faire de deux manières différentes (Figure 2.1) : avec ou sans marqueurs physiquement présents dans les images à analyser. Le but de tels marqueurs est évidement de faciliter la détection de points d'intérêts. Il est ainsi plus simple de rechercher des gommettes bleues sur le visage de quelqu'un que de détecter précisément le contour de lèvres.

Des techniques avec et sans marqueurs adaptées à notre situation sont ci-après présentées.

2.1.2 Sans marqueurs

Le suivi des lèvres sans marqueurs se base uniquement sur l'apparence normale du visage. Ce type de traitement a été récemment fort développé dans toutes sortes

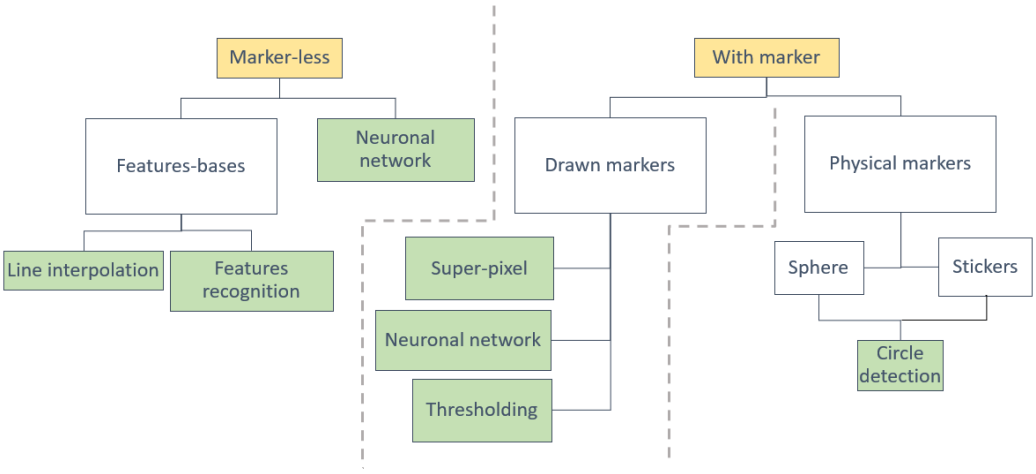


FIGURE 2.1 – Comparaison des méthodes de traitement d'images avec et sans marqueurs

d'applications liées au quotidien pour lesquelles il n'est pas envisageable d'utiliser des repères supplémentaires sur les personnes : reconnaissance de parole, de visages, d'émotions, ... Malgré la multiplicité des techniques, il est pourtant possible de séparer la plupart de celles-ci en deux catégories :

- Segmentation d'image basé sur les contours et suivi de ceux-ci par un modèle géométrique,
- Segmentation d'image par utilisation de la couleur.

L'utilisation de réseaux neuronaux (basé sur de l'intelligence artificielle) peut être utiles dans les deux cas.

Segmentation d'image basé sur les contours

La segmentation d'image consiste à trouver des groupes de pixels "qui vont ensemble" [10]. Celle-ci se base souvent sur un calcul de contours. Les contours sont obtenus par calculs des gradients de luminance $I(\mathbf{x})$ (ou de couleur) :

$$\mathbf{J}(\mathbf{x}) = \nabla I(\mathbf{x}) = \Big(\frac{\delta I}{\delta x}, \frac{\delta I}{\delta y}\Big)(\mathbf{x})$$

Les méthodes développées dans les articles [11], [12], [13] et [14] utilisent des méthodes dite de contours actifs (souvent appelés "snake") plus ou moins évoluées. De manière générale, il s'agit d'une minimisation énergétique entre l'obtention d'une courbe "lisse" et la distance de celle-ci par rapport aux contours donnés par les maxima de gradient de luminance. Sachant qu'une courbe elliptique fermée peut

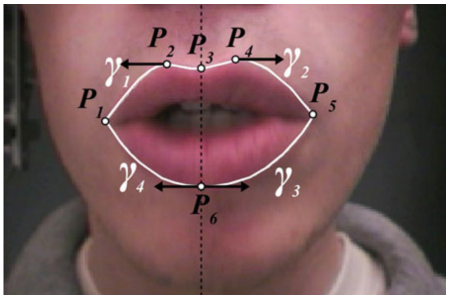


FIGURE 2.2 – Courbe paramétrique formée de 6 points clés P_i, 4 fonctions cubiques γ_i et deux segments droits [14].

se représenter mathématiquement comme [15] :

$$v_N(\theta) = \begin{pmatrix} x_N(\theta) \\ y_N(\theta) \end{pmatrix} = \sum_{k=0}^{N} A_k \begin{pmatrix} cos(\theta k) & sin(\theta k) \\ cos(\theta k) & sin(\theta k) \end{pmatrix} \tag{2.1}$$

Où N est le nombre d'harmoniques utilisées pour décrire la courbe,

$$A_k = \begin{pmatrix} a_{11,k} & a_{12,k} \\ a_{21,k} & a_{22,k} \end{pmatrix}$$

est un ensemble de paramètres permettant de décrire la courbe désirée.
La fonction d'énergie à minimiser est la suivante :

$$E(v) = \int_0^{2\pi} -|G(\nabla I)|^2 + \lambda \cdot (v_\theta(\theta))^2 \, d\theta \tag{2.2}$$

Où G est une fonction de lissage et λ le poids associé à la contrainte de lissage de la courbe. La méthode de minimisation (itérative) peut varier d'une source à l'autre ainsi que la fonction d'énergie elle-même. L'utilisation de différents types de courbes est aussi étudiée, comme des courbes ouvertes dans certains cas [15]. Stillittano et al. (2013) [14] développent par exemple un modèle de courbe paramétrique à 6 points clés reliés par 4 fonctions cubiques et deux segments (Figure 2.2). L'optimisation du suivi du contour intérieur et extérieur des lèvres est généralement très dépendante des conditions initiales données à la courbe. Et la gestion du contour intérieur, pour lequel les situations de bouche ouverte et fermée sont très différentes, est un problème ardu, notamment à cause de la visibilité des dents à l'intérieur de la bouche qui évolue. Le choix de la flexibilité des courbes qui sont utilisées pour approcher les bords des lèvres est une affaire de compromis : un modèle peu flexible donnera toujours une forme de lèvre plausible mais qui ne collera pas correctement aux formes inhabituelles, alors qu'un modèle très flexible risque de donner des formes non-réalistes [16]. Afin de contrer ce problème, une approche

par modélisation statistique est plébiscitée. La forme est apprise grâce à une série d'exemples servant de base de données, de sorte que la seule limitation soit la diversité des exemples d'apprentissage face à des configurations spéciales. Gacon et al. (2005) [16] présentent un modèle statistique hybride basé sur une minimisation de l'erreur quadratique d'une interpolation obtenue notamment par un réseau de neurones entraîné. Seyedarabi et al. (2006) [13] quant à eux utilisent un réseau neuronal pour analyser les contours obtenus par un contour actif et ainsi déterminer les émotions affichées par le visage.

Utilisation de la couleur :

Une manière très différente de segmenter les lèvres est de ne pas se baser sur les contours, mais sur les lèvres elles-même et plus particulièrement leur couleur. A cause des variations de luminosité, la présence ou non de poils, de maquillage, de rouge à lèvre, ... sur le visage à analyser, l'utilisation d'un réseau de neurones se révèle intéressante [17, 18]. Pour chaque pixel formant une image, l'objectif est de déterminer dans un premier temps si il fait partie d'un visage, puis si il est potentiellement partie des lèvres ou non. Le type de résultats obtenus est illustré Figure 2.3 où chaque zone légèrement rouge passe dans un premier temps pour des lèvres et où tous les faux positifs sont évacués par un post-traitement très sensible à la géométrie des lèvres mais malheureusement peu adapté à une mise à l'échelle.

2.1.3 Avec marqueurs

Les techniques de suivi de mouvement avec marqueur ne sont évidemment pas spécifiques au suivi des mouvements labiaux. Certaines d'entre-elles sont cependant applicable à notre cas. Comme nous l'avons déjà mentionné, l'utilisation de marqueur a pour but de simplifier ou tout du moins de globaliser le traitement : quelque soit le sujet des déplacements à retracer, l'objectif est de déterminer les positions des marqueurs apposés. Cela signifie également que l'utilisation pour les lèvres pourrait être aisément transposé à bien d'autre parties du corps ou même à d'autres objets.

Suivi de cercles colorés

Cette méthode utilise comme marqueurs des billes colorées (voir Figure 2.4). Elle exploite les propriétés visuelles et géométriques des marqueurs pour les distinguer des autres éléments de l'image. Dans un premier temps, l'algorithme va sélectionner les parties de l'image possédant la même couleur que les marqueurs. Une fois cela fait, la forme sphérique des marqueurs (apparaissant toujours comme un cercle à l'image) va être utilisée afin d'écarter les objets parasites. Le contour des différentes

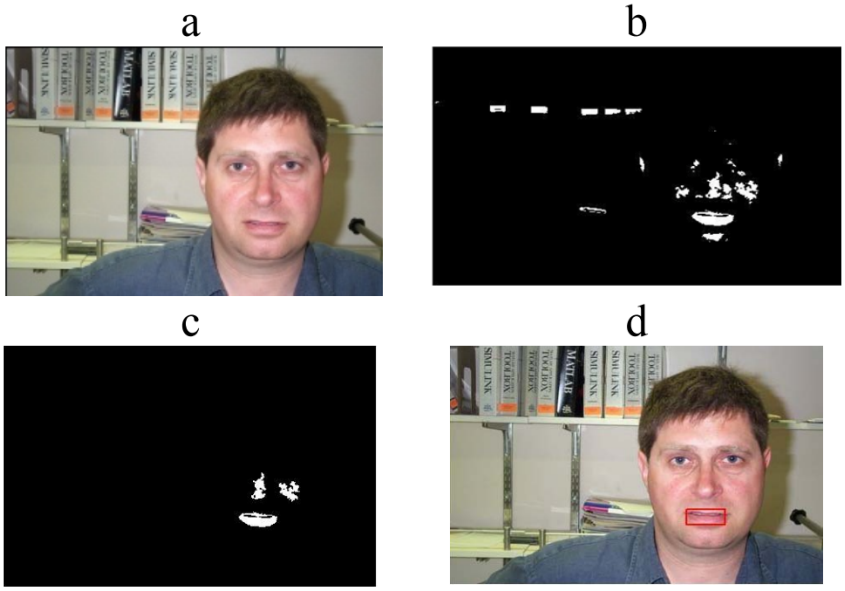


FIGURE 2.3 – Test de segmentation basé sur la couleur [18]. (a) Image originale, (b) image segmentée sur base d'un réseau entraîné, (c) après opération d'ouverture sur 200 pixels, (d) lèvres détectées en utilisant une contrainte entre la hauteur h et la largeur w de la zone segmentée : $w > 1,2 \cdot h$.

zones trouvées va être extrait et comparé à un cercle de même aire que la zone. Si le contour est suffisamment circulaire, il sera considéré comme un marqueur tandis que s'il ne l'est pas, il sera considéré comme objet parasite. Certains logiciels commerciaux utilisent cette méthode comme le Zign Track 2 afin de capturer les mouvements du visage en temps réel (voir Figure 2.5).



FIGURE 2.4 – Billes en polypropène [19]

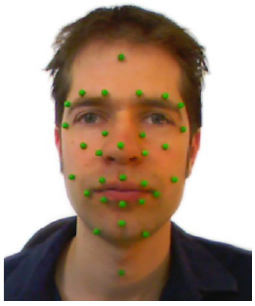


FIGURE 2.5 – Zign Track 2 [19]

Suivi d'ellipses colorées

Cette méthode fonctionne plus ou moins selon le même principe expliqué au point précédent. La principale différence vient du type de marqueurs utilisés : des gommettes adhésives (montrées sur la Figure 2.6). Celles-ci n'auront pas une forme circulaire à l'image mais elliptique. Pour déterminer quelle ellipse est la plus proche d'un certain contour, une transformée de Hough est appliquée. Cependant, le problème n'est pas aussi facile que de déterminer le cercle le plus proche. En effet, le cercle est caractérisé par 2 paramètres (la position de son centre et son rayon) tandis que l'ellipse est caractérisée par 4 paramètres (la position de son centre, son orientation, la longueur de son grand axe et la longueur de son petit axe). C'est cette méthode qui a été utilisée pour la première version du LMTS (voir Figure 2.7).

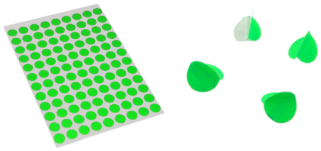


FIGURE 2.6 – Gommettes adhésives [19]

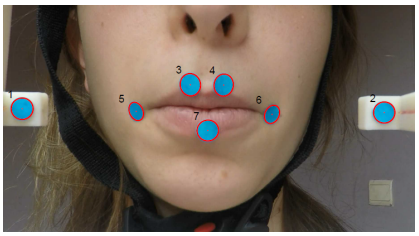


FIGURE 2.7 – Première version du LMTS [2]

Marqueurs rétro-réceptifs

Ce sont les principaux marqueurs utilisés dans la recherche et l'industrie audio-visuelle pour faire du suivi de mouvements (voir Figure 2.8). Il en existe de différentes tailles selon la partie du corps que l'on souhaite suivre. Pour ce qui est des marqueurs faciaux, leur taille varie entre 1.5 et 5mm et se collent grâce à du papier collant double face. Ceux-ci sont illuminés avec de la lumière infrarouge (longueur d'onde de 940 nm) provenant de sources lumineuses attachées à chaque caméra filmant la scène. A noter que ces marqueurs sont de forme sphérique afin que la lumière infrarouge puisse se réfléchir vers la caméra, peu importe son orientation par rapport au sujet. Ensuite, un filtre infrarouge est installé devant l'objectif de la caméra afin de bloquer la lumière visible d'une longueur d'onde inférieure à 820 nm [20]. Les couleurs de l'image vont donc saturer au niveau des positions des différents marqueurs ce qui permet de les discriminer plus facilement en utilisant des méthodes de seuillage d'image par exemple.



FIGURE 2.8 – Marqueurs rétro-réceptifs [21]

Marqueurs dessinés à la main

Dessinés au crayon, au maquillage ou au marqueur de couleur, ce sont les marqueurs les plus simples. Le principal défaut de ces marqueurs est la non-homogénéité de leur forme et, dans une moindre mesure, de leur couleur. Cette dernière étant la principale référence servant aux algorithmes de suivi. D'un autre côté, le grand avantage de ces marqueurs est le fait qu'ils n'entravent que très peu les gestes des sujets étudiés, ce qui les rendent très intéressants dans l'analyse des mouvements des rats par exemple (voir Figure 2.9) afin qu'ils puissent avoir un comportement le plus naturel possible.

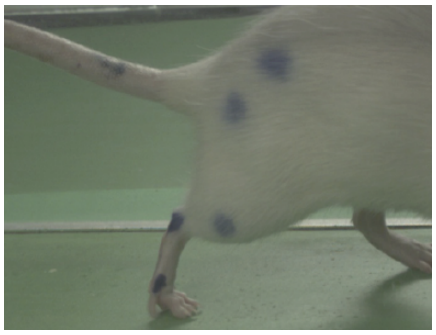


FIGURE 2.9 – Marqueurs dessinés sur un rat [22]

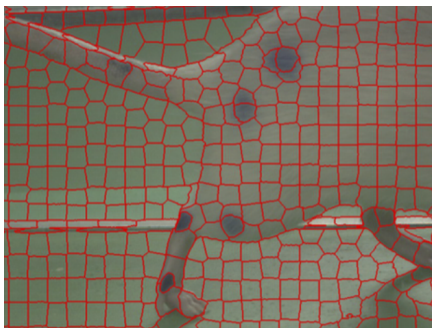


FIGURE 2.10 – Segmentation par "super-pixel" [22]

Pour la détection de ce type de marqueurs, une segmentation de couleur est généralement utilisée [22, 23]. Celle-ci peut se faire dans différents espaces colorimétrique comme nous le verrons par la suite.

Afin de combiner les informations spatiales et de couleurs, une segmentation dite **super-pixel** peut être utilisée [22, 24]. Une telle segmentation consiste à séparer une image de N pixels en K groupes de pixels (appelés super-pixels). L'algorithme SLIC (*simple linear iterative clustering*) est le plus utilisé [25]. Celui-ci se base sur une minimisation itérative de la "distance" entre les différents pixels d'un même

groupe. Cette distance D peut s'exprimer comme la moyenne pondérée entre une distance en couleur D_c et une distance en position D_p [22] :

$$D_c = \sqrt{(R_j - R_i)^2 + (G_j - G_i)^2 + (B_j - B_i)^2}$$
$$D_p = \sqrt{(x_j - x_i)^2 + (y_j - y_i)^2}$$
$$D = \sqrt{\alpha_c \cdot D_c^2 + \alpha_p \cdot D_p^2}$$

Où $(R, G, B)_i$ est la composante de couleur d'un pixel i et $(x, y)_i$ sa position. α_c et α_p sont des coefficients.

À chaque itération, les pixels formant les N groupes sont échangés entre les différents groupes pour obtenir un résultat final comme dans la Figure 2.10. Une fois l'image en super-pixel obtenue, les groupes dont la taille et la couleur correspondent à des marqueurs sont retenus comme tel.

Le désagrément d'une telle technique est le côté itératif qui augmente le coût en terme de processing. Shen et al. (2016) [25] présentent un algorithme légèrement différent pour un fonctionnement en temps-réel.

2.2 Algorithme

Afin d'atteindre plus aisément l'objectif d'avoir un traitement en temps réel, l'utilisation de marqueurs s'est rapidement imposée. Le traitement qui y est associé est généralement moins coûteux en terme de calcul. De plus, les métriques principales utilisées en logopédie se basent sur la position de 5 points bien déterminés sur le contours des lèvres. Le contour complet des lèvres n'est donc pas nécessaire et la récupération des 5 points d'intérêt sur base de l'expression du contour complet n'est pas une tâche triviale.

Comme annoncé précédemment, nous avons opté pour l'utilisation de marqueurs maquillés sur la peau du patient. Ceux-ci pouvant être suivi par une segmentation de couleur. La couleur initialisée manuellement pour chaque "tache" étant recherchée avec plus ou moins de précision (voir plus loin). Ce choix a été fait pour les raisons suivantes :

Rétro-compatibilité : la segmentation de couleur permet de traiter les vidéos que pouvaient traiter les algorithmes du LMTS précédent [2, 26]. Les gommettes auto-collantes apparaissent simplement comme des taches "parfaites".

Bonne tenue des marqueurs : Un problème régulièrement encouru lors de l'utilisation de marqueurs auto-collants près de la bouche est que ceux-ci se décollent ou "glissent" lorsque le patient, parfois porteur d'un handicap,

bave ou transpire beaucoup. Si le patient a de la barbe ou une moustache, les marqueurs auto-collants ne peuvent pas être posés.

Confort du patient : Celui qui n'est pas convaincu de ce point pourra toujours essayer par lui-même de coller des petits bouts de papier ou de plastique sur son visage ! Le ressenti est particulièrement désagréable, ce qui peut rendre le patient réticent à une utilisation postérieure et peut potentiellement affecter la façon dont la personne parle. Notons que nous avons testé plusieurs façon de dessiner les marqueurs (feutres et maquillages). Nous avons préféré l'utilisation de maquillage qui présente d'une part une bonne résistance à l'eau (bave et transpiration) tout en s'enlevant très facilement à l'eau savonneuse !

Polyvalence : La position et le nombre de marqueurs placés, la couleur ainsi que le choix de marqueurs dessinés ou auto-collants est laissé au praticien en fonction de la situation. Les marqueurs dessinés peuvent par-exemple être placés directement sur les lèvres pour un suivi plus rapproché par rapport à des gommettes.

Afin de réduire le coût de traitement, le suivi des marqueurs se fera dans des sous-images autrement appelées région d'intérêt (ROI pour *Region Of Interest*). De cette manière, seule les parties de l'image complète contenant les marqueurs est prise en compte. L'algorithme de suivi d'un marqueur sera donc "répété" pour chaque marqueur présent.

Avant d'aborder et pour préparer l'explication de l'algorithme mis en place (sous-section 2.2.2), quelques concepts et outils seront présentés à tour de rôle.

2.2.1 Outils de traitement d'images :

Une image numérique, obtenue grâce à un capteur digital, se présente de manière générale comme une matrice de points de couleur. Chacun de ces points est ce que l'on appelle un pixel. Les couleurs d'une image peuvent être représentées de différentes manières tout en gardant le même nombre d'informations. Ces représentations s'expriment généralement sous la forme d'un triplet pour chaque pixel d'une image. Les différents espaces de couleurs utilisés (RGB, HSV, Lab et YUV) sont décrits de manière complète dans l'Annexe A.

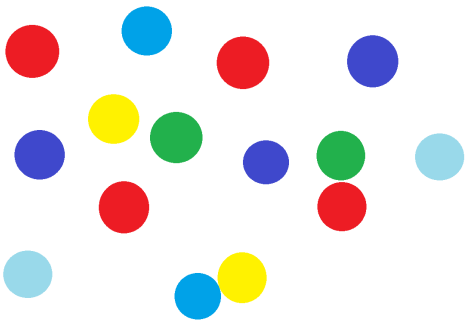
Les différents outils expliqués ci-après sont à la base de l'algorithme que nous avons implémenté.

Segmentation de couleur Les marqueurs colorés sont retrouvés par une segmentation de couleur. La technique utilisée pour détecter les marqueurs est nommé seuillage des couleurs (*thresholding* en anglais). Définissant les valeurs de couleur à

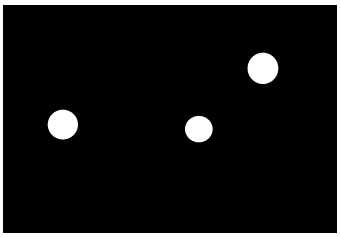
suivre C_i (dans l'espace de trois composantes choisi) ainsi que les tolérances Δ_i sur ces couleurs, chaque pixel in_j de l'image à segmenter devient [27] :

$$\begin{cases} out_j = 1 & \text{si } C_i - \Delta_i \leq in_j \leq C_i - \Delta_i \quad \forall i = 1, 2, 3 \\ out_j = 0 & \text{sinon} \end{cases} \tag{2.3}$$

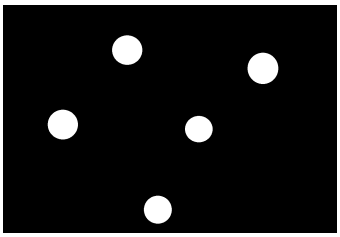
L'image segmentée est donc une image binaire (dont chaque pixel vaut 0 ou 1, ou autrement dit, il s'agit d'une image monochrome sans nuances). La segmentation de l'Équation 2.3 prend en compte chaque pixel indépendamment. Un exemple de segmentation de couleur est présenté Figure 2.11 pour différentes tolérances Δ_H sur la teinte de bleu à segmenter.



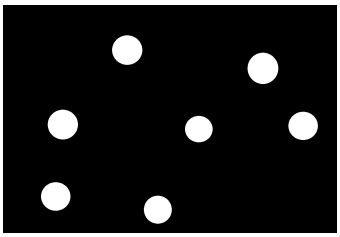
(a) Image couleur à segmenter



(b) Image segmentée ($\Delta_H = 10$)



(c) Image segmentée ($\Delta_H = 20$)



(d) Image segmentée ($\Delta_H = 30$)

FIGURE 2.11 – Exemple de segmentation d'une image sur base de ses composantes de couleur HSV $C_H = 118$ $C_S = 176$ $C_V = 204$ avec $\Delta_S = \Delta_V = 100$.

Sur base d'images binaires (telle que Figure 2.11b, 2.11c, 2.11d) deux outils mathématiques vont être utilisés. Ces deux outils, combinés, sont utiles pour pré-traiter les images binaires.

Érosion : L'érosion, comme le nom l'indique érode une image binaire. Mathématiquement, l'érosion de l'image binaire A par l'élément structurant B, notée $A \ominus B$,

peut se définir comme :

$$(A \ominus B)_z = \begin{cases} 1 & \text{si } B_z \subseteq A \\ 0 & \text{sinon} \end{cases}$$

Où z est un des pixels de A et B_z est la translation de B au niveau du pixel z. $B_z \subseteq A$ signifiant que B_z est inclus (ou égal) dans l'image A.
Prenons l'exemple simplifié suivant :

$$\widetilde{A} = \begin{matrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \mathbf{1} & \mathbf{1} & \mathbf{1} & \mathbf{1} & 0 & 0 & 0 \\ 0 & 0 & 0 & \mathbf{1} & \mathbf{1} & \mathbf{1} & \mathbf{1} & 0 & 0 & 0 \\ 0 & 0 & 0 & \mathbf{1} & \mathbf{1} & \mathbf{1} & \mathbf{1} & 0 & 0 & 0 \\ 0 & 0 & 0 & \mathbf{1} & \mathbf{1} & \mathbf{1} & \mathbf{1} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{matrix} \quad \widetilde{B} = \begin{matrix} \mathbf{1} & \mathbf{1} & \mathbf{1} \\ \mathbf{1} & \mathbf{1} & \mathbf{1} \\ \mathbf{1} & \mathbf{1} & \mathbf{1} \end{matrix}$$

L'érosion de $\widetilde{A}$ par $\widetilde{B}$ donne :

$$(\widetilde{A} \ominus \widetilde{B}) = \begin{matrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \mathbf{1} & \mathbf{1} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \mathbf{1} & \mathbf{1} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{matrix}$$

Dilatation : La dilatation est l'opération inverse de l'érosion.

$$(A \oplus B)_z = \begin{cases} 1 & \text{si } B_z \cap A \neq \varnothing \\ 0 & \text{sinon} \end{cases}$$

Où z est un des pixels de A et B_z est la translation de B au niveau du pixel z. $B_z \cap A \neq \varnothing$ signifiant que B_z et A s'interceptent. Si nous reprenons l'exemple

donné pour l'érosion, la dilatation de $\widetilde{A}$ par $\widetilde{B}$ donne :

$$(\widetilde{A} \oplus \widetilde{B}) = \begin{matrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \mathbf{1} & \mathbf{1} & \mathbf{1} & \mathbf{1} & \mathbf{1} & \mathbf{1} & 0 & 0 \\ 0 & 0 & \mathbf{1} & \mathbf{1} & \mathbf{1} & \mathbf{1} & \mathbf{1} & \mathbf{1} & 0 & 0 \\ 0 & 0 & \mathbf{1} & \mathbf{1} & \mathbf{1} & \mathbf{1} & \mathbf{1} & \mathbf{1} & 0 & 0 \\ 0 & 0 & \mathbf{1} & \mathbf{1} & \mathbf{1} & \mathbf{1} & \mathbf{1} & \mathbf{1} & 0 & 0 \\ 0 & 0 & \mathbf{1} & \mathbf{1} & \mathbf{1} & \mathbf{1} & \mathbf{1} & \mathbf{1} & 0 & 0 \\ 0 & 0 & \mathbf{1} & \mathbf{1} & \mathbf{1} & \mathbf{1} & \mathbf{1} & \mathbf{1} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{matrix}$$

Ouverture : L'ouverture (*opening* en anglais) est une érosion suivie d'une dilatation :

$$(A \circ B) = (A \ominus B) \oplus B$$

Dans l'exemple donné précédemment, le résultat de l'ouverture sera identique à l'image originale : $(\widetilde{A} \circ \widetilde{B}) = \widetilde{A}$. Prenons un autre exemple :

$$\widetilde{C} = \begin{matrix} 0 & 0 & 0 & 0 & 0 & \mathbf{1} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & \mathbf{1} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & \mathbf{1} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \mathbf{1} & \mathbf{1} & \mathbf{1} & \mathbf{1} & 0 & 0 & 0 \\ \mathbf{1} & \mathbf{1} & \mathbf{1} & \mathbf{1} & \mathbf{1} & \mathbf{1} & \mathbf{1} & 0 & 0 & 0 \\ 0 & 0 & 0 & \mathbf{1} & \mathbf{1} & \mathbf{1} & \mathbf{1} & \mathbf{1} & \mathbf{1} & \mathbf{1} \\ 0 & 0 & 0 & \mathbf{1} & \mathbf{1} & \mathbf{1} & \mathbf{1} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \mathbf{1} & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \mathbf{1} & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \mathbf{1} & 0 & 0 & 0 & 0 & 0 \end{matrix} \quad (\widetilde{C} \circ \widetilde{B}) = \begin{matrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \mathbf{1} & \mathbf{1} & \mathbf{1} & \mathbf{1} & 0 & 0 & 0 \\ 0 & 0 & 0 & \mathbf{1} & \mathbf{1} & \mathbf{1} & \mathbf{1} & 0 & 0 & 0 \\ 0 & 0 & 0 & \mathbf{1} & \mathbf{1} & \mathbf{1} & \mathbf{1} & 0 & 0 & 0 \\ 0 & 0 & 0 & \mathbf{1} & \mathbf{1} & \mathbf{1} & \mathbf{1} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{matrix}$$

L'ouverture a pour effet de faire disparaître les zones plus étroites de l'image binaire originale $\widetilde{C}$.

Fermeture : La fermeture (*closing* en anglais) est une dilatation suivie d'une érosion :

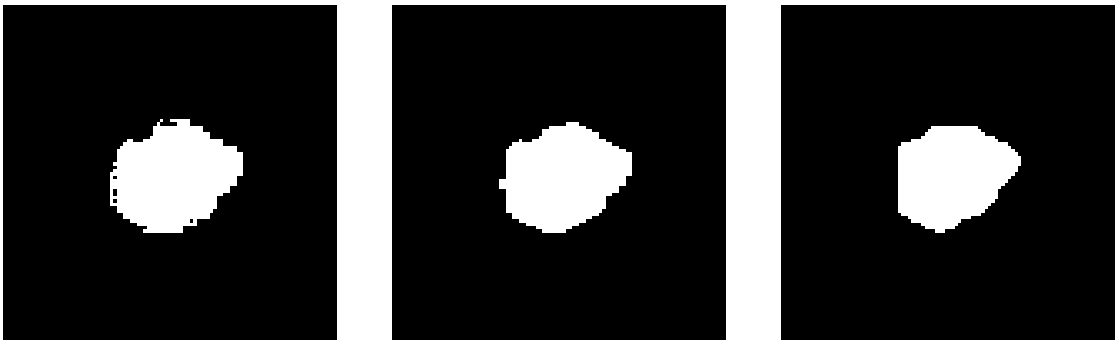
$$(A \bullet B) = (A \oplus B) \ominus B$$

Un exemple d'application de la fermeture est le suivant :

$$\widetilde{D} = \begin{matrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \mathbf{1} & \mathbf{1} & 0 & 0 & \mathbf{1} & \mathbf{1} & 0 & 0 \\ 0 & 0 & \mathbf{1} & \mathbf{1} & 0 & 0 & \mathbf{1} & \mathbf{1} & 0 & 0 \\ 0 & 0 & \mathbf{1} & \mathbf{1} & 0 & 0 & \mathbf{1} & \mathbf{1} & 0 & 0 \\ 0 & 0 & \mathbf{1} & \mathbf{1} & 0 & 0 & \mathbf{1} & \mathbf{1} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{matrix} \quad (\widetilde{D} \circ \widetilde{B}) = \begin{matrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \mathbf{1} & \mathbf{1} & \mathbf{1} & \mathbf{1} & \mathbf{1} & \mathbf{1} & 0 & 0 \\ 0 & 0 & \mathbf{1} & \mathbf{1} & \mathbf{1} & \mathbf{1} & \mathbf{1} & \mathbf{1} & 0 & 0 \\ 0 & 0 & \mathbf{1} & \mathbf{1} & \mathbf{1} & \mathbf{1} & \mathbf{1} & \mathbf{1} & 0 & 0 \\ 0 & 0 & \mathbf{1} & \mathbf{1} & \mathbf{1} & \mathbf{1} & \mathbf{1} & \mathbf{1} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{matrix}$$

La fermeture a pour effet de "fermer" les espaces entre les différentes zones de l'image binaire $\widetilde{D}$.

Utilisation comme filtrage : Sur base d'une image binaire brute venant d'une segmentation de couleur (voir plus loin), une ouverture suivie d'une fermeture permet de filtrer un marqueur afin d'obtenir des bords plus nets. La taille de l'élément structurant B utilisé permet un lissage des bords plus ou moins important. La Figure 2.12 donne une vue avant et après ouverture/fermeture avec une taille W de l'élément structurant de 5 (2.12b) et de 10 (2.12c).



(a) Image binaire brute

(b) Image binaire après ouverture et fermeture ($W = 5$)

(c) Image binaire après ouverture et fermeture ($W = 10$)

FIGURE 2.12

2.2.2 Algorithme de suivi des marqueurs :

Le choix de la couleur et du type de marqueur étant laissé à l'utilisateur, une initialisation manuelle de ceux-ci est nécessaire. Sur base des positions obtenues,

chaque marqueur sera par la suite recherché dans une sous-image. Cette région d'intérêt (ROI) se déplace dans l'image complète en fonction de la position du marqueur.
L'Algorithme 1 présente la procédure suivie pour chaque marqueur :

1 : Pour chaque nouvelle image, la première étape consiste à récupérer la ROI (fonction *GetRoiGom* - Algorithme 10 dans l'Annexe B).

2 : Sur base de l'image de la ROI dont l'espace de couleurs est éventuellement transformé, les pixels susceptibles de faire partie du marqueurs sont obtenus par seuillage (fonction *ThresholdGom* - Algorithme 11 dans l'Annexe B).

3 : Si la taille du marqueur détecté est suffisamment proche de ce qui est recherché, le marqueur est considéré comme trouvé et sa nouvelle position ainsi que l'évolution de sa couleur sont déterminées (fonction *TrackGom* - Algorithme 12 dans l'Annexe B).

(4) : Si le marqueur n'est pas trouvé, une stratégie est mis en place en fonction de la situation. Des itérations sur la même image peuvent être effectuées (voir Algorithme 2).

Algorithm 1 suivi d'un marqueur dans une ROI

Input : Image complète : $ImageOriginal$

```
1:
2:  for toutes les images do
3:      iter_counter = 0
4:      while marqueur pas trouvé  and  iter_counter <= iter_max do
5:          // Récupération de la ROI
6:          ImageROI_RGB = GetRoiGom(ImageOriginal)
7:
8:          // Thresholding de la ROI dans l'espace de couleur choisi
9:          ImageROI_Thresholded = ThresholdGom(ImageROI_RGB)
10:
11:         if marqueur trouvé then
12:             // Calcul de la nouvelle couleur et de la nouvelle position
13:             (x, y), couleur = TrackGom(ImageROI_Thresholded)
14:             iter_counter = 0
15:         else
16:             gestion des marqueurs perdus
17:             iter_counter + +
18:         end if
19:     end while
20: end for
```

1 : Récupération de la ROI

La récupération de la région d'intérêt consiste à créer une sous-image de l'image complète venant de la vidéo. La ROI est prise centrée autour de la dernière position trouvée du marqueur. Cela signifie que la ROI est mobile dans l'image. La largeur et hauteur de cette zone est un paramètre propre à chaque marqueur et pouvant être adapté si besoin (lors de mouvements rapides par exemple). Notons (voir Algorithme 10 dans l'Annexe B) que si un marqueur venait à se rapprocher du bord de l'image complète, la région d'intérêt ne sera plus centrée autour de celui-ci puisqu'elle sera limitée par les bords. La suite du traitement d'un marqueur se fera exclusivement sur la sous-image (celle-ci pouvant être agrandie si le marqueur n'est pas trouvé).

2 : Recherche d'un marqueur

Avant de rechercher le marqueur, un changement d'espace colorimétrique peut être appliqué sur l'image de la ROI. Une ouverture suivi d'une fermeture est appliquée afin d'obtenir des bords plus nets. L'image érodée est récupérée pour évaluer l'évolution de la couleur du marqueur.

La taille du marqueur est définie comme le nombre de pixels non nul que compte l'image segmentée après ouverture et fermeture. Le marqueur est considéré comme trouvé si sa taille $CurentSize$ est suffisamment proche de sa taille d'origine $InitialSize$, c'est-à-dire si sa variation est inférieure à un coefficient $VarSize$:

$$\begin{cases} \text{trouvé} & \text{si } \dfrac{InitialSize}{VarSize} < CurentSize < InitialSize \cdot VarSize \\ \text{non trouvé} & \text{sinon} \end{cases} \tag{2.4}$$

3 : Suivi des marqueurs

Le suivi des marqueurs (Algorithme 12 dans l'Annexe B) consiste à déterminer les positions centrales des marqueurs ainsi que leur variation de couleur. La couleur est obtenue comme la moyenne des couleurs du marqueur segmenté et érodé afin de garder uniquement la couleur intérieure. En effet, il a été observé que les contours d'un marqueur maquillé sur la peau (et même d'une gommette auto-collante) donnent des pixels dont la couleur est un mélange entre la peau et le maquillage. Prendre en compte ces pixels ferait tendre la couleur suivie vers celle de la peau.

Afin d'empêcher des changements trop brusques de la couleur suivie au pas suivant $color_{i,f+1}$, celle-ci est déterminée pas une moyenne mobile entre la couleur moyenne

obtenue sur l'image segmentée érodée et la couleur suivie précédemment :

$$color_{i,f+1} = new_color_i \cdot \alpha + color_{i,f} \cdot (1 - \alpha)$$

Concernant la position, la première étape consiste à déterminer les différents groupes distincts de pixels obtenus dans l'image segmentée filtrée via la détermination des contours. Le groupe le plus proche de la position précédente du marqueur est retenu et la **moyenne de son contour** servira de centre [1]. Pour cette raison, il est préférable de dessiner des taches concaves et les plus circulaires possible. Enfin, une moyenne mobile semblable à celle effectuée sur les valeurs de couleurs peut également être utilisée comme filtrage des plus hautes vitesses.

Un récapitulatif des différentes étapes décrites jusqu'à ici est donné dans la Figure 2.13 : partant de l'image originale (2.13a), une ROI est obtenue (2.13b). L'espace colorimétrie de la sous-image est éventuellement transformé (ici vers du HSV) pour effectuer le *thresholding*. L'effet de l'érosion (qui sert comme zone où calculer la couleur moyenne - 2.13e), ainsi que la dilatation (2.13f) et la fermeture (2.13g) est montré. Le contour de l'image (2.13g) est récupéré et la position moyenne des points qui le compose est utilisée comme centre du marqueur (2.13h).

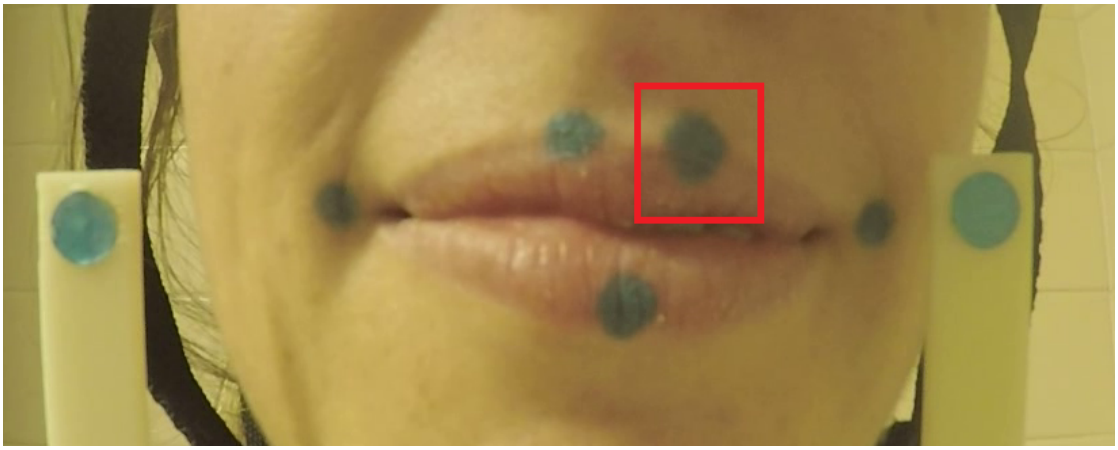
(4) : Procédure en cas de marqueur perdu

Comme nous l'avons vu, un marqueur est considéré perdu si sa taille évolue de trop comparé à la taille à l'initialisation. La perte peut survenir dans trois types de cas :

- le marqueur n'est simplement physiquement plus visible (obstruction, roulement des lèvres importants, ...). Ce cas englobe aussi les moments où le marqueur réellement visible est trop petit pour être considéré comme trouvé.
- le marqueur est faussement considéré comme perdu car ses caractéristiques de couleur ou de position ont brusquement changé (mouvements très rapides, ombre ou éclat de lumière sur le marqueur, ...).
- la zone de couleur suivie ne correspond plus au marqueur initialisé (l'adaptation de la couleur a convergé vers la peau, l'intérieur de la bouche, ...)

Dans les deux premiers cas, la taille du marqueur trouvé est trop petite tandis qu'une taille trop grande signifiera théoriquement que la zone suivie n'est plus bonne. Si la zone suivie n'est plus bonne, la solution la plus simple consiste à revenir à la couleur et à la position initialisée. Ceci devrait en principe permettre, après éventuellement plusieurs images, de retomber sur le marqueur recherché.
Dans les autres cas, jouer sur les seuils de couleurs et la taille de la ROI peut permettre de récupérer un marqueur perdu. Lors des itérations successives sur la

1. Le centre obtenu est bien la moyenne du contour et non le centre géométrique du marqueur qui peut s'en différer légèrement.



(a) Image Originale complète



(b) ROI (c) en HSV (d) Thresholded (e) Érosion (f) Dilatation (g) Fermeture (h) Contour (rouge) et moyenne du contour servant de centre (bleu)

FIGURE 2.13 – Étapes de récupération d'un marqueur.

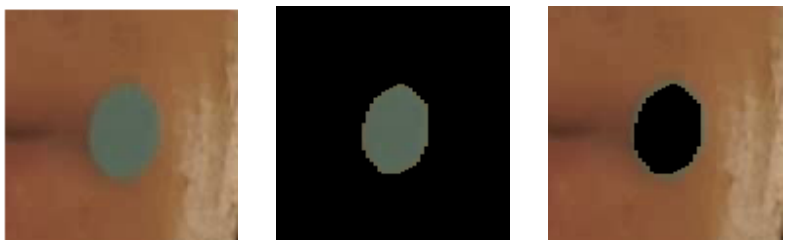
même image, ceux-ci sont augmentés de 10% à chaque fois. Si cela ne fonctionne pas après 3 itérations, les paramètres précédents sont rétablis. Pensons, par exemple, au cas où le marqueur n'est juste pas visible sur l'image : dans ce cas là, c'est en laissant passer quelques images que le marqueur peut être retrouvé.
Si après un nombre not_found_max d'images le marqueur n'est pas retrouvé, la couleur et la position de l'initialisation est d'office ré-utilisée. Si malgré les étapes précédentes, un nombre $not_found_reselection$ d'échecs avec de nouvelles images est atteint, il est demandé à l'utilisateur de re-sélectionner le marqueur.
La procédure en cas de marqueurs perdus est présentée dans son ensemble dans l' Algorithme 2.

Algorithm 2 Procédure en cas de marqueur perdu

```
1:
2: // Première itération sur la même image
3: if iter_counter == 0 then
4:    Enregistrement paramètres de couleur précédents
5:    iter_max = 2 // Définit le nombre d'itérations à faire
6: end if
7:
8: // Si taille trop petite
9: if CurentSize < (InitialSize/VarSize) then
10:   seuil_couleur *= 1.1
11:   (dx, dy) *= 1.1
12:   // Si taille trop grande ou perte trop longue
13: else if CurentSize > (InitialSize · VarSize) or not_found > not_found_max then
14:   Retour au paramètres de couleur et de position de l'initialisation
15:   // Si marqueur non retrouvable
16:   if not_found > not_found_reselection then
17:      Demander re-sélection du marqueur
18:   end if
19: end if
20:
21: // Dernière itération sur la même image
22: if iter_counter == iter_max then
23:   Retour au paramètres de couleur et de position précédents
24: end if
```

2.2.3 Tests de colorimétrie et de marqueurs

Après une série de tests de marqueurs, feutres, maquillages et de gommettes auto-collantes, nous limiterons la présentation des tests colorimétriques à 4 types de marqueurs : 2 couleurs de gommettes auto-colantes rondes ($\varnothing \approx 8\ mm$) vendue en papeterie et 2 couleurs de maquillages (type eye-liner). Ceux-ci sont représentés dans la Figure 2.14. Sur base de l'algorithme d'extraction présenté précédemment ceux-ci ont été extraites puis masquées afin de caractériser non seulement leur couleur, mais également la couleur de la zone les entourant. Notons que toutes les images de cette section ont été prise avec la *GoPro Hero 4 Black* du système LMTS originel.



(a) Gommette bleue.



(b) Gommette verte.



(c) Maquillage bleu foncé.



(d) Maquillage bleu clair.

FIGURE 2.14 – Différents marqueurs étudiés : région du marqueur (gauche), marqueur identifié (centre), région du marqueur avec marqueur masqué (droite).

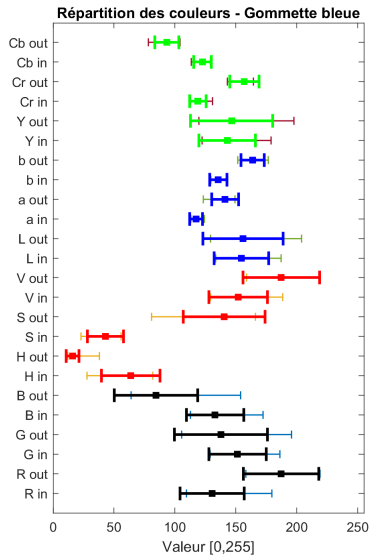
Pour chacun de ces marqueurs, nous pouvons observer les histogrammes de couleurs suivants dans les différents espaces colorimétriques présentés (Annexe C).

Afin d'avoir une vue plus générale sur les variations de couleurs des marqueurs et de leur entourage, l'évolution des valeurs moyennes de couleurs dans les différents espaces colorimétriques présentés ont été représenté Figure 2.15). Dans le but d'être le plus indépendant possible du choix du marqueur utilisé, nous allons analyser les

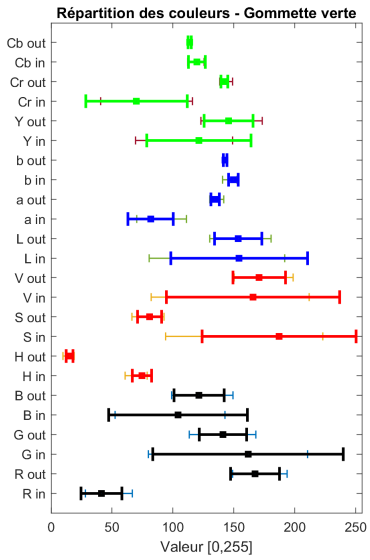
résultats de la Figure 2.15 par espace de couleur :

- **RGB :** Remarquons tout d'abord que les plages de valeurs observées sont très larges : de ce fait, il y a généralement un chevauchement important entre l'intérieur et l'extérieur du marqueur. Contrairement à ce qui pourrait être attendu, la composante rouge est plus discriminante que la composante bleue. L'utilisation d'un seuil de rouge et de bleu fixe ne pourrait vraisemblablement pas fournir de bon résultats.
- **HSV :** Mis à part pour le maquillage bleu foncé, la composante de teinte "H" permet à elle seule de différencier le marqueur de la peau. Si on observe les 4 graphes en même temps, la teinte de la peau a varié dans l'ensemble suivant : $H_{out} \in [0; 80]$, ce qui inclus les teintes de certains marqueurs. La valeur de saturation peut aussi être utilisée pour tous les marqueurs (sauf le maquillage bleu clair).
- **Lab :** Première remarque : l'extraction de la luminescence "L" semble réduire drastiquement les variations des composantes "a" et "b". Ces deux dernières composantes sont discriminantes entre l'intérieur et l'extérieur des gommettes.
- **YCbCr :** Malgré la simplicité de sa transformation (à partir du RGB), les résultats sont plutôt bons et la discrimination efficace.

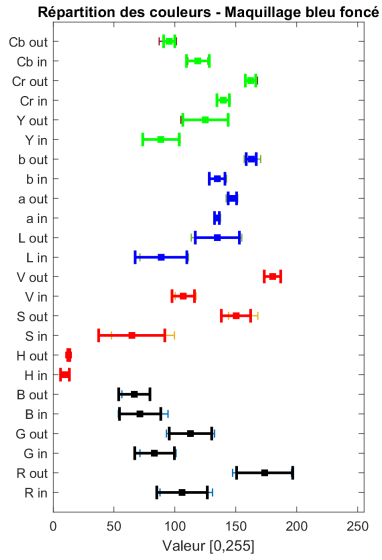
L'objectif étant de ne pas imposer le choix des marqueurs utilisés, et comme son usage est commun, nous avons choisi d'utiliser l'espace HSV par défaut. Ce choix n'est par contre pas fixe et obligatoire, et pourra être modifié par l'utilisateur dans les paramètres présents dans l'interface graphique (présentée dans la section 4.5).



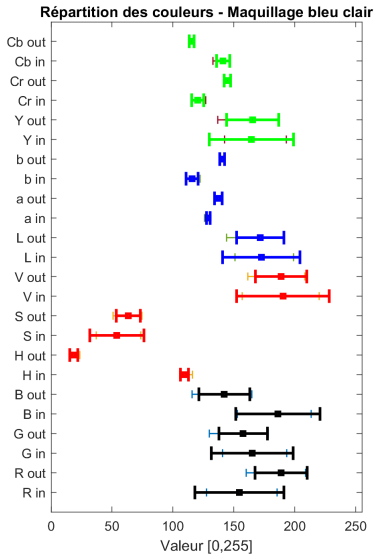
(a) Gommette bleue auto-collante GBGB



(b) Gommette verte auto-collante



(c) Maquillage bleu foncé



(d) Maquillage bleu clair

FIGURE 2.15 – Répartition des couleurs dans les différents espaces proposés pour les différents marqueurs (RGB, HSV, Lab et YCbCr).
En **trait fort** : valeurs comprises entre $\mu - 2 \cdot \sigma$ et $\mu + 2 \cdot \sigma$ où μ est la moyenne des couleurs des pixels formant un marqueur et σ est l'écart-type.
En **trait fin** : valeur comprises entre la moyenne minimale et maximale observée sur un marqueur.

2.3 Tests de l'algorithme

Maintenant que nous avons expliqué dans le détail la façon dont le traitement d'image est effectué, nous pouvons présenter les tests réalisés. Nous nous concentrerons ici sur les tests liés à l'algorithme en tant que tel. L'algorithme ayant été réalisé pour traiter une pléthore de vidéos venant de sources différentes, les tests présentés ici ont été réalisés avec des vidéos de la GOPRO, d'une des caméras du système stéréo présenté chapitre 3 et sur base de données synthétiques. Les tests effectués ont plusieurs objectifs :

— tester le suivi
— tester la fiabilité (% de marqueurs trouvés).
— tester la "récupération" des marqueurs perdus

Sauf indication contraire, les paramètres utilisés sont ceux que nous avons décidé de mettre par défaut dans notre application qui sera présentée chapitre 4, voir Tableau 2.1.

Espace de couleur utilisé	HSV
ΔH	15
ΔS	20
ΔV	20
Variation de couleur	$\alpha = 0.4$
Taille de la ROI	100×100
Variation de la position	pas de moyenne mobile ($\alpha = 1$)
Variation de taille autorisée	$VarSize = 5$
Taille de l'élément structurant pour l'opening/closing	$W = 5$

TABLE 2.1 – Paramètres par défaut utilisés pour le traitement d'images.

2.3.1 Données Réelles

L'utilisation de données faites en situation réelle permet d'obtenir des résultats au plus proche de ce qui sera obtenu lors de l'utilisation par un praticien. Dans le cas qui nous occupe, comme nous l'avons présenté, des tâches de couleurs sont faites au maquillage sur le contour des lèvres d'un patient. Ceci étant, la position réelle des points à suivre n'est à priori pas connue sans l'utilisation d'un algorithme de traitement tel que le nôtre. Cependant, afin de comparer les résultats que nous obtenons à une référence qui peut être jugée probante, un positionnement à la main du centre des marqueurs à suivre a été fait (malgré que l'évaluation manuelle du centre d'une tache soit un source d'erreur importante). N'étant pas

réaliste d'annoter manuellement et plusieurs fois des milliers d'images, deux vidéos représentatives (une venant d'une des caméras du système stéréo, et l'autre de la GOPRO) ont été sélectionnées aléatoirement parmi les acquisitions que nous avons faites.

La première vidéo, où les mouvements y sont rapides, a été annotée manuellement toute les 5 images par plusieurs personnes et à plusieurs reprises différentes. En raison de capacités techniques limitées, la vidéo enregistrée à partir de la caméra stéréo est seulement à $20fps$. La taille des ROI a dés lors été augmentée (les déplacements entre chaque images sont plus grands). Les résultats bruts sont présentés Figure 2.16 où deux sets de paramètres de traitement ont été comparés : le premier est celui par défaut, le second permet une plus grande plage et une variations plus rapide des couleurs ($\Delta H = 30$, $\Delta S = \Delta V = 25$, $\alpha_S = 0.6$). Nous remarquons que dans ce cas-ci, les paramètres par défaut ne sont pas les plus adaptés même si le suivi est dans l'ensemble bon. Notons la variance importante entre les positions annotées manuellement.

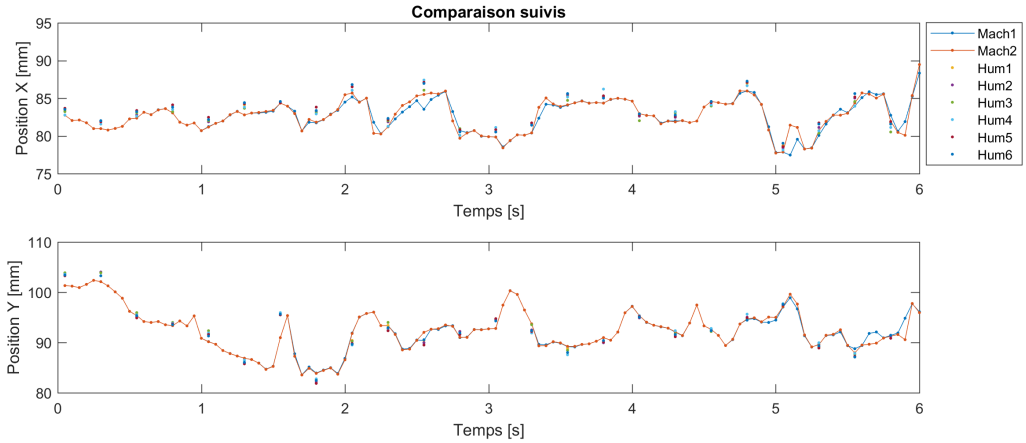


FIGURE 2.16 – Comparaison Humain/machine sur une courte durée. Mach1 = paramètres par défaut, Mach2 = plus grande plage et variations plus rapide de couleur autorisée.

La Figure 2.17 compare les positions moyennes obtenues par le traitement d'image et par les annotations manuelles. Malgré une erreur entre les deux courbes, les mouvements observés sont les mêmes (il faut garder en tête que l'annotation manuelle est très imprécise).

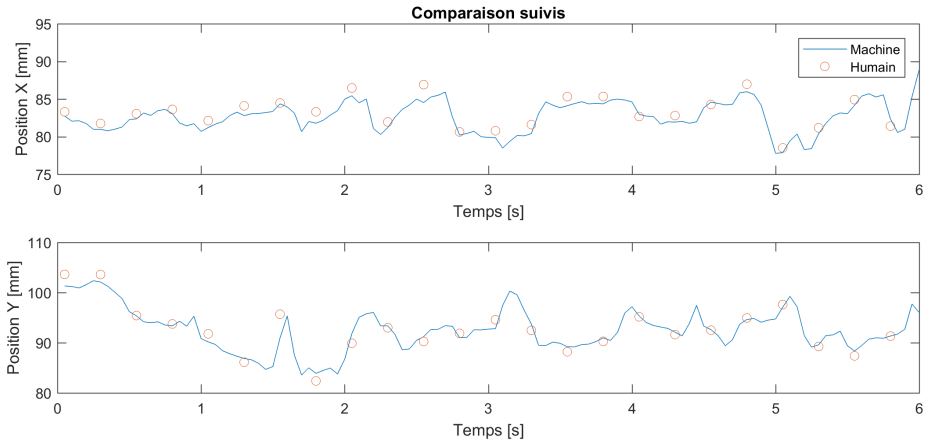


FIGURE 2.17 – Comparaison Humain/machine sur une courte durée

L'erreur observée entre les deux courbes "moyennes" est représentée Figure 2.18. L'erreur de positionnement entre les deux sets de paramètres du traitement est également représentée.

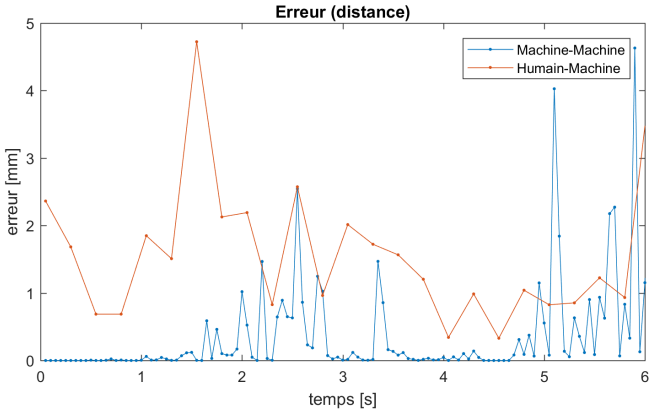


FIGURE 2.18 – Comparaison Humain/machine et machine/machine sur une courte durée. L'erreur moyenne est respectivement de $10.9\,pixels = 1.62mm$ et de $6.55\,pixels = 0.98mm$

La deuxième vidéo sélectionnée a permis de faire un test sur une plus longue durée (Figure 2.19). Les instants auxquels une annotation manuelle est faite sont davantage espacés, mais nous remarquons que le suivi ne présente pas de déviation avec le temps. Plusieurs pics d'une durée supérieure à la seconde sont couverts par les annotations manuelles qui concordent à nouveau bien.

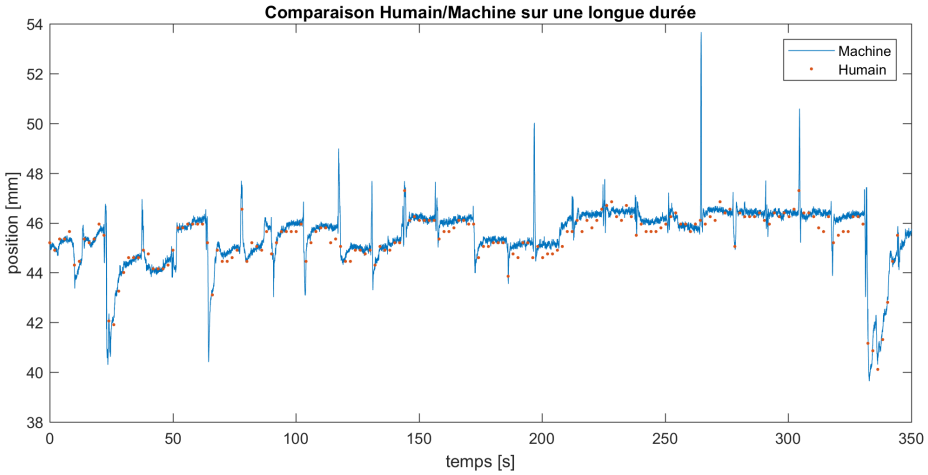


FIGURE 2.19 – Comparaison Humain/machine sur une longue durée (position verticale Y seule).

2.3.2 Données synthétiques

En raison du confinement, nous n'avons pas été en mesure de réaliser des tests de suivi de mouvement sur base d'un système contrôlé, comme cela a été réalisé par Lise Colmant pour son algorithme de suivi d'ellipse [2]. Nous nous sommes donc tournés vers l'utilisation d'images de synthèse pour explorer les limites de notre algorithme. Ce type de test, moins représentatif d'une utilisation réelle, permet tout de même de contrôler entièrement la scène à suivre et de réaliser des tests difficilement faisables réellement.

Création des images Nous avons utilisé *Matlab* pour créer des vidéos d'un ou de plusieurs marqueurs mobiles. Nous nous sommes basés sur les données de couleurs de peau et de marqueurs observés durant les tests de colorimétrie. Chaque pixel formant soit un marqueur, soit la peau, a une couleur choisie aléatoirement dans une certaine plage de couleur. Cette façon de faire, indépendante pour chaque pixel, provoque du "grain" dans l'image qui est enlevé en appliquant un flou Gaussien sur la peau et le marqueur séparément. Un marqueur en forme d'ellipse est ensuite ajouté sur l'image de peau créée. Enfin, pour obtenir un effet plus réaliste de maquillage sur la peau, un flou Gaussien plus intense est appliqué sur l'image obtenue, ce qui rend les bords du marqueur moins nets. Un exemple est montré Figure 2.20.

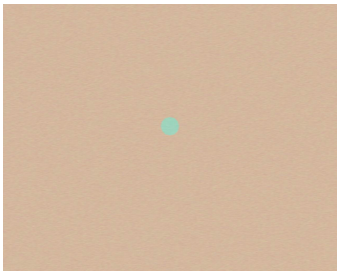


FIGURE 2.20 – Exemple d'image de synthèse.

Notons que tous les paramètres de taille, résolution et de frame-rate ont été choisis identiques aux vidéos prises avec la GOPRO de sorte de pouvoir exprimer les résultats en mm.

Les mouvements donnés aux marqueurs sont systématiquement des oscillations purement verticales, sinusoïdales en fonction du temps et à une fréquence fixe ou variable. Ceci permet d'observer un mouvement plausible tout en permettant de ne se concentrer que sur une position verticale retrouvée.

Vitesse maximale suivie La vitesse maximale des mouvements suivis dépend de plusieurs paramètres du traitement et de la vidéo traitée : lorsqu'un point mobile est filmé, des positions instantanées de celui-ci sont capturées l'une après l'autre. D'une image à la suivante, la position de ce point a changé d'une distance (en pixels)

$$d = v_p \cdot \Delta t = v_p \cdot \frac{1}{fps}$$

Où v_p est la vitesse du point convertie en $pixels/s$ et fps est le nombre d'images capturées par la caméra par seconde. Dans notre algorithme, des problèmes de suivi apparaîtront lorsque le déplacement d d'une image à l'autre est plus grand que la moitié de la taille de ROI utilisée : dans ce cas, le marqueur n'apparaît pas dans la sous-image prise autour de sa position précédente. Prenant en compte un facteur d'agrandissement maximal k_{loss} de la ROI lors de la procédure de recherche des marqueurs perdus, la condition de vitesse maximale du point est :

$$v_p < k_{loss} \cdot \frac{1}{2} \cdot ROI \cdot fps$$

Dans le cas d'une situation "GOPRO" avec un marqueur rond homogène faisant de grands déplacements et avec une ROI de $300\,pixels$ de hauteur, la vitesse maximale atteinte est de $440mm/s$ avant d'observer une erreur de plus de $1mm$ (Figure 2.21). L'erreur avant d'attendre cette vitesse élevée est de l'ordre de $1\,pixel$. Remarquons

qu'une telle vitesse de $440mm/s$ provoquerait un flou important sur le marqueur dans le cas d'une acquisition réelle : la vitesse de suivi de grands déplacements sera donc d'avantage limitée par l'acquisition des images que par le traitement.

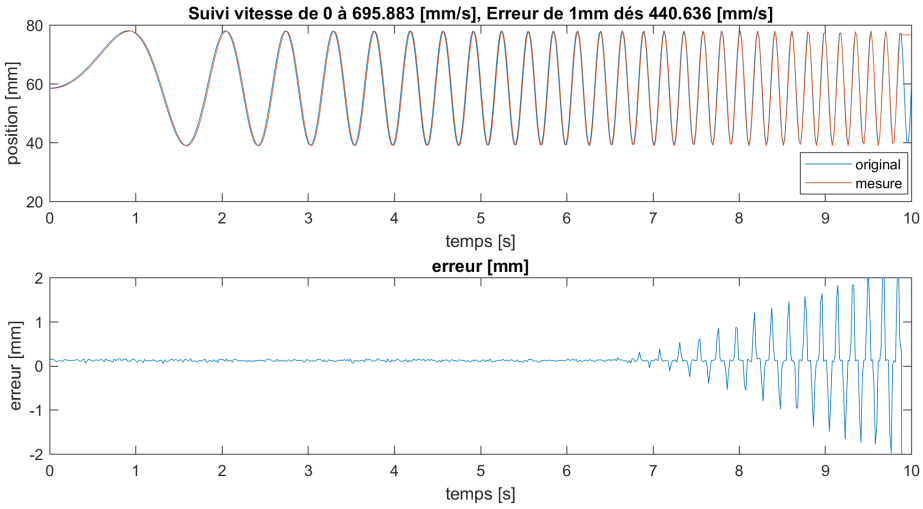


FIGURE 2.21 – suivi d'un marqueur de plus en plus rapide sur une image de synthèse.

Distance minimale entre marqueurs Un problème fréquemment rencontré lors de nos essais est que deux marqueurs proches s'intervertissent lors de certains mouvements spécifiques. Étudier la distance minimale entre les marqueurs est donc quelque chose d'intéressant. La situation étudiée est la suivante : deux marqueurs ont un mouvement sinusoïdal opposé pour représenter un marqueur au-dessus et en-dessous de la bouche qui se rapprochent et s'éloignent lors de la parole. En plus de cela, la position autour de laquelle ils oscillent est de plus en plus rapprochée (comme si les lèvres étaient de plus en plus pincées lors de la parole). Les mouvements originaux ainsi que les résultats sont représentés Figure 2.22.

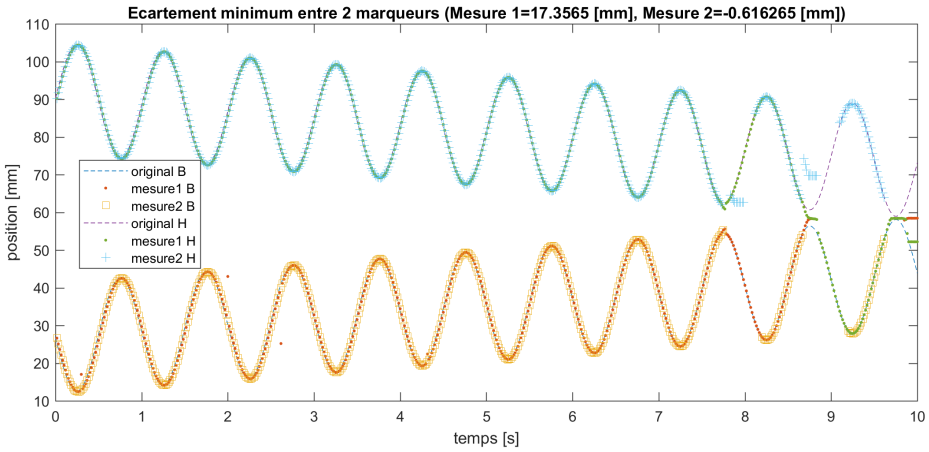


FIGURE 2.22 – Test de distance minimale entre le centre de deux marqueurs de $8mm$ de diamètre. La mesure 1 utilise les paramètres par défaut tandis que la mesure 2 est réalisée quant à elle avec $VarSize = 1.2$. (H = haut, B = bas)

Nous y observons que les marqueurs sont bien suivis jusqu'à ce que la distance séparant les centres soit inférieure à la taille des marqueurs. Dans le cas de la première mesure (paramètres par défaut) nous observons que le marqueur du haut est confondu avec celui du bas vers $9s$. L'erreur de position est de l'ordre de la taille du marqueur à partir d'un écartement (bord à bord) de $17.36mm$. En réduisant la variation de la taille du marqueur autorisée ($VarSize$), il est possible d'éviter de confondre deux marqueurs (tant que ceux-ci restent tous les deux visibles). En choisissant $VarSize < 2$ lorsque les deux marqueurs sont confondus, ceux-ci seront considérés comme perdus (car trop grand). Une fois ceux-ci à nouveaux différentiables, le suivi reprend. Il s'agit en fait ici de permettre une perte momentanée afin de ne plus commettre d'erreurs par la suite.

Une piste d'amélioration future serait d'implémenter un comportement spécial lorsque les centres des marqueurs se rapprochent de trop. Ceci ne sera pas développé dans ce mémoire et comme nous le verrons dans la section 4.2, l'utilisation du multithreading complique la communication nécessaire entre le traitement des différents marqueurs dans le but d'implémenter ce type de réaction.

2.3.3 Taux de réussite du suivi de points en conditions réelles

Notre dispositif a été utilisé afin d'effectuer du suivi du mouvement des lèvres par une mémorante de la faculté de psychologie de l'UCLouvain. Une trentaine de tests en conditions réelles a été produite dans ce cadre pour un total d'environ 10h de

vidéos. Nous en avons donc profité pour traiter celles-ci par tranche de 10 minutes avec notre algorithme et ensuite analyser le taux de réussite du suivi des différents marqueurs. Pour rappel, les marqueurs sont considérés comme ayant été trouvés lorsque leur taille détectée est suffisamment proche de leur taille d'origine et donc que leur couleur n'a pas varié subitement entre deux images consécutives. Il est également à noter que considérer un marqueur comme trouvé ne garantit pas la précision de la mesure de la position de celui-ci.



FIGURE 2.23 – Position des différents marqueurs : 1) Marqueur haut gauche 2) Marqueur haut droit 3) Marqueur bas 4) Commissure gauche 5) Commissure droite

Dans la suite de cette section, nous désignerons les différents marqueurs par leur nom donné dans la Figure 2.23. Sur les tests réalisés en conditions réelles, nous avons traité 4 vidéos de 10 minutes (ayant posé des problèmes de précision de la mesure avec les paramètres par défaut) de trois façons différentes.
La première n'utilise pas la stratégie de recherche de marqueurs perdus et les paramètres par défaut du traitement d'images (voir Table 2.1). L'algorithme attend alors à l'endroit où le marqueur a été perdu que celui-ci repasse au même endroit.
Le deuxième utilise la stratégie de rechercher de marqueurs perdus et les paramètres de traitement par défaut.
Le troisième cas utilise aussi la stratégie de recherche de marqueurs perdus mais avec des paramètres de traitement affinés pour chaque marqueur.

Pour rappel, la stratégie de marqueurs perdus consiste à revenir aux paramètres d'origine , dès la première image de marqueur perdu, lorsque la taille du marqueur détecté est plus grande que le marqueur d'origine. Dans le cas contraire (marqueur détecté trop petit), la zone de recherche du marqueur ainsi que les seuils de couleurs sont agrandis.
Si malgré cela, le marqueur est perdu pendant 10 images de suite, les paramètres reviennent d'office à ceux d'origine. Finalement, si la perte excède 60 images de suite, le traitement s'arrête et l'application demande à l'utilisateur de recliquer sur le marqueur.

	Marqueur haut gauche	Marqueur haut droit	Marqueur bas	Commissure gauche	Commissure droite
1	64.55%	77.35%	77.87%	36.45%	48.52%
2	100%	100%	99.9898%	99.8971%	99.9725%
3	100%	100%	100%	99.9623%	99.9848%

TABLE 2.2 – Pourcentage d'images dans 4 vidéos sur lesquelles le marqueur a été trouvé. 1) Avec les paramètres par défaut et sans la stratégie de recherche de marqueurs perdus 2) Avec les paramètres par défaut et avec la stratégie de recherche des marqueurs perdus 3) Avec des paramètres affinés et avec la stratégie de recherche de marqueurs perdus

Dans le cas où la stratégie de recherche de marqueurs perdus n'est pas appliquée, les marqueurs ayant le taux de réussite du suivi le plus bas sont les marqueurs situés aux commissures des lèvres (voir Table 2.2). Ces taux ainsi que ceux des autres marqueurs augmentent grandement lorsque la stratégie est appliquée, allant même à des taux avoisinant les 100% de réussite. En affinant les paramètres du traitement d'images, ces taux n'augmentent que légèrement [2].

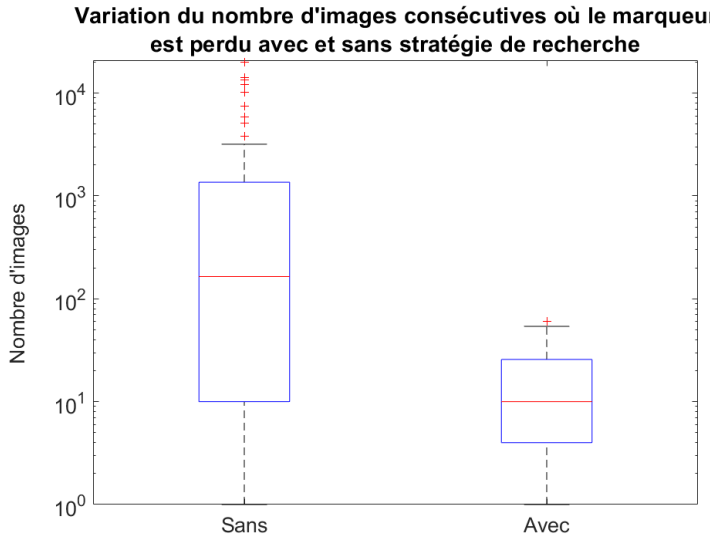


FIGURE 2.24 – Variation du nombre d'images où le marqueur est perdu avant d'être retrouvé en fonction de la stratégie adoptée. Les marqueurs perdus jusqu'à la fin de la vidéo ne sont pas pris en compte.

2. L'intérêt d'affiner les paramètres réside plus dans l'amélioration de la précision de la mesure de la position des marqueurs. En effet, avec les paramètres par défaut, les mesures présentaient des oscillations parasites qui ont été réduites en améliorant les paramètres du traitement d'images.

Dans un second temps, une analyse de la variation du nombre d'images consécutives où un marqueur a été perdu avant d'avoir été retrouvé a été réalisée (voir Figure 2.24). Ces données ne prennent pas en compte les marqueurs perdus jusqu'à la fin de la vidéo (qui représentent 52.18% des marqueurs perdus dans le cas sans stratégie et 7.39% dans le cas avec stratégie) et utilisent les paramètres par défaut du traitement d'image. Dans le cas sans stratégie (voir partie de gauche de la Figure 2.24 [3]), la médiane est à environs 165 images consécutives tandis que dans le cas avec stratégie (voir partie de gauche de la Figure 2.24 [4]), la médiane tombe à 24. Il est à noter que dans ce dernier cas le nombre d'images consécutives avant de retrouver le marqueur n'excède pas 60. En effet, l'application s'arrête automatiquement lorsqu'on dépasse ce seuil pour demander à l'utilisateur de resélectionner le marqueur perdu.

Marqueur haut gauche	**Marqueur haut droit**	**Marqueur bas**	**Commissure gauche**	**Commissure droite**
99.8385%	99.9966%	99.9928%	99.8861%	98.8353%

TABLE 2.3 – Taux de réussite par marqueur

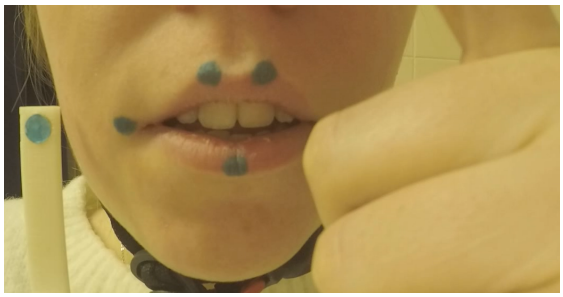


FIGURE 2.25 – Marqueur non visible

La Table 2.3 indique un taux de réussite de 99.71% en moyenne sur l'ensemble des marqueurs lorsque l'on traite l'intégralité des vidéos avec les paramètres par défaut de l'application. Cependant, certaines pertes ne sont pas dues au traitement mais à la non visibilité du marqueur comme représenté sur la Figure 2.25.

Même si cela ne ferait qu'améliorer légèrement le taux de réussite, une piste d'amélioration future serait une modification automatique des paramètres de traitement afin que le meilleur taux de réussite possible soit atteint sans nécessiter l'intervention de l'utilisateur.

3. Basé sur les données obtenues par le traitement des 4 vidéos utilisées précédemment dans cette section.

4. Basé sur les données obtenues par le traitement de l'intégralité des 10h de vidéos des tests en conditions réelles.

2.4 Impact des paramètres de traitement

Comme nous l'avons vu durant ce chapitre, plusieurs paramètres liés au traitement entrent en jeu dans la réussite du suivi. Nous aborderons ici les principaux dans l'optique de permettre au lecteur de comprendre les choix ou modifications à faire dans les quelques cas où les paramètres par défaut ne sont pas appropriés.

Espace de couleur et plage de couleur pour la segmentation Comme nous l'avons vu, l'espace de couleur optimal dépend des marqueurs utilisés. L'utilisation du HSV reste cependant un choix très "passe-partout". La teinte H y est la coordonnée la plus importante : une couleur homogène du marqueur permettra d'utiliser un ΔH petit, mais une variation plus importante peut être utilisée si la tache est hétérogène et que la surface considérée comme le marqueur est nettement inférieure au marqueur réel. La variation de saturation ΔS et de valeur ΔV sont généralement prises plus larges et peuvent être augmentées dans le cas où le marqueur reflète fort la lumière et que les changements de luminosité en cours de vidéo sont importants. Lorsque la peau du patient brille (légère sudation), il peut être utile de diminuer ces valeurs.

Taille des ROI Comme nous l'avons vu, l'utilisation de ROI's a principalement pour objectif de réduire le temps de traitement nécessaire. Cependant, la taille des ROI's limite à la fois la vitesse et la taille des marqueurs qui peuvent être suivis. La taille de la ROI étant définie en pixel, la définition de la caméra utilisée, la position de celle-ci et le frame-rate utilisé auront un impact sur la taille à choisir. Si une scène est capturée à 30 fps à la place de 60 fps, le déplacement entre deux images consécutives du point à suivre sera doublé, ce qui nécessite d'utiliser une ROI deux fois plus large et haute : donc 4 fois plus grande en terme de surface. La taille des sous-images à traiter jouant directement sur le temps de traitement, traiter une vidéo à plus bas frame-rate (fps) peut être plus lent que de traiter des plus petites zones sur une vidéo avec plus d'images.

Dans l'idéal, y a donc un compromis à faire entre le frame-rate et la taille des ROI utilisées pour obtenir un traitement le plus rapide possible.

Variation de la taille autorisée La variation de la taille autorisée ($VarSize$) donne les limites pour lesquelles un marqueur sera considéré comme trouvé. Un $VarSize > 1$ doit être utilisé car lorsqu'un la lèvre sur laquelle le marqueur est dessiné bouge, la surface de celui-ci sur l'image change. Pour s'en convaincre, observons une feuille de papier face à nous, si celle-ci est tournée librement : sa surface visible diminue jusqu'au cas extrême où nous ne voyons plus que sa tranche. Si la feuille est éloignée ou rapprochée, sa taille perçue évolue également. Les

taches faites autour de la peau sont victimes des mêmes effets. En plus de cela, la couleur segmentée peut, lors de changements de luminosité ou lors de mouvements importants, ne plus contenir l'entièreté du marqueur. Une variation importante de la taille autorisée permet de conserver au maximum le suivi quitte à ne pas avoir une position exacte du centre du marqueur en permanence. À contrario, comme nous l'avons vu dans les tests de distance minimale, choisir une faible variation autorisée ($VarSize < 2$) permet d'éviter d'intervertir plusieurs marqueurs très proches. Les interversions étant relativement rares (exception faite de l'utilisation du casque de la GOPRO sur lequel les marqueurs fixes, lorsque le casque est mal serré sur la tête, passent devant les commissures), il est raisonnable de laisser un $VarSize \leq 5$ afin de ne se préoccuper uniquement des pertes "réelles" des marqueurs.

2.5 Conclusions sur l'algorithme développé

L'algorithme qui a été présenté dans ce chapitre est somme toute très générique. Celui-ci a été développé dans l'optique de faire du suivi du mouvement de lèvres, mais le suivi de couleur réalisé peut bien-sûr être utilisé pour bien d'autres applications. Nous avons volontairement gardé un maximum de polyvalence en ne choisissant pas de maquillage en particulier. Des gommettes auto-collantes peuvent d'ailleurs toujours être utilisées. Cette polyvalence implique cependant une initialisation manuelle des positions des marqueurs à suivre. Les retours de l'utilisation du maquillage à la place des gommettes auto-collantes de la part de la mémorante ayant utilisé notre application sont très positifs : après 20 minutes de tests, les patients avaient pour la plupart oublié le maquillage qu'ils avaient sur la bouche et étaient prêts à repartir sans l'enlever.

Les paramètres du traitement d'image sont tous modifiables par l'utilisateur si besoin est (voir section 4.5). Sur plus de 10h de vidéos faites avec la GOPRO et traitées avec notre algorithme, seulement 4 vidéos de 10 minutes nécessitaient des paramètres légèrement modifiés. Le taux de réussite ainsi atteint est de plus de 99.7%. Autrement dit, 3 marqueurs sur 1000 recherchés sont manqués, ce qui dans le cas où 7 marqueurs sont présents (cas des vidéos prises avec la GOPRO) cela fait en moyenne la perte d'un marqueur toutes les 48 images.

Chapitre 3

Dispositif d'acquisition des images

L'algorithme que nous avons implémenté se base sur une scène relativement fixe dont les mouvements visibles sont uniquement ceux des lèvres. Avoir une caméra simplement placée devant le patient (par exemple sur une table) et cadrée sur le visage poserait plusieurs problèmes importants. D'une part, le moindre mouvement de la personne risquerait de le faire sortir du champ, d'autre part, les positions des marqueurs sont désirées dans un repère fixe par rapport au visage. Pour contrer le premier problème, un cadrage plus large serait nécessaire (une image qui contient plus que le visage seul). Ceci étant, une caméra plus onéreuse présentant une plus grande définition est nécessaire pour avoir la même précision (avoir le même nombre de pixels sur la largeur du visage). Le second problème imposerait de déterminer, en plus des positions des lèvres, l'orientation du visage. Ceci est une problématique à part entière qui nécessite des repères fixes par rapport au visage : soit les yeux et le nez [15], soit des marqueurs supplémentaires dont la position doit être indépendante des mouvements liés à la parole.

Le choix a donc été fait, comme pour le LMTS de Lise Colmant [2], de positionner la caméra de manière fixe par rapport au visage grâce à une sorte de casque mis sur la tête du patient. Ce chapitre abordera ce dispositif d'acquisition dans sa globalité : le casque à mettre sur le patient et la caméra faisant partie de celui-ci. La façon de réaliser la stéréovision (afin de pouvoir déterminer des positions 3D) sera également au coeur de ce chapitre tant il conditionne le dispositif d'acquisition réalisé.

3.1 Modélisation et calibration d'une caméra

Avant de s'intéresser au développement du dispositif et la façon de réaliser de la stéréovision, il est intéressant de se pencher sur le principal élément de celui-ci : la caméra. En effet, avant d'utiliser plusieurs capteurs d'images pour réaliser une stéréovision, comprendre comment un tel capteur perçoit une scène est utile. Dans cette section, nous verrons comment un objet du monde réel est capté par la caméra et ensuite converti en image.

Pour rappel, une image numérique peut être représentée comme un tableau (matrice) de pixels ayant chacun une ou des valeurs représentant une luminance ou une couleur.

3.1.1 Le modèle sténopé

Une caméra est généralement modélisée par le modèle sténopé (*pinhole* en anglais). Ce modèle représente la caméra comme la combinaison d'un centre optique et d'un plan image (respectivement l'origine du repère R_c et le plan grisé sur la Figure 3.1). Un point P est projeté sur l'image suivant une droite entre P et le centre optique de la caméra. La projection de P sur l'image correspond au pixel (u, v), soit la u-ième colonne et la v-ième ligne en partant du coin supérieur gauche. L'axe optique, normal au plan image et comprenant le centre optique F_c, définit le centre optique de l'image par son intersection avec celle-ci. Les coordonnées en pixel du centre optique de l'image sont notées (c_x, c_y) et permettent d'établir le repère x, y.

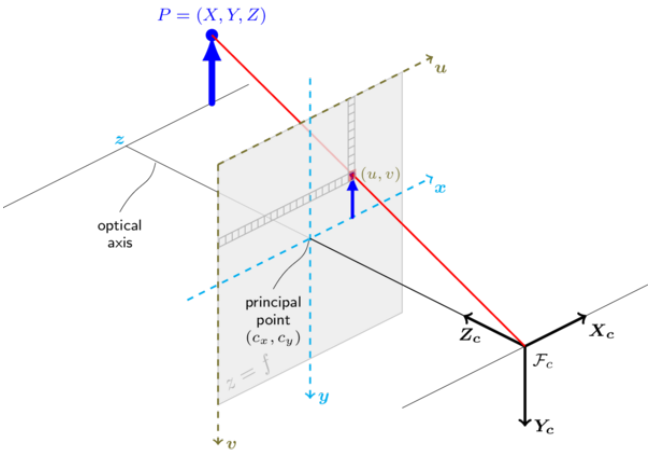


FIGURE 3.1 – Projection d'un point P sur le capteur d'une caméra (plan grisé).[27]

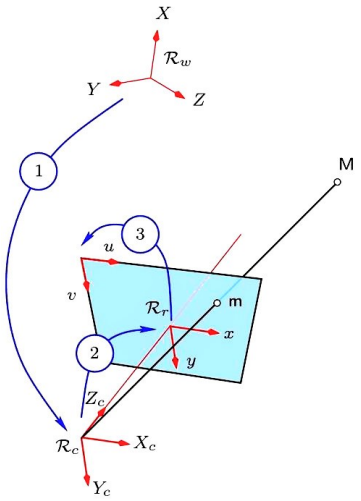


FIGURE 3.2 – Transformations entre les différents repères du modèle sténopé [28]

Le modèle sténopé permet de transformer, en trois étapes successives (Figure 3.2), un point M appartenant au repère monde R_w (coordonnées en X, Y, Z) en un point m appartenant au repère image (coordonnées en x, y). Les coordonnées des différents points seront exprimées en coordonnées homogènes pour simplifier la notation des équations suivantes.

1) Transformation du repère monde R_w vers le repère caméra R_c La première étape consiste à transformer les coordonnées dans le repère quelconque R_w dans un repère R_c fixé sur le centre focal de la caméra et alignées avec celle-ci. De manière générale R_w et R_c sont séparés par une rotation et une translation :

$$\begin{bmatrix} X_c \\ Y_c \\ Z_c \\ 1 \end{bmatrix} = \mathbf{R} \cdot \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} + [\mathbf{t}] = \begin{bmatrix} \mathbf{R} & \mathbf{t} \\ 0^T & 1 \end{bmatrix} \cdot \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

où $\mathbf{R}$ est la matrice de rotation et $\mathbf{t}$ le vecteur de translation pour passer du repère R_w au repère R_c.

2) Transformation du repère caméra R_c vers le repère capteur R_r
Pour passer d'un espace en trois dimensions (X_c, Y_c, Z_c) vers un espace en deux dimensions (x, y), il faut utiliser un projection perspective :

$$z \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \mathbf{P} \cdot \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} = \begin{bmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \cdot \begin{bmatrix} X_c \\ Y_c \\ Z_c \\ 1 \end{bmatrix}$$

où $\mathbf{P}$ est la matrice de projection dans laquelle f la distance focale de la caméra, c'est à dire la distance qui sépare le centre optique F_c du plan image. Notons que x et y sont en fait normés par rapport à $z = Z_c$ (par exemple : $x = \dfrac{f \cdot X_c}{z}$).

3) Transformation du repère capteur R_r vers le repère image R_i
Cette dernière opération consiste à convertir les coordonnées en mètres de l'image en coordonnées en pixels :

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \mathbf{A} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} k_x & 0 & c_x \\ 0 & k_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

où $\mathbf{A}$ est la matrice des paramètres intrinsèques de la caméra composée de :

— k_u et k_v : les facteurs d'agrandissement de l'image
— c_u et c_v : les coordonnées de la projection du centre optique de la caméra sur le plan image

Les deux dernière étapes se résument donc à la transformation suivante :

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \mathbf{C} \cdot \begin{bmatrix} X_c \\ Y_c \\ Z_c \end{bmatrix} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} X_c \\ Y_c \\ Z_c \end{bmatrix} \tag{3.1}$$

Notons que f_x et f_y sont exprimés en pixel et sont généralement différents car les facteurs d'agrandissement de l'image k_x et k_y ne sont pas obligatoirement identiques.

3.1.2 Distorsion de l'image

La présence d'une lentille devant un capteur d'image résulte en une déformation de la projection obtenue. Cet effet est particulièrement visible avec un objectif "grand-angle" comme sur une GOPRO (Figure 3.3).

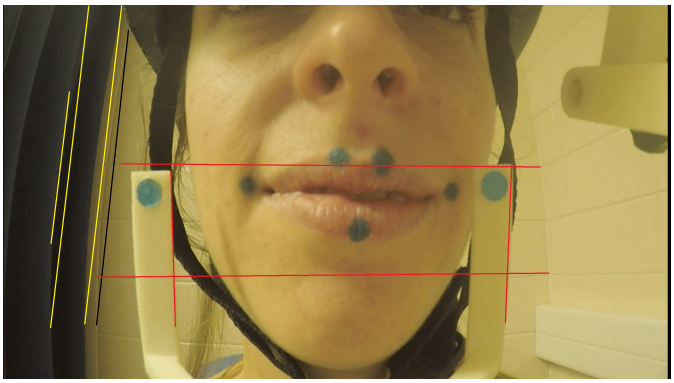


FIGURE 3.3 – Image prise par la GOPRO où l'effet de la distorsion ("en barillet") est visible. Des lignes droites sont tracées à différents endroits de l'image contenant normalement des lignes droites.

La distorsion de la lentille réelle peut être modélisée comme étant principalement une déformation radiale par rapport au centre optique de l'image et éventuellement une déformation tangentielle ([27] - "Camera Calibration and 3D Reconstruction"). Une rectification (x'', y'') peut être faite pour corriger la position (x, y) :

$$x'' = x\frac{1 + k_1 r^2 + k_2 r^4 + k_3 r_6}{1 + k_4 r^2 + k_5 r_4 + k_6 r^6} + 2p_1 xy + p_2(r^2 + 2x^2)$$
$$y'' = y\frac{1 + k_1 r^2 + k_2 r^4 + k_3 r_6}{1 + k_4 r^2 + k_5 r_4 + k_6 r^6} + 2p_2 xy + p_1(r^2 + 2y^2)$$

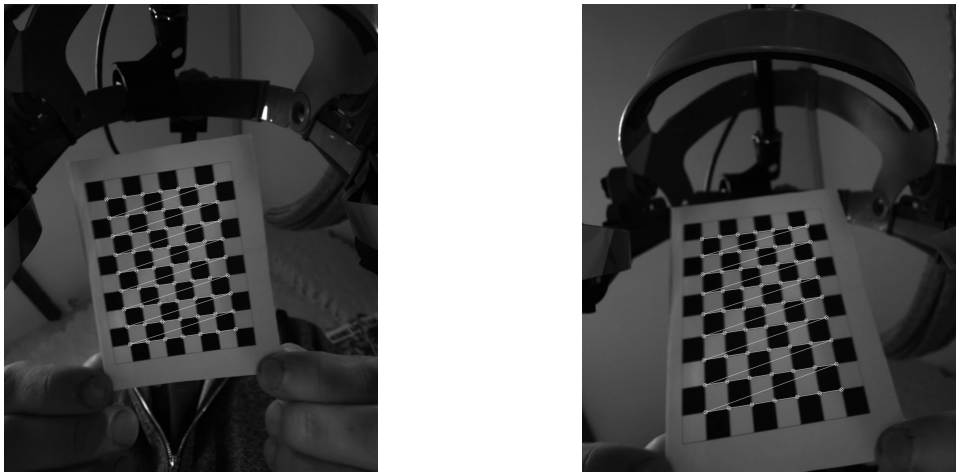


FIGURE 3.4 – Exemple d'images d'échiquier pour la calibration de deux caméra (caméra stéréo). Les coins intérieurs sont ici déjà marqués.

Où k_1, k_2, k_3, k_4, k_5 et k_6 sont les coefficients de distorsion radiale, p_1 et p_2 sont les coefficients de distorsion tangentielle et $r^2 = x^2 + y^2$ est la distance par rapport au centre optique de l'image.

3.1.3 Calibration

La calibration d'une caméra consiste à déterminer d'une part la matrice $\mathbf{C}$ (Équation 3.1) contenant les distances focales f_x et f_y ainsi que les coordonnées du centre optique (c_x, c_y) et d'autre part les coefficients de distorsion présentés précédemment.

Une technique couramment utilisée est l'algorithme de Zhang [29, 30, 31] qui se base sur différentes vues de géométries connues de points selon des orientations inconnues. Cet algorithme est celui implémenté par les fonctions intégrées dans la librairie OpenCV utilisée.

Une vidéo ou une série d'images d'un échiquier sont souvent utilisées car les coins intérieurs y sont facilement retrouvés grâce à l'alternance de cases blanches et noires (voir Figure 3.4). La technique permet aussi de déterminer les positions relatives entre deux caméras (un système stéréo par exemple) sur base de la correspondance entre les points observés dans les différentes vues. Les résultats de la calibration de la caméra stéréo (dont le choix est fait dans la section suivante) sont présentés dans l'Annexe F.

3.2 La stéréovision

Une des améliorations à apporter à l'ancien dispositif est la possibilité de mesurer les mouvements labiaux non seulement dans le plan de la bouche mais bien dans un espace 3D.

L'utilisation d'une caméra "time-of-flight" (qui combine la prise d'images à la mesure de profondeur) a été écartée car la définition de celle-ci est de nos jours très limitée (nombre de points de mesure de la profondeur). L'utilisation d'un LIDAR (sorte de scanner 3D) a également été rejetée car ce type de capteur est sensible et très onéreux.

Le suivi du mouvement dans le plan étant fait par traitement d'images, le choix d'utiliser la stéréovision s'est fait naturellement. La méthode permet d'obtenir une notion de profondeur en ayant au moins deux images de la même scène sous différents angles de vue.

Le fonctionnement de la stéréovision sera tout d'abord abordé avec les relations mathématiques en jeu. La possibilité d'utiliser une seule caméra et des miroirs sera discutée pour finalement présenter le choix fait, d'utiliser une caméra stéréo à deux capteurs.

3.2.1 Principe de base

Le principe de base peut être résumé comme suit : la position d'un point dans un espace tri-dimensionnel requiert trois informations (ses coordonnées (X, Y, Z)). Or, le suivi de la projection de ce point ne donne que deux informations (x_1, y_1). En suivant la projection de ce point depuis un autre angle de vue (connu), nous obtenons des informations (x_2, y_2). Il est dès lors possible de déduire la profondeur d'une scène à partir de deux images de celle-ci grâce à une triangulation.

Le cas de deux caméras parallèles et dans le même plan est développé dans l'Annexe D. Il s'agit, en effet, d'une configuration courante et simple au niveau mathématique qui permet de comprendre l'intérêt de la stéréoscopie.

Cas général

De manière générale, les caméras peuvent être inclinées l'une par rapport à l'autre. Ceci rend cependant les équations entre les coordonnées "caméra" (u_1, v_1, u_2, v_2) et les coordonnées réelles plus complexes que dans le cas d'une stéréovision avec des caméras dans un même plan (caméras parallèles - Annexe D).

La Figure 3.5 représente une configuration géométrique quelconque. Les centres optiques des caméras sont représentés par les points c_1 pour celle de gauche et c_2

pour celle de droite. Un objet, symbolisé par le point P_w, de coordonnées homogènes $(X_w, Y_w, Z_w, 1)$ (de coordonnées $(X_c, Y_c, Z_c, 1)$ dans le repère de la première caméra) a son image projetée sur les plans images de ces caméras, respectivement aux points P_L (de coordonnées homogènes $(x_1, y_1, 1)$) et P_R (de coordonnées homogènes $(x_2, y_2, 1)$).

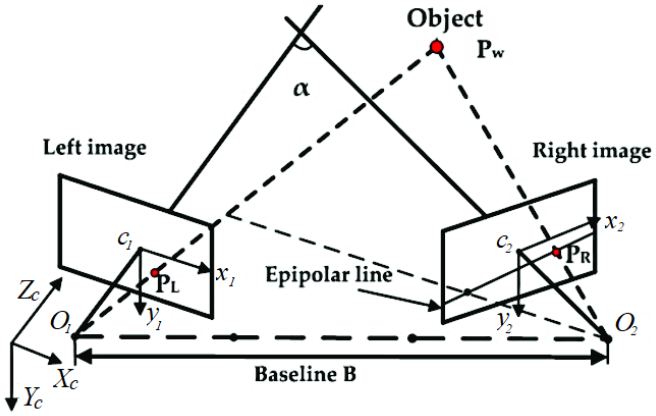


FIGURE 3.5 – Principe de la stéréovision binoculaire (image remaniée de [32])

En utilisant le modèle sténopé, le système à résoudre afin de trianguler la position réelle d'un point est le suivant :

$$\begin{cases} \begin{bmatrix} u_1 \\ v_1 \\ 1 \end{bmatrix} = \mathbf{P}_1 \cdot \begin{bmatrix} X_c \\ Y_c \\ Z_c \\ 1 \end{bmatrix} = \mathbf{K}_1 \cdot \begin{bmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{bmatrix} \\ \begin{bmatrix} u_2 \\ v_2 \\ 1 \end{bmatrix} = \mathbf{P}_2 \cdot \begin{bmatrix} X_c \\ Y_c \\ Z_c \\ 1 \end{bmatrix} = \mathbf{K}_2 \cdot \begin{bmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{bmatrix} \end{cases}$$

Où $\mathbf{K}_i = \mathbf{P}_i \cdot \begin{bmatrix} \mathbf{R}_i & \mathbf{t}_i \\ \mathbf{0}^T & 1 \end{bmatrix}$ est une matrice 3×4. Il est important de remarquer que le système ci-dessus est sur-contraint : trois inconnues sont à déterminer (X_c, Y_c, Z_c) [1] en utilisant quatre équations en u_1, v_1, u_2 et v_2, les quatre informations de position venant des deux images 2D. Dans le cas général où les caméras sont placées de manière quelconque l'une par rapport à l'autre, une technique de résolution par les

1. Les coordonnées (X_w, Y_w, Z_w) sont ensuite directement obtenues connaissant le changement de repère entre "c" et "w" (rotation $\mathbf{R}$ et translation $\mathbf{t}$.

moindres carrés est généralement employée ([33] section 12.2, simplifié dans [34]). Une autre technique consiste à re-projetter les images des deux caméras sur un seul et même plan de sorte de pouvoir les traiter comme dans le cas de caméras co-planaires [35, 36].

3.2.2 Stéréovision à miroirs

La stéréovision à deux caméras pose certains problèmes. Le premier est économique : les caméras représentent le poste de dépense le plus important et donc, un dispositif de stéréovision coûte en principe au moins le double du prix d'un dispositif de monovision. Le deuxième vient de la nécessité de synchroniser les deux caméras. En effet, afin d'obtenir une position (X, Y, Z) précise d'un point en un moment donné, il est nécessaire que les coordonnées (x, y) dans les deux images traitées correspondent au même moment dans le temps.
Afin d'éviter ces désagréments, une solution est d'utiliser un jeu de miroirs afin qu'une seule caméra puisse capter deux points de vue tout en évitant les problèmes de synchronisation [2]. L'utilisation de réflexions et/ou d'autres effets optiques permet d'obtenir l'équivalent d'une ou plusieurs caméras "virtuelles" placées à d'autres endroits que la caméra réelle filmant les miroirs ou prismes.
Dans cette section, nous étudierons dans un premier temps les différentes façons d'utiliser des miroirs afin de faire de la stéréovision. Ensuite, nous discuterons de la possibilité qu'elles soient applicables à notre dispositif, celui-ci devant répondre à des critères de robustesse, de légèreté et de compacité.

2. La vitesse de la lumière étant très grande ($3 \cdot 10^8 [m/s]$), son déplacement peut être considéré comme instantané et des distances à parcourir différentes entre les différentes prises de vue n'auront aucun impact.

Topologies existantes

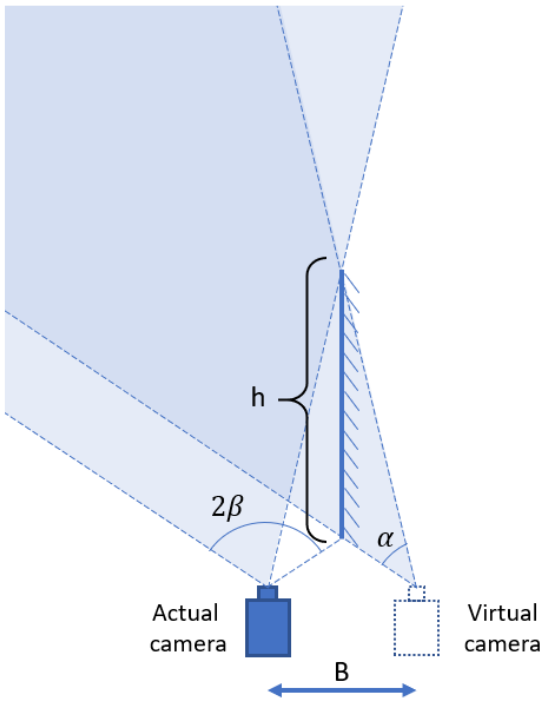


FIGURE 3.6 – Dioptrie à 1 miroir

a) Dioptrie à 1 miroir : Cette solution est composée d'une caméra orientée perpendiculairement par rapport à la normale d'un miroir de hauteur h (voir Figure 3.6). Cette disposition nous permet d'avoir une caméra virtuelle située à une distance B de la caméra réelle, B étant égale à deux fois la distance séparant la caméra réelle du miroir. L'équation ci-dessous nous permet de déterminer le champ de vision α de la caméra virtuelle :

$$\alpha = arctan\left(\frac{2h}{B}\right) - 90° + \beta \tag{3.2}$$

β étant le demi angle du champ de vision de la caméra réelle[37].

b) Dioptrie à biprisme : Cette solution est composée d'une caméra et d'un biprisme de Fersnel.

Comme montré sur la Figure 3.7, le biprisme est constitué de deux faces π_d et π_g inclinées d'un angle α par rapport à son grand côté (sa base) nommé π et faisant face à la scène. Grâce à la réfraction dans celui-ci, un point matériel $\mathbf{X}_p$ sera changé

en deux points virtuels $\mathbf{X}_d$ et $\mathbf{X}_g$ captés respectivement par la partie droite et gauche du photo-senseur de la caméra [38]. Cette situation est équivalente à avoir deux caméras virtuelles comme illustré sur la Figure 3.9.

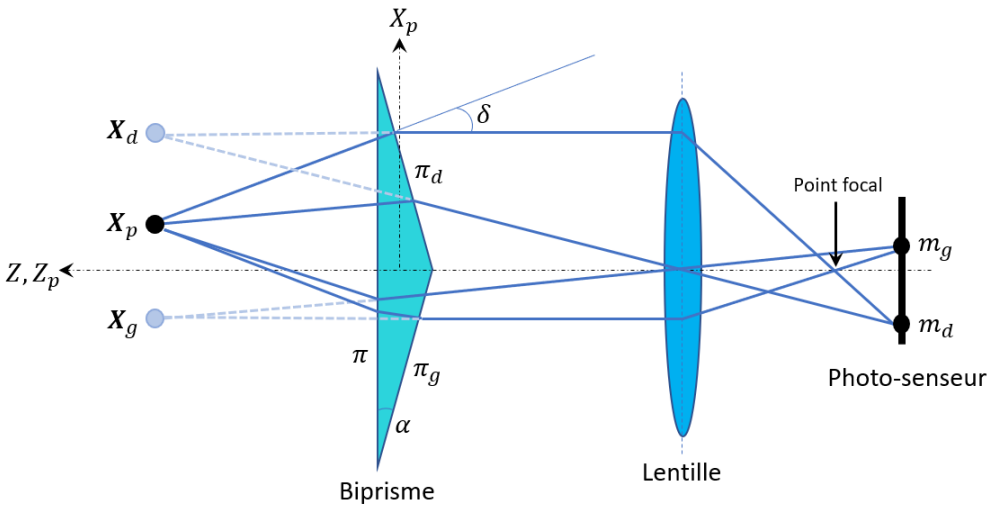


FIGURE 3.7 – Principe du biprisme

L'angle δ est l'angle de déviation (Figure 3.8), l'angle entre le rayon incident et le rayon réfracté [38]. Celui-ci est fonction de l'indice de réfraction du matériau et de l'angle α.

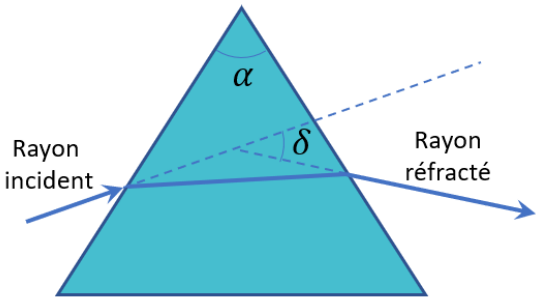


FIGURE 3.8 – Angle de déviation

Avec de simples relations géométriques, il est possible de déterminer une relation entre $\mathbf{X}_p$ et $\mathbf{X}_d$ et $\mathbf{X}_g$:

1. Pour le point virtuel $\mathbf{X}_d = (X_{pd}, Y_{pd}, Z_{pd})$

$$\begin{bmatrix} X_{pd} \\ Y_{pd} \\ Z_{pd} \end{bmatrix} = \begin{bmatrix} 1 & 0 & tan\delta \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X_p \\ Y_p \\ Z_p \end{bmatrix}$$

2. Pour le point virtuel $\mathbf{X}_g = (X_{pg}, Y_{pg}, Z_{pg})$

$$\begin{bmatrix} X_{pg} \\ Y_{pg} \\ Z_{pg} \end{bmatrix} = \begin{bmatrix} 1 & 0 & -tan\delta \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X_p \\ Y_p \\ Z_p \end{bmatrix}$$

Ce qui peut permettre de trouver la disparité D, la distance entre les deux points virtuels :

$$D = X_{pd} - X_{pg} = 2Z_p \cdot tan\delta$$

Notons que la distance B entre les centres optiques des caméras virtuelles (voir Figure 3.9) peut être obtenue par :

$$B = 2t_z \cdot tan\delta \tag{3.3}$$

où t_z est la distance entre le centre optique de la caméra et le biprisme.

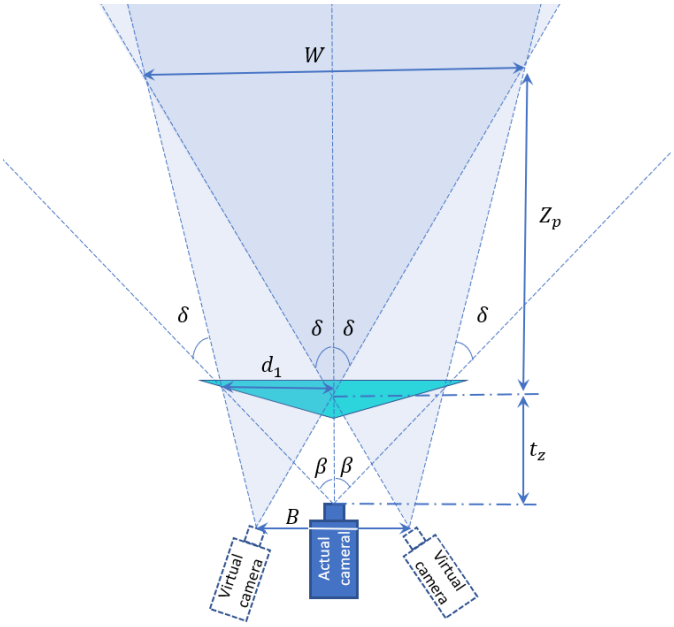


FIGURE 3.9 – Dioptrie à biprisme

Le reste de la géométrie et de l'optique de cette configuration sont décrites par les équations suivantes :

$$\frac{W}{2} = tan\delta \cdot Z_p \tag{3.4}$$

$$\frac{W}{2} \approx d_1 + \frac{Z_p}{tan(90^\circ - \beta + \delta)} \tag{3.5}$$

$$d_1 = tan\beta \cdot t_z \tag{3.6}$$

où W est la largeur maximale de la zone d'intérêt, β la moitié de l'angle du champ de vision de la caméra réelle, Z_p la distance entre le biprisme et l'objet observé et d_1 correspond à la moitié de la largeur du biprisme.

c) Dioptrie à 4 miroirs : Cette configuration est composée en premier lieu de deux miroirs de longueur l_1 formant un angle α avec le plan de symétrie (voir Figure 3.10). La caméra est disposée à une distance h_3 du sommet de ces deux miroirs et a un angle de vue égale à 2β. Ensuite, deux miroirs de longueur l_2 sont disposés à une distance d par rapport au plan de symétrie. Le prolongement de ceux-ci forme un angle γ avec le prolongement des miroirs précédents. La longueur W est la taille maximale de la zone d'intérêt afin de pouvoir réaliser de la stéréovision [39].

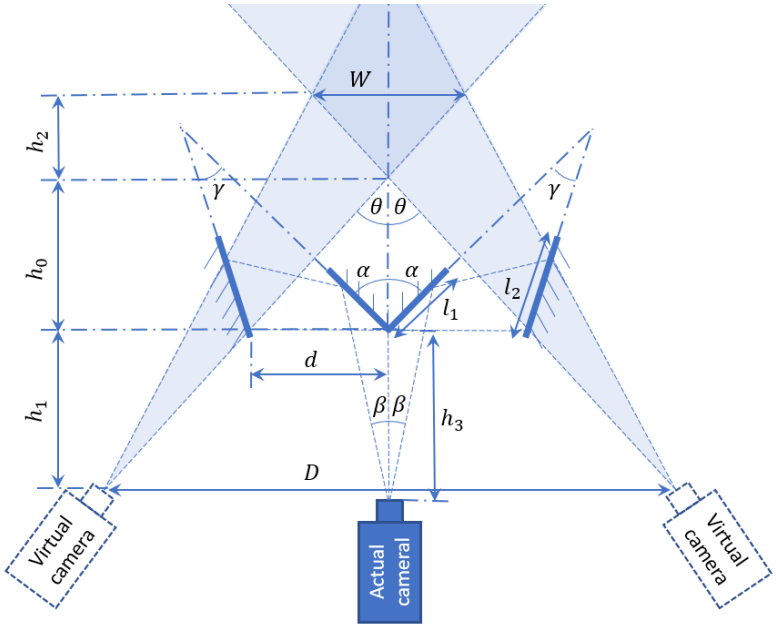


FIGURE 3.10 – Dioptrie à 4 miroirs

En utilisant la loi de Snell-Descartes pour la réflexion, nous obtenons les relations suivantes :

$$\theta = 90° - 2(\alpha - \gamma) \tag{3.7}$$

$$\theta_{max} = 90° - arctan\left(\frac{l_1 sin(90° - \alpha)}{d - l_1 cos(90° - \alpha)}\right) \tag{3.8}$$

$$l_1 = \frac{h_3}{cos\alpha\left(\frac{tan\alpha}{tan\beta} - 1\right)} \tag{3.9}$$

$$\frac{W}{2} = d\left(1 - \frac{tan(90° + \beta - 2\gamma)}{tan\left(2(\alpha - \gamma)\right)}\right) \tag{3.10}$$

$$h_2 = \frac{W/2}{tan\theta} \tag{3.11}$$

3.2.3 Étude de faisabilité

Afin de pouvoir déterminer la faisabilité (ou non) de chacune de ces topologies, il faut établir les objectifs que doit atteindre notre dispositif ainsi que les contraintes qui lui sont imposées. Tout d'abord, la zone qui nous intéresse, celle de la bouche, a des dimensions de $15 \times 15cm^2$. Ensuite, le demi angle du champ de vision β de la caméra considéré est de 40°. Et enfin, les caméras virtuelles doivent être séparées d'au moins $5cm$.

Dioptrie à 1 miroir La taille de l'unique miroir est un facteur limitant l'utilisation de cette topologie. En effet, considérant un champ de vision pour la caméra virtuelle α d'un tiers du champ de vision total de la caméra réelle, supposant ce dernier égal à 80° et en considérant un écartement entre le miroir et la caméra de 2.5cm (et donc B=5cm). On obtient via l'équation 3.2 une hauteur de miroir d'environ $10.5cm$. De telles dimensions nous contraignent en terme de distance entre le dispositif et la bouche du patient.

Dioptrie à biprisme La largeur de la zone que nous souhaitons observer est de $W = 15cm$ et celle-ci est à environ $Z_p = 10cm$ de distance. En se basant sur l'équation 3.4, on obtient un angle de déviation δ égal à 36.87°. Avec les équations 3.3, 3.5 et 3.6 , on trouve $d_1 = 6.95cm$, $t_z = 8.28cm$ et $B = 12.42cm$.

Le paramètre paraissant le plus critique avec cette solution est la dimension du biprisme. En effet, dans cette configuration, nous avons un biprisme d'environ $14cm$ de longueur ! En plus d'avoir un poids important, celui-ci sera difficile à se procurer et onéreux.

De plus, on remarque que la caméra sera à environ $18cm$ de la bouche, ce qui risque de créer un important porte-à-faux qui devra être compensé par l'ajout d'un contrepoids à l'arrière du casque.

Dioptrie à 4 miroirs La compacité est un critère important que notre dispositif doit prendre en compte. Dans le cas de la dioptrie à 4 miroirs, les principaux paramètres (voir Figure 3.10) sont : d, h_0, h_2, h_3, l_1 et l_2. Le but ici est donc de minimiser ces derniers tout en imposant certains paramètres afin de simplifier les calculs.

Commençons d'abord par déterminer α. En se penchant sur l'équation 3.9, pour obtenir l_1 positif, il faut que $\alpha > \beta$.
Ensuite, dans l'équation 3.10 nous remarquons que d est plus petit que $\frac{W}{2}$ uniquement lorsque $\frac{tan(90° + \beta - 2\gamma)}{tan(2(\alpha - \gamma))} < 0$ et donc si $\alpha > \gamma$ (ce qui est bien le cas), $\beta > 2\gamma$.
D'autre part, étant donné que nous voulons filmer proche de la bouche (disons

$h_2 < 10cm$ et $W = 20cm$) et en se basant sur l'équation 3.11, nous obtenons $\theta > 45°$.
En combinant cette dernière expression à l'équation 3.7, nous avons $\alpha - \gamma < 22.5°$.
Avec la lentille choisie pour la caméra, $\beta = 40°$.
On obtient finalement que α doit être compris entre 40° et 42.5°.

En imposant $d = 3cm$, $h_3 = 0.1cm$, $\gamma = 15°$ et considérant $\alpha = 42°$, nous obtenons des miroirs intérieurs d'une longueur $l_1 = 1.8cm$, et des miroirs extérieurs de $l_2 = 7.3cm$.

Conclusion : Quelle que soit la technique utilisée, de grands miroirs (ou un prisme) de l'ordre de 10 cm de côté sont nécessaires. Ceux-ci sont sensibles aux postillons, aux saletés, aux poussières et à la buée alors que le traitement d'images est dépendant de leur qualité de réflexion. Le poids et la fragilité de miroirs en verre seraient contraignant tandis que des miroirs métalliques ou synthétiques seraient extrêmement sensibles aux griffures.

Mis à part le côté pratique de leur encombrement et de la façon de fixer tout cet attirail devant le visage du patient, nous avons réalisé, en faisant un test très simple avec des miroirs, que ceux-ci risqueraient d'apporter un grand inconfort pour le patient, d'autant plus si la caméra est placée au niveau du front du patient et les miroirs en face de la bouche. En effet, dans une telle situation, les miroirs auraient tendance à refléter la lumière de l'éclairage de la pièce (des tubes néons par exemple) directement dans les yeux du sujet. La présence de grand miroirs bloqueraient également le champ de vision de l'utilisateur qui ne pourrait pas lire facilement un texte face à lui. Ce qui est pourtant régulièrement nécessaire lors des tests logopédiques.

Concernant la caméra à utiliser en combinaison des miroirs, sa définition doit au minimum être deux fois plus grande pour obtenir la même précision qu'en mono-vision (pour avoir le même nombre de pixels pour une même distance physique). En effet, l'image obtenue avec la caméra contiendra les deux vues des caméras "virtuelles" ainsi que des zones inutilisées (autour et entre les miroirs par exemple). Ce besoin d'une définition plus importante provoque un sur-coût non négligeable sur l'achat d'une caméra, auquel se rajoute le prix des miroirs et/ou prismes.

Nous nous sommes donc tournés vers l'utilisation de deux caméras telle que présentée ci-après.

3.3 Choix de la caméra

Comme annoncé dans nos objectifs, une des finalités est de permettre un traitement "on the flight" (en direct). Cela n'était pas réalisable avec une caméra du type "GOPRO" qui enregistre le flux vidéo sur une carte mémoire qui peut ensuite, lorsque l'acquisition est terminée, être lue sur un ordinateur pour traiter les images. Afin de pouvoir traiter le flux d'images au cours d'un test sur un patient, l'acquisition doit se faire via l'ordinateur traitant la vidéo et qui doit donc être connecté à une caméra.

Le choix de notre caméra s'est opéré dans l'optique de pouvoir l'utiliser aussi bien dans une configuration monovision que dans une configuration stéréovision. Deux choix ce sont alors présentés à nous : la stéréovison catadioptrique et la stéréovision à deux caméras. Pour des raisons techniques évoquées plus tôt, cette solution à une caméra et miroirs n'a pas été retenue. Nous nous concentrerons dans cette section sur le deuxième cas.

Nous ne fournirons pas un état complet des différentes caméras présentes sur le marché, qui évolue d'ailleurs constamment. Mais nous présenterons deux modèles parfaitement adaptés à notre usage afin d'expliquer le choix fait. L'utilisation d'une caméra "industrielle" permet un fonctionnement robuste et précis à haute vitesse ainsi qu'une plus grande plage de choix de paramètres. L'intérêt principal étant de pouvoir interfacer la caméra directement dans un code de traitement.

Caractéristiques recherchées De manière générale, les caractéristiques minimales recherchées sont de sorte que le nouveau dispositif soit au moins capable de performances équivalentes à celles de la *GOPRO Hero4 Black* utilisée précédemment : une image couleur d'une résolution de **1080p**, un framerate stable de **60 fps** et une masse limitée. La question de la connectique est aussi un point important puisque il faut pouvoir brancher le périphérique sur un ordinateur classique, de préférence sans nécessiter d'interfaces supplémentaires. Pour ces raisons, nous nous sommes tournés vers les caméras industrielles en USB3.0 . L'USB3.0 est un standard apparu en 2008 qui permet un débit de données plus important que l'USB2.0. Certes, les ordinateurs moins récents ne disposent pas de ce type de port mais l'USB3.0 tend à se généraliser. L'USB2.0 ne permettant pas un débit suffisant, l'alternative aurait été la caméra éthernet mais dont l'utilisation est moins commune. Enfin, les caméras devant être positionnées en porte-à faux devant le visage, la masse, l'encombrement mais aussi le prix qui doivent être limités ont été pris en compte dans le choix. Les deux caméras proposées correspondent aux caractéristiques recherchées. La première solution consiste en deux caméras indépendantes. La seconde est un système de caméra stéréo pré-monté.

3.3.1 Deux caméras indépendantes

L'utilisation de deux caméras indépendantes présente des avantages et des inconvénients. D'un côté, elles peuvent être placées de façon libre ce qui permet de n'en utiliser qu'une en cas de mono-vision. De plus, elles ne doivent pas obligatoirement être identiques (la définition peut, par exemple, être inférieure pour l'une des deux si une précision moins grande est désirée sur la profondeur). De l'autre côté, afin de récupérer la position des lèvres en des instants connus, l'acquisition des images doit se faire sensiblement en même temps. Dans le but de garantir cette synchronisation un modèle disposant de PIO (*Input/Output Port*) a été sélectionné. Le modèle *aA1600-60uc - Basler dart* (Figure 3.11) du fabricant *Basler* permet de travailler avec une caméra "maître" et une seconde "esclave".

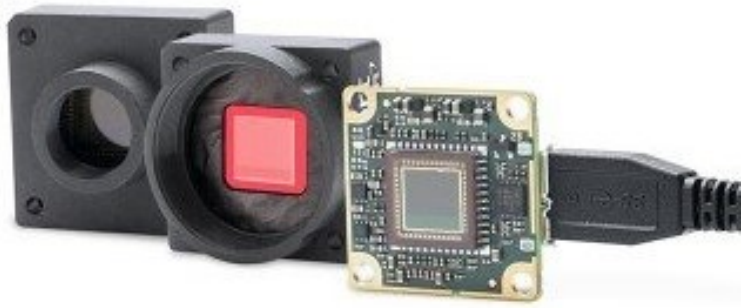


FIGURE 3.11 – daA1600-60uc (S-Mount) - Basler dart

3.3.2 Caméra stéréo pré-montées

L'utilisation d'un système de caméra stéréo pré-monté comme la caméra VEN-134-90U3C-D du fournisseur *GetCameras* est très séduisante dans notre cas. Le système (Figure 3.12) se compose de deux capteurs d'images reliés par des câbles plats et souples à une carte de contrôle (elle-même reliée à un ordinateur). Sans entrer dans les détails de fonctionnement (avec un FPGA), le système se comporte exactement comme une caméra classique sauf que l'image obtenue est en fait l'assemblage des images des deux capteurs.

Pour un peu plus de 300€ HTVA (soit environ l'équivalent de deux caméras chez le même fournisseur) nous avons deux caméras synchronisées ne nécessitant qu'une seule connexion avec un ordinateur (un câble USB3.0) et qui peuvent être placées de manière libre (les câbles entre les capteurs et la carte de contrôle peuvent faire jusqu'à $460mm$ chacun). Nous avons choisi ce produit pour ces différentes raisons et pour la facilité de déploiement qu'elles apportent.

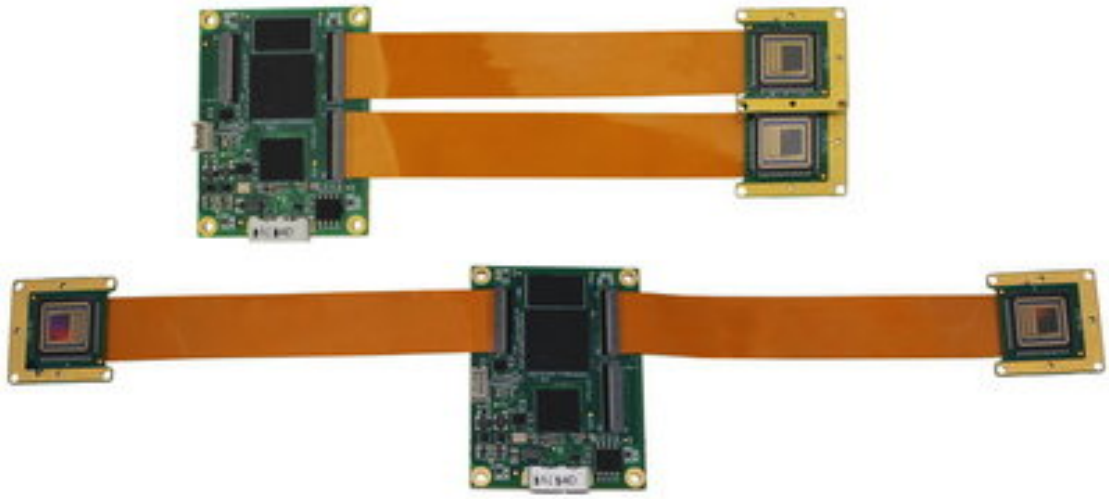


FIGURE 3.12 – Caméra VEN-134-90U3C-D

3.4 Choix des optiques

Une fois les capteurs choisis, le choix des lentilles qui seront placées devant ceux-ci est tout aussi important. Dans notre cas, la caméra devra être placée à environ $10cm$ de la bouche du patient et filmer une zone d'environ $15 \times 15cm^2$. La situation n'impliquant pas de zoom, le choix de la lentille qui passe tout d'abord par le calcul de sa longueur focale peut se faire via une formule simplifiée [40].
La géométrie simplifiée du fonctionnement d'une lentille est illustrée à la Figure 3.13. Par utilisation des triangles semblables, nous obtenons directement la longueur focale f nécessaire :

$$f = \frac{L \cdot Y'}{Y} \tag{3.12}$$

Où L est la distance de travail, Y est la largeur de la zone à observer et Y' est la largeur du capteur d'images.

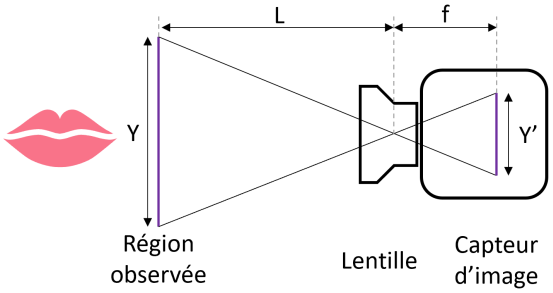


FIGURE 3.13 – Géométrie simplifiée pour le calcul de la longueur focale de la lentille nécessaire.

La caméra sélectionnée a un capteur d'une largeur $Y' = 6.4mm$ ce qui avec $Y = 150mm$ et $L = 100mm$ donne $f = 4.26mm$. Ceci est une première itération car les modèles de lentilles n'existent pas dans toutes les longueurs focales. Désirant pouvoir rapprocher un peu les caméras de la bouche (pour éviter un porte-à-faux trop important du casque), nous avons opté pour une longueur focale de $\mathbf{f = 3.3mm}$ ce qui pour une même largeur observée Y donne une distance de travail de $80mm$.

Une fois une longueur focale sélectionnée, il faut toujours sélectionner le modèle exact de lentille utilisé. Nous avons choisi le modèle *LM12-5MP-03MM-F2.8-2-ND1* chez le même fournisseur que la caméra stéréo : *GetCameras* (Figure 3.14). Ce modèle, bien qu'onéreux (environ 60 €HTVA par pièce) par rapport à d'autres, présente le grand avantage de ne présenter quasiment aucune distorsion, ce qui pour l'élaboration de notre prototype simplifie le traitement des données venant du suivi. Un choix plus économique pourra parfaitement être fait à l'avenir, puisque une lentille par capteur (soit deux lentilles) est nécessaire au système.



FIGURE 3.14 – Lentille de $3.3mm$ sans distorsion du fournisseur *GetCameras* (modèle *LM12-5MP-03MM-F2.8-2-ND1*)

Notons enfin que la lentille choisie a une monture M12 ce qui implique l'utilisation d'un adaptateur pour les installer sur les boîtiers des caméras. Celui-ci rend par

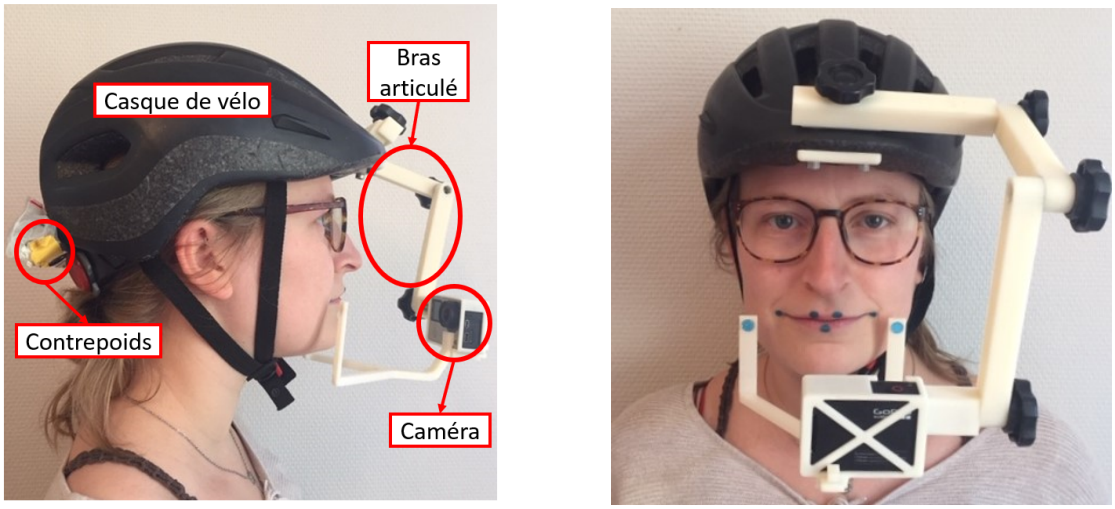
contre la mise au point des caméras plus aisées et une petite vis de pression permet d'immobiliser les lentilles une fois réglées.

3.5 Conception du casque

La caméra stéréo avec ses lentilles étant choisie, nous pouvons maintenant aborder la conception du casque fabriqué pour les positionner devant la bouche du patient. En se basant sur le casque et les quelques essais faits pour le LMTS originel, nous avons tâché de créer un prototype le plus abouti possible.

3.5.1 Casque du LMTS originel

L'ancien dispositif (voir Figure 3.15a) est composé d'un casque de vélo, d'une caméra GoPro Hero4 Black, d'un bras articulé permettant d'ajuster la caméra ainsi que d'un contrepoids qui équilibre le porte-à-faux causé par la caméra. Deux marqueurs fixes sont placés sur un support amené au niveau de la bouche afin d'avoir un repère de distance sur les images obtenues.



(a) Vue de côté (b) Vue de face

FIGURE 3.15 – Ancienne version du casque

Quelles sont les choses à améliorer ?

Le principal reproche fait à l'ancienne version du casque est son inconfort. Celui-ci est principalement imputable au poids total du dispositif. De plus, le poids de la caméra ajouté à celui du bras articulé (imprimé en 3D en PLA plein) entraînant

un basculement du casque vers l'avant, un contrepoids a dû être rajouté à l'arrière de la tête afin de rééquilibrer le dispositif. Portant le poids total du dispositif à 900g, ce qui n'est pas négligeable.

D'un autre côté, le casque de vélo ne fait pas très "sérieux" pour un usage dans un cabinet logopédique ; bien qu'il fut très pratique pour le prototype. De plus, seule une partie (structurelle) de la coque extérieure nous est utile. Le poids excédentaire amené est certes faible (matériaux léger), mais le fait que le casque englobe tout le crâne est incompatible avec certaines coiffures (cheveux très longs, chignons, queue de cheval, ...).

Enfin, la non symétrie du dispositif (voir Figure 3.15b) fait que ce dernier penche légèrement vers la gauche. En glissant doucement sur le crâne du patient durant un test, le marqueur fixe de droite (sur l'image Figure 3.15b) se retrouve parfois devant la commissure gauche.

3.5.2 Cahier des charges pour le nouveau casque

Le nouveau casque doit :

- Être facilement fabricable (sur base d'impressions 3D).
- Avoir un prix raisonnable ($<$ 1000 € caméras comprises).
- Être le plus léger possible.
- Être équilibré (ne pas devoir rajouter de contrepoids).
- Être symétrique.
- Faire en sorte que les caméras soient solidaires de la tête (pour que celles-ci ne bougent pas par rapport à la bouche).
- Permettre de positionner les deux caméras à environ $10cm$ de la bouche et plus ou moins en face de cette dernière.
- Permettre de régler la position des caméras en fonction du visage du patient.
- Comporter un boîtier pour protéger le FPGA des caméras.
- Faire en sorte que le boîtier soit fixé sur le casque en sachant que les nattes (câbles plats) le reliant aux caméras ont une longueur de $460mm$ chacune et sont peu flexibles.
- Faire en sorte que ces nattes ne puissent pas être tordues dans tous les sens ce qui risquerait de les endommager.

3.5.3 Premier prototype du casque

La Figure 3.16 représente la première version de notre casque. Bien que fort différent de la version finale, celui-ci en a posé les bases tout en nous permettant de tester la manière de positionner les deux capteurs d'images l'un par rapport à l'autre.



FIGURE 3.16 – $1^{ère}$ version du casque



FIGURE 3.17 – Casque à visière

Nous avons tout d'abord cherché une alternative au casque de vélo pour la partie du dispositif tenant à la tête. Nous nous sommes alors penchés sur un casque du type "casque de soudeur" dont la partie soutenant la visière remplit la fonction que nous recherchons : permettre de soutenir une légère charge face au visage. Après avoir exploré ce qui était disponible sur le marché, nous avons choisi le casque *Honeywell 1002297* (voir Figure 3.17) dont le prix est de moins de 25€HTVA. Celui-ci offre une série de réglages permettant une tenue ferme et confortable sur la tête ainsi que des points de fixation déjà présents (initialement prévus pour la visière). Nous l'avons utilisé comme base de notre dispositif. Le reste des pièces sont imprimées en 3D (voir Annexe G) et de la visserie.

Utilisation d'un éclairage indépendant Afin d'uniformiser la lumière présente à l'image, améliorer la qualité de celles-ci (des images sombres nécessitent l'application d'un gain pour les éclaircir, ce qui ajoute du bruit/grain) et être indépendant des conditions d'éclairage de l'environnement, nous avons fait le choix d'ajouter un éclairage indépendant sur le casque. Pour ce faire, nous avons conçu un anneau lumineux à fixer autour de l'objectif d'une des caméras. Les LEDs composant celui-ci formant des halos de lumière, nous avons ajouté un couvercle translucide afin diffuser la lumière. Le résultat est visible sur la Figure 3.18.



(a) Situation sans éclairage (b) avec éclairage led (c) avec éclairage led et diffuseur

FIGURE 3.18 – Effet de l'éclairage

Positionnement des capteurs pour la stéréovision Ce premier prototype nous a permis de tester la stéréovision avec des capteurs alignés horizontalement et verticalement. Nous avons également pu tester différentes distances horizontales tandis que le casque de la GOPRO (Figure 3.15a et 3.15b) nous a permis de tester différents angles de vue avec un décalage vertical.

Sans entrer dans le détail, la configuration horizontale a le défaut que les capteurs ne peuvent pas être espacés de plus de quelques centimètres sans que la commissure opposée soit occultée par les mouvements de lèvres. De plus, il est préférable que la vidéo enregistrée soit faite depuis le point de vue faisant face au visage. Ainsi, lors d'une consultation à posteriori de celle-ci par le praticien, les non-symétries des mouvement pourront être observées. C'est pour ces raisons que nous avons préféré positionner les capteurs l'un au dessus de l'autre, et afin de garder la bouche au centre des deux images, un angle a également été choisi sur base d'essais successifs. Les dimensions approximatives sont représentées Figure 3.19.

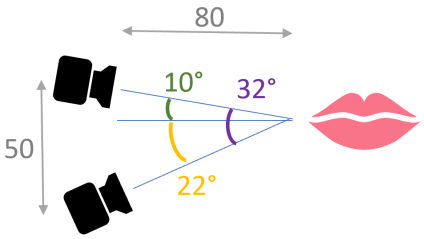


FIGURE 3.19 – Position relative entre les caméras (en noir) par rapport à la bouche

3.5.4 Version finale du casque



FIGURE 3.20 – Dernière version du casque

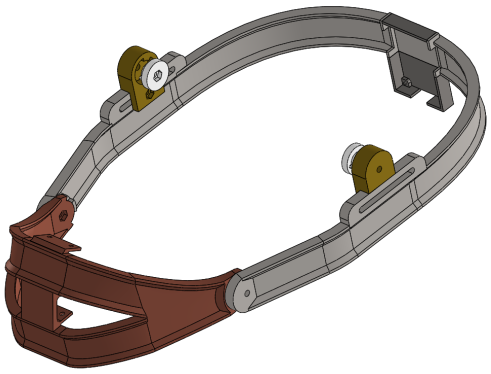


FIGURE 3.21 – Assemblage des éléments imprimés en 3D

La Figure 3.20 est une photo du dernier dispositif construit. Le dispositif complet est ainsi composé du même casque que précédemment (voir Figure 3.17), des deux capteurs d'images reliés par des câbles plats à leur carte de contrôle, le tout fixé ensemble par différentes pièces imprimées en 3D (dont on peut voir l'assemblage sur la Figure 3.21). À cause de l'angle mis entre les caméras, l'anneau lumineux a été remplacé par un ruban LED (voir Figure 3.22) dont le boîtier de piles est fixé avec le circuit de contrôle, faisant office de contre-poids à l'arrière du casque. Il est à noter que le ruban LED n'a pas besoin d'être utilisé avec un diffuseur car la lumière émise est déjà suffisamment diffuse.

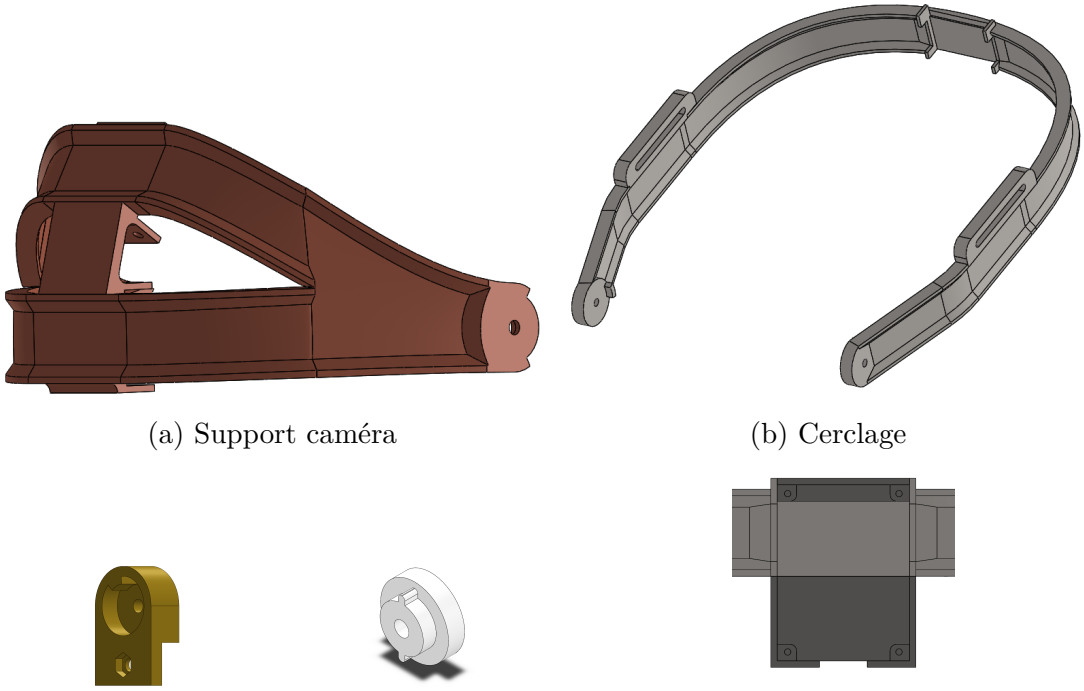


FIGURE 3.22 – Caméras et ruban LED



FIGURE 3.23 – Le casque pèse 520g et est équilibré entre l'avant et l'arrière

Les différents éléments principaux imprimés en 3D sont représentés individuellement



(a) Support caméra (b) Cerclage

(c) Plug casque (d) Vis casque (e) Pièces du boîtier FPGA (en gris foncé)

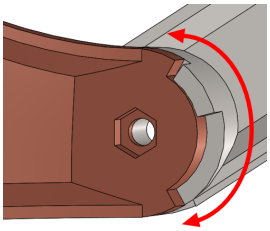
FIGURE 3.24 – Pièces du casque imprimées en 3D (les images ne sont pas à la même échelle)

sur la Figure 3.24. La "vis casque" combinée à une vis à molette et un écrou possède la même utilité que dans la première version de notre dispositif : maintenir le "plug casque" attaché au casque. La version de la Figure G.2 (Annexe G) a été abandonnée car la solidité de la pièce imprimée en 3D laissait à désirer après plusieurs serrages/dé-serrages.

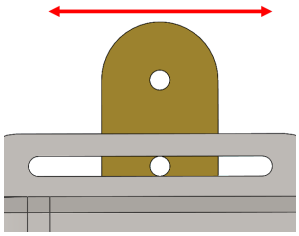
Le "plug" fonctionne également de la même manière que dans la première version de notre dispositif et se fixe au "cerclage" via une autre vis à molette et un écrou. Ce dernier est conçu avec une section en forme de "I" afin de garantir sa rigidité et sa solidité tout en restant léger et permettant le passage des câbles plats des capteurs d'image. Il accueille également le boîtier protégeant le contrôleur de la caméra. Celui-ci est imprimé à part (voir Figure 3.24e) et collé sur le "cerclage" afin de simplifier l'impression de ce dernier.

Enfin, le "support caméra" est vissé au "cerclage" et permet de conserver la position ainsi que l'orientation des caméras l'une par rapport à l'autre.

Notre nouveau dispositif remplit donc les critères que nous avons fixés plus tôt



(a) Articulation entre le support des caméras (en rouge) et le cerclage (en gris)



(b) Glissière entre le cerclage (en gris) et le plug du casque (en jaune)

FIGURE 3.25 – Réglages possibles

(sous-section 3.5.2) : toutes ses pièces sont soit achetables sans besoin de les modifier soit imprimables en 3D, l'ensemble coûte moins de 1000 € (environ 750 € TVA de 21% comprise), pèse environ 520g comme montré sur la Figure 3.23 (soit près de deux fois moins que le casque du LMTS originel), ne nécessite aucun contrepoids supplémentaire et est symétrique. Le casque offre également la possibilité de régler l'orientation des caméras par rapport à la bouche via l'articulation entre le "support caméra" et le "cerclage" (Figure 3.25a). Mais aussi permet de régler la distance entre les caméras et la bouche en fonction du visage du sujet via la glissière entre le "cerclage" et le "plug casque" (Figure 3.25b). Une liste complète des pièces composant le casque est donnée dans l'Annexe H.

Cependant, ce casque n'a pas encore été testé lors d'expériences logopédiques. Certains défauts auxquels nous ne nous sommes pas encore confrontés pourraient alors apparaître.

3.6 Mise en pratique de la stéréoscopie

La configuration des deux capteurs d'images étant définie, intéressons-nous maintenant à la mise en pratique de la stéréoscopie.

3.6.1 Transformation des coordonnées "caméra" vers des coordonnées 3D

Comme nous l'avons vu précédemment, le choix d'orienter les caméras l'une par rapport à l'autre rend les équations entre les coordonnées "caméra" (u_1, v_1, u_2, v_2) et les coordonnées réelles plus complexes que dans le cas d'une stéréovision avec des caméras parallèles dans le même plan.
Nous allons ici nous concentrer sur la disposition particulière des caméras du LMTS 2.0 pour déterminer géométriquement la transformation. La position relative entre

celles-ci sera considérée comme étant une translation $T_Y \cdot \overrightarrow{Y} + T_Z \cdot \overrightarrow{Z}$ et une rotation $\theta \cdot \overrightarrow{X}$, comme prévu dans leur placement. Un désalignement selon $\overrightarrow{X}$ ainsi que d'autres rotations provenant d'un placement imparfait ne sont donc pas pris en compte. Un schéma de la géométrie étudié, vu de côté est présenté Figure 3.26. La caméra 1 est la caméra supérieure. C'est elle qui est utilisée comme repère pour exprimer la position 3D d'un point (X, Y, Z). Autrement dit, $X_1 = X$, $Y_1 = Y$ et $Z_1 = Z$.

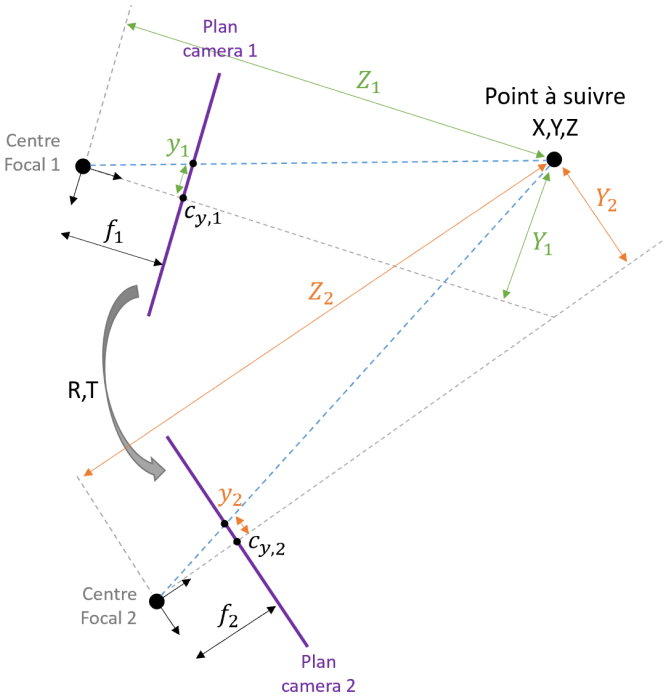


FIGURE 3.26 – Vue latérale

La règle des triangles semblables permet d'écrire les relations suivantes :

$$\frac{f_{x,1}}{Z_1} = \frac{x_1}{X_1} \tag{3.13}$$

$$\frac{f_{y,1}}{Z_1} = \frac{y_1}{Y_1} \tag{3.14}$$

$$\frac{f_{y,2}}{Z_2} = \frac{y_2}{Y_2} \tag{3.15}$$

La disposition des caméras l'une au dessus de l'autre donne théoriquement :

$$x_1 = x_2 \;\rightarrow\; u_1 = u_2 \text{ Si } c_{x,1} = c_{x,2}$$

La rotation autour de l'axe $\overrightarrow{X}$ reviens à :

$$\begin{pmatrix} X_2 \\ Y_2 \\ Z_2 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & cos\theta & sin\theta \\ 0 & -sin\theta & cos\theta \end{pmatrix} \cdot \begin{pmatrix} X_1 \\ Y_1 - T_Y \\ Z_1 - T_Z \end{pmatrix}$$

L'Équation 3.15 peut donc s'écrire :

$$\frac{f_{y,2}}{-sin\theta(Y_1 + T_Y) + cos\theta(Z_1 + T_Z)} = \frac{y_2}{cos\theta(Y_1 + T_Y) + sin\theta(Z_1 + T_Z)}$$

$$f_{y,2}cos\theta(Y_1 - T_Y) + f_{y,2}sin\theta(Z_1 - T_Z) = -y_2 sin\theta(Y_1 - T_Y) + y_2 cos\theta(Z_1 - T_Z)$$

En remplaçant Y_1 par $y_1 \frac{Z_1}{f_{y,1}}$ (3.15) et en isolant les termes en Z_1 on obtient :

$$y_1 Z_1 \frac{f_{y,2}}{f_{y,1}} cos\theta + Z_1 f_{y,2} sin\theta + y_1 y_2 \frac{Z_1}{f_{y,1}} sin\theta - y_2 Z_1 cos\theta$$
$$= f_{y,2} T_Y cos\theta + f_{y,2} T_Z sin\theta + y_2 T_Y sin\theta - y_2 T_Z cos\theta$$

Nous sommes donc en mesure de retrouver la position d'un point dont on connaît les coordonnées dans chaque image (u_1, v_1) et (u_2, v_2) :

$$x_1 = u_1 - c_{x,1}$$
$$y_1 = v_1 - c_{y,1}$$
$$x_2 = u_2 - c_{x,2}$$
$$y_2 = v_2 - c_{y,2}$$

$$Z = \frac{f_{y,2} T_Y cos\theta + f_{y,2} T_Z sin\theta + y_2 T_Y sin\theta - y_2 T_Z cos\theta}{y_1 \frac{f_{y,2}}{f_{y,1}} cos\theta + f_{y,2} sin\theta + \frac{y_1 y_2}{f_{y,1}} sin\theta - y_2 cos\theta} \tag{3.16}$$

$$X = x_1 \frac{Z}{f_{x,1}} \tag{3.17}$$

$$Y = y_1 \frac{Z}{f_{y,1}} \tag{3.18}$$

La première caméra (celle supérieure) est prise comme référence pour le calcul de la position réelle. Notons que tout comme pour le cas des caméras parallèles, les coefficients $\frac{Z}{f_{x,1}}$ et $\frac{Z}{f_{y,1}}$ jouent le rôle de facteurs de conversion d'une distance en pixels vers une distance en mm.

3.6.2 Tests de stéréoscopie

Les tests suivants sont effectués avec les caméras positionnées dans le casque du LMTS 2.0. Plutôt que de filmer des lèvres, nous allons ici filmer une règle en bois avec des pastilles de couleurs collées dessus (Figure 3.27). Celle-ci sont séparées par une distance connue ce qui permet d'évaluer la qualité des coordonnées 3D calculées. Les pastilles de couleurs sont suivies grâce à l'application de traitement d'images que nous avons développée et la position de celles-ci dans chaque image est récupérée. Le calcul des coordonnées réelles se fait via une fonction Matlab parfaitement transposable un C++ par la suite.

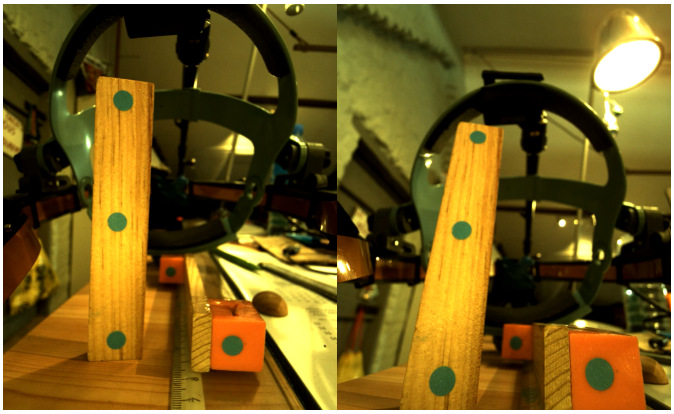


FIGURE 3.27 – Exemple de vue servant de test.

La calibration des deux caméras nous a donné les longueurs focales et la position des centres optiques sur les deux images obtenues (voir Annexe F). La matrice de rotation et la translation obtenue n'ont cependant pas été retenues à cause de l'erreur causée sur le calcul de la profondeur comme nous allons le voir. À la place, les distances et l'angle théoriquement présent entre les caméras seront utilisés.

Mesure de profondeur Dans un premier temps, la mesure de profondeur d'un seul point est étudiée. Comme nous l'avons vu dans la section précédente, c'est la profondeur mesurée qui donne les facteurs de conversion vers une distance réelle. La profondeur mesurée via la stéréoscopie est comparée à la profondeur réelle à laquelle se trouve le point dans le repère de la caméra supérieure. Une règle graduée est fixée solidairement aux caméras durant tout le test et permet la mesure de la distance réelle. Le bout de bois sur lequel est posé le marqueur est déplacé le long de la règle graduée tous les $10mm$ de $30mm$ à $200mm$. Pour chaque position (statique), plusieurs centaines d'images sont traitées et une moyenne de la position en pixels obtenue sur chaque image est utilisée dans l'Équation 3.16.

Les profondeurs obtenues sont présentées sur la Figure 3.28 dans le graphe de gauche, l'erreur relative est présentée à droite. Les résultats sont montrés pour les transformations obtenues avec la calibration (vert) et pour les transformations théoriques (rouge).

Remarquant que l'erreur relative est pratiquement linéaire dans la plage de profondeur qui nous intéresse ($40mm$ à $120mm$), nous pouvons approximer l'erreur relative par une droite et ainsi compenser celle-ci. L'erreur relative observée est au voisinage de $80mm$ est :

$$\xi_Z \approx -\frac{Z - 80}{6} \, [\%]$$

En appliquant la compensation de celle-ci sur la profondeur :

$$Z_{rect} = Z \cdot \left(1 - \frac{\xi_Z}{100}\right)$$

Nous obtenons les courbes bleues (Figure 3.28).

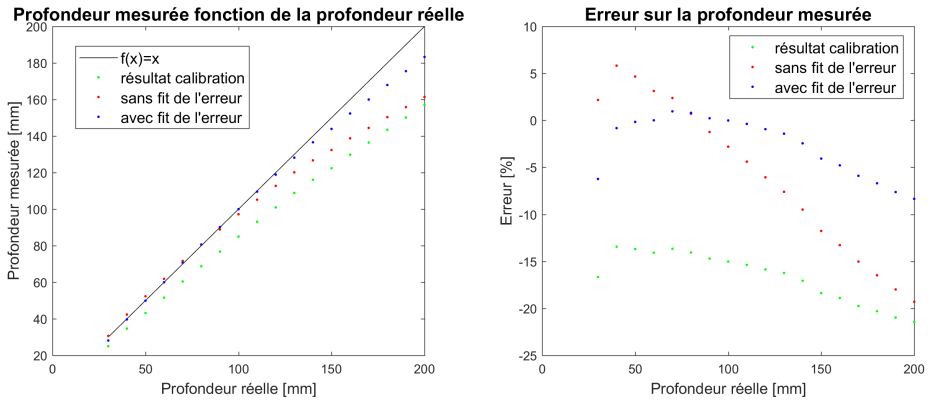


FIGURE 3.28 – Profondeur mesurée et erreur relative sur celle-ci par rapport à la profondeur réelle.

L'erreur résiduelle est inférieure à 1% dans la plage d'intérêt (de $40mm$ à $120mm$) et varie moins même à une plus grande distance (Figure 3.29). Notons que de la même manière que pour la profondeur, un fit de l'erreur (linéaire) sur les positions horizontales et verticales a été fait. [3]

3. Le fit de l'erreur en X et Y donne :

$$X_{rect} = X \cdot \left(1 + \frac{Z_{rect} - 190}{12.5 \cdot Z_{rect}}\right)[\%]$$

$$Y_{rect} = Y \cdot \left(1 + \frac{Z_{rect} - 150}{16.7 \cdot Z_{rect}}\right)[\%]$$

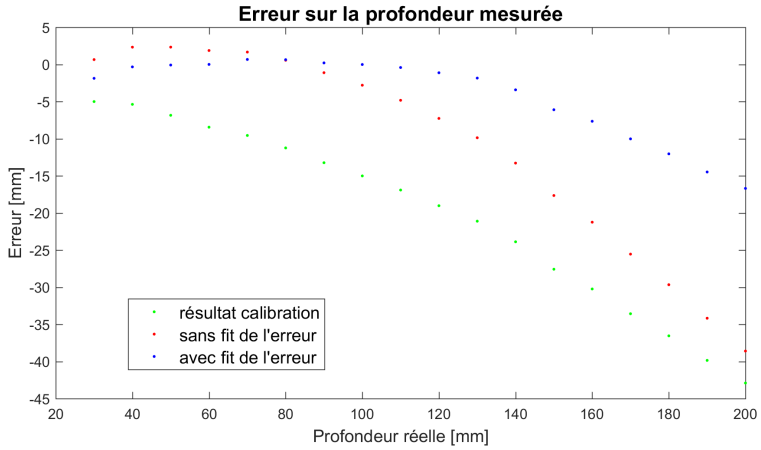


FIGURE 3.29

Mesure de distances Intéressons-nous maintenant au suivi de deux marqueurs de couleurs dans l'image. Grâce aux positions (X, Y, Z) des deux points, nous pouvons calculer la distance qui les séparent :

$$d = \sqrt{(X_1 - X_2)^2 + (Y_1 - Y_2)^2 + (Z_1 - Z_2)^2}$$

Et comparer celle-ci à la distance réelle entre les centres des marqueurs ($50mm$ ou $100mm$).

La variation de l'erreur sur la distance mesurée est illustrée Figure 3.30 pour deux acquisitions différentes (une pour chaque distance entre les marqueurs) qui sont traitées avec les trois modèles présentés. Les résultats confirment à nouveau que la translation $\mathbf{t}$ et la rotation $\mathbf{R}$ obtenues avec la calibration ne sont pas optimaux pour obtenir de bons résultats. Nous pouvons également observer l'effet positif de la compensation de l'erreur proposée, avec des moyennes de l'erreur tendant vers 0, mais surtout avec une variance qui est réduite.

La Figure 3.31 montre les mêmes tests en fonction de la profondeur (Z) mesurée. Les cas où les marqueurs sont placés verticalement et horizontalement l'un par rapport à l'autre sont séparés. L'erreur dans les deux cas étant relativement linéaire, le fit de l'erreur effectué en X et en Y permet de diminuer l'erreur statique (offset) et de gain. Les résultats obtenus montrent une erreur limité à 5% dans toute la plage de mesure (courbes bleues).

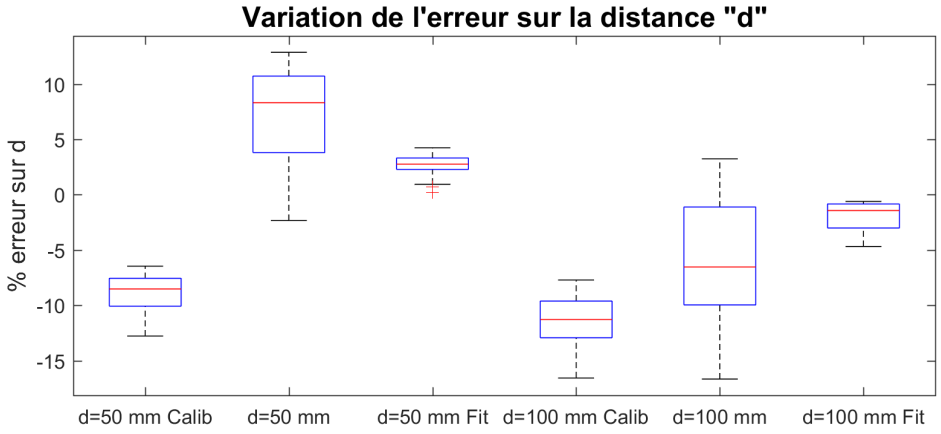


FIGURE 3.30 – Variation de l'erreur sur la mesure de la distance entre deux marqueurs séparés de $50mm$ ou de $100mm$. Les deux situations sont traitées avec les trois modèles présentés : *Calib* pour la transformation sur base des résultats de la calibration, le modèle basé sur les distances théoriques entre les caméras et *Fit* pour le modèle avec compensation de l'erreur.

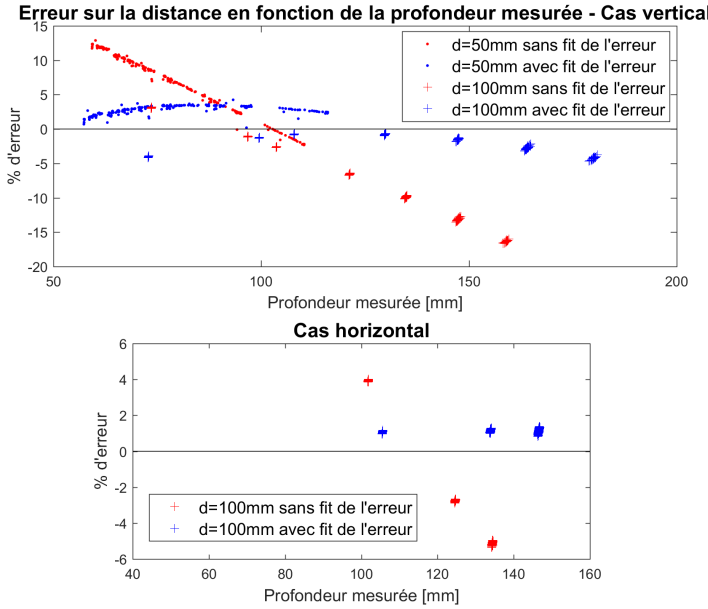


FIGURE 3.31 – La variance de l'erreur est réduite grâce à une compensation simple de l'erreur.

Non idéalités Les erreurs de profondeur et de distance mesurées sont causées par deux raisons principales : d'une part les hypothèses simplificatrices faites dans

l'élaboration de notre modèle et d'autre part l'utilisation des résultats imprécis de la calibration des caméras (voir Annexe F). C'est pour cette dernière raison, et plus précisément l'utilisation de longueurs focales quelque peu erronées, que nous avons observé une erreur variant linéairement avec la profondeur.

La transformation que nos avons utilisée (Équations 3.16 à 3.18) fait l'hypothèse que les caméras sont parfaitement alignées verticalement et donc que $D_x = x_1 - x_2 \approx 0$. Cela n'est cependant pas parfaitement le cas, et sur les deux tests présentés ci-dessus nous observons une différence moyenne de -27.82 pixels. Pour chacune des images des tests, nous avons transformé la disparité D_x en mm en la multipliant par le coefficient $\frac{Z}{f_{x,1}}$. La variation obtenue est illustrée Figure 3.32. Notons que les valeurs obtenues ne correspondent pas à une erreur sur la position des points suivis, mais plutôt la différence qui serait obtenue si la caméra inférieure était prise comme référence.

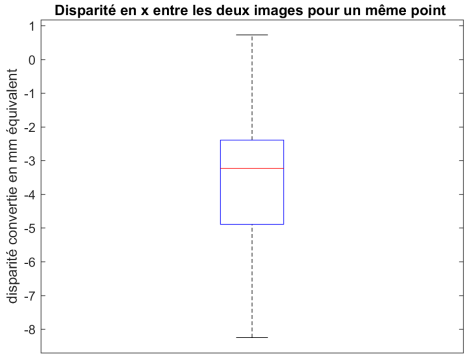


FIGURE 3.32 – La disparité $D_x = x_1 - x_2$ est ici multipliée par le coefficient $\frac{Z}{f_{x,1}}$ pour obtenir une valeur en mm.

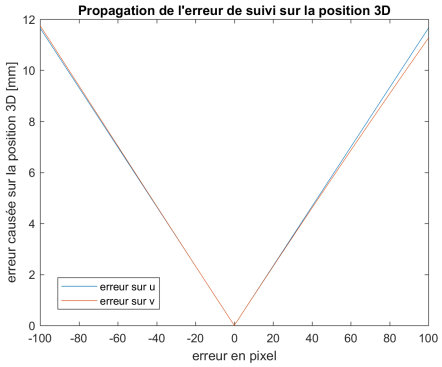


FIGURE 3.33 – Effet d'une erreur de position (u, v) dans les images sur la position 3D calculée. La pente maximale vaut environ $0.116\left[\frac{mm}{pix}\right]$ pour une erreur en u ou sur v.

Remarquons qu'une erreur lors du suivi d'un point dans une des images se propagera sur le calcul de la position 3D du point. Ceci est illustré Figure 3.33.

3.7 Conclusions sur le dispositif final

En raison des conditions particulières que le Covid-19 a imposé, le dispositif final n'a pas pu être testé suffisamment. Des essais sur quelques personnes différentes

ont tout de même été faits. La taille du cerclage faisant le tour de la tête (limitée par la longueur des câbles de la caméra stéréo) peut poser des difficultés pour positionner le casque sur la tête d'un patient ayant une tête assez large et portant des lunettes. Le boîtier du circuit de contrôle de la caméra et le câble usb qui le relie à un ordinateur peuvent également être gênants si le patient est assis sur un siège avec un repose-tête.

De manière générale, les performances obtenues pour la stéréovision - bien que modestes - semblent suffisantes pour l'application qui nous occupe. Une perspective d'amélioration future reste cependant d'explorer des techniques de triangulation plus complexes et donnant probablement des résultats plus justes. Faciliter la calibration (ce y compris un éventuel fit de l'erreur) est à prévoir pour qu'une personne sans qualification particulière puisse, après avoir monté le dispositif, le rendre pleinement utilisable.

Chapitre 4

Intégration : LMTS Studio 2.0

Ce chapitre fait en quelque-sorte le lien entre les deux chapitres précédents : la conception de notre application Windows intègre l'algorithme présenté, l'interface de la caméra stéréo choisie et fait le lien avec l'utilisateur via une interface graphique. Le développement de notre logiciel que nous appellerons *LMTS Studio 2.0* y est donc décrit. La parallélisation du traitement d'images, utilisée afin de rencontrer notre objectif de traitement en temps réel y est particulièrement détaillée. Le principe est de séparer le programme en processus indépendant (threads) pouvant fonctionner en même temps. Ce point constitue, comme nous le verrons, un véritable challenge dans notre situation tant les différentes composantes sont inter-connectées. Ensuite, l'interface avec la caméra stéréo et l'enregistrement du son seront abordés. Enfin, l'interface graphique sera présentée (section 4.5) brièvement. Toutes ses fonctionnalités et composantes seront développées dans l' section J.2.

Notons que l'aboutissement des différents points de ce chapitre constitue notre principale réalisation.

4.1 Langage et bibliothèques utilisées :

Afin d'atteindre notre objectif d'une vitesse de traitement élevé, le choix de coder l'intégralité de notre application en C++ a été fait. L'utilisation de ce langage très répandu a permis d'utiliser deux outils primordiaux pour le développement de notre application : OpenCV pour le traitement d'images et Qt pour la création de l'interface graphique. Afin d'utiliser aisément ces différents outils ensemble, nous avons utilisé l'environnement de développement (IDE) Visual Studio.

Notons que les deux outils (détaillés ci-dessous) permettent le développement multi-plateforme (Windows, Linux, macOS, android). L'utilisation d'une implémentation

"Windows" des Sémaphores (voir plus loin) ainsi que l'interfaçage de la caméra stéréo nous a cependant restreint à ce système d'exploitation.

OpenCV OpenCV est une bibliothèque de traitement d'images dont l'usage est très répandu et libre (sous licence BSD). Multi-plateforme et multi-langage, OpenCV dispose d'un très grand nombre de fonctions permettant par exemple : de modifier l'espace colorimétrique d'une image, de la segmenter, de calibrer une caméra stéréo, ... De plus, la librairie "OpenCV Contrib", non utilisée dans notre cas, contient davantage de fonctions et algorithmes si nécessaire.

Qt : Qt est une infrastructure de développement complet permettant de développer une application (API) multi-plateforme. Développé par *The Qt Company*, il offre des outils de création d'interfaces graphiques. Ceux-ci étant assez faciles à prendre en main, nous avons décidé de les utiliser pour la création de notre application.

L'utilisation de Qt et des outils qui s'y rapportent est gratuit sous licience LGPL (GNU Lesser General Public License) : l'application que nous avons créée l'est donc également, ce qui convient à l'utilisation actuelle qui en est faite.

Qt VS Tool : Qt VS Tool est un outil développé par *The Qt Company* permettant d'intégrer les outils de développement Qt dans le logiciel de développement C++ Microsoft Visual Studio.

4.2 Multithreading :

Obtenir un fonctionnement "temps réel" nécessite d'utiliser un maximum les ressources disponibles sur l'ordinateur utilisé afin de traiter le flux d'images, si possible, à la même vitesse que la cadence d'aquisition des images.

Pratiquement tous les ordinateurs modernes ont un processeur multi-coeur. Chaque coeur est une unité de calcul qui peut exécuter différentes opérations (à la base de tout programme informatique) en parallèle des autres coeurs.

Afin d'accélérer l'exécution d'un programme, il est donc commun de permettre l'utilisation de l'entièreté des coeurs en séparant le code en différents processus pouvant être exécutés simultanément. Cette technique est appelée parallélisation ou *multithreading* en anglais. [41, 42]

Lorsque l'on fait du multitheading, l'idéal est d'avoir des processus (des *threads*) les plus indépendants possible. L'indépendance s'exprime ici par rapport aux ressources utilisées (emplacement mémoire, outils, périphériques, ...). Si par exemple un thread

A nécessite d'utiliser le résultat x d'un autre thread B, rien en l'état ne garantit que B a calculé et stocké son résultat au moment ou A va lire x en mémoire.

Une indépendance compète entre thread n'est cependant pas toujours possible. Prenons notre application comme exemple : Pour chaque image à traiter, il y a potentiellement N marqueurs qui ont été initialisés et qui sont donc à récupérer dans des sous-images. De manière séquentielle, l'algorithme reviendrait à :

1 Récupérer l'image à traiter

2 Rechercher le marqueur 1 dans sa sous-image.

3 Rechercher le marqueur 2.

...

2+N Rechercher le marqueur N.

3+N Afficher et enregistrer résultats.

Le temps total nécessaire pour effectuer ces opérations est donc (de manière théorique) :

$$t_{\text{traitement}} = t_{\text{lecture}} + t_{\text{Affichage}} + N \cdot t_{\text{ROI}}$$

Où t_{lecture} et $t_{\text{Affichage}}$ sont respectivement le temps pour récupérer l'image à traiter (d'un fichier ou d'une caméra) et le temps nécessaire pour afficher les résultats et les enregistrer. t_{ROI} est le temps pour rechercher 1 marqueur dans sa sous-image (ROI). Nous remarquons directement que plus le nombre de marqueurs à rechercher N est grand, plus le traitement sera long.

Nous remarquons cependant que les opérations sur les sous-images sont indépendantes entre-elles et pourraient être effectuées en parallèle. C'est à dire créer un thread pour chaque marqueur. Ainsi, si nous avons 2 coeurs du processeur de l'ordinateur à disposition [1] :

1 Récupérer l'image à traiter

2 Rechercher le marqueur 1 et 2.

...

2+N/2 Rechercher le marqueur N-1 et N.

3+N/2 Afficher et enregistrer résultats.

Le temps de traitement est cette fois réduit en fonction du nombre de thread pouvant être exécutés simultanément.

$$t_{\text{traitement}} = t_{\text{lecture}} + t_{\text{Affichage}} + \frac{N}{\#\text{ coeurs}} \cdot t_{\text{ROI}}$$

1. Dans la pratique, les choses sont plus complexes que cela : c'est le système d'exploitation (ex : Windows) qui répartit les threads entre les différentes ressources en fonction et ce pour tous les programmes en cours.

En séparant le code en processus pratiquement indépendants, nous pouvons donc traiter chaque image d'une vidéo plus rapidement et donc traiter une vidéo complète en moins de temps (ou traiter les images d'une caméra en temps réel). Cependant, la façon de séparer les tâches que nous venons d'aborder peut être encore améliorée. Les opérations de récupération des images, d'affichage et d'écriture des résultats sont relativement coûteuses en terme de nombre d'opérations à exécuter pour le processeur et donc de temps. Il serait donc intéressant de pouvoir également exploiter le multithreading pour ces opérations. Par exemple, pouvoir afficher les résultats de l'image précédente pendant que les positions sur l'image suivante sont calculées. L'objectif est d'atteindre un temps de traitement qui dans l'idéal serait :

$$t_{\text{traitement}} = MAX\Big(t_{\text{lecture}}\ ,\ t_{\text{Affichage}}\ ,\ \frac{N}{\#\ \text{coeurs}} \cdot t_{\text{ROI}}\Big)$$

Il est évident que dans une telle situation, les différents threads ne sont plus indépendants les uns des autres : l'affichage des résultats a, par exemple, besoin de toutes les positions des marqueurs trouvés sur la même image. Comment s'assurer que toutes les positions en mémoire correspondent bien à la même image et qu'un des marqueurs n'a pas été actualisé depuis plusieurs images ? Ou alors, comment s'assurer qu'une valeur n'est pas en train d'être lue par un thread exactement au moment où un autre est en train de la modifier ? Tous ces problèmes provoquent une incertitude sur le fonctionnement désiré de notre programme.

Différents outils permettent cependant un fonctionnement sûr lorsque différents threads doivent communiquer ensemble. Nous allons les expliquer de manière simplifiée avant de présenter la manière dont ils ont été utilisés pour bâtir notre programme.

4.2.1 Outils de communication entre threads

Signal Un signal est utilisé pour la communication entre objets. Lorsqu'un signal est émis, la fonction qui y est liée est exécutée immédiatement. Celui-ci peut contenir des données (une image par exemple). Nous les utiliserons essentiellement pour ce qui concerne l'interface graphique, notamment pour changer l'affichage de l'image ou d'informations à l'écran. Un signal est aussi utilisé pour demander une re-sélection d'un marqueur depuis le code de traitement.

Mutex Un mutex est un indicateur sur une ressource pour dire si elle est ou pas occupée (*locked*). Si nous comparons la ressource (un emplacement mémoire, un périphérique, ...) à une pièce dans un bâtiment, le mutex serait une pancarte libre/occupé accrochée sur la porte d'entrée. Lorsqu'un processus a besoin de la

ressource, il vérifie d'abord le mutex : si celui-ci n'est pas verrouillé (la pancarte dit "Libre"), alors il *lock* celui-ci (la pancarte est retournée sur "Occupé"). Une fois que le processus en a fini avec la ressource, il *release* le mutex (la pancarte dit à nouveau "Libre"). Si lorsque le processus vérifie le mutex, celui-ci est déjà "*lock*", alors il attend.
Dans le cas de l'accès à une valeur dans la mémoire partagée entre différents processus, nous pouvons résumé la procédure par [43, 42] :

- Vérifier le statut du mutex
- si libre
  - le processus *lock* le mutex
  - la valeur en mémoire est lue ou modifiée
  - le processus *release* le mutex
- si verrouillé
  - le processus recommence au premier point

Sémaphore La sémaphore est semblable au mutex mais avec un compteur en plus. Prenons un exemple semblable au précédent : une pièce a une capacité limitée à un certain nombre de personnes où un tableau affiche le nombre x de personnes qui peuvent encore entrer. Lorsque quelqu'un veut entrer, il vérifie le nombre x (on dit que la sémaphore est *wait*). Si celui-ci est positif, la personne entre et soustrait 1 à la valeur de x. Lorsque la personne ressort de la pièce, à l'inverse, la valeur de x est incrémentée de 1 si personne n'est dans la file d'attente, sinon une personne de la file peut entrer à son tour (la sémaphore est *post*). Si nous résumons la procédure pour l'utilisation d'une ressource par un processus [43, 42] :

- Vérifier la valeur de la sémaphore (*wait*)
- si la valeur de la sémaphore > 0
  - $\mathbf{-1}$ sur la valeur de la sémaphore
  - la ressource est utilisée
  - le processus *post* la sémaphore
    - si file d'attente vide :
      - $\mathbf{+1}$ sur la valeur de la sémaphore
    - sinon
      - un autre processus qui est dans la file peut utiliser la ressource (la valeur de la sémaphore reste nulle)
- si la valeur de la sémaphore $= 0$
  - le processus attend le *post* d'un autre thread

Notons que si la valeur maximale choisie pour la sémaphore est de un, celle-ci est équivalente à un mutex.

L'utilisation de mutex et de sémaphores est à la base de toutes communications entre threads. Nous allons expliquer les différentes techniques de communications mises en place dans notre application. Ces techniques sont différentes en fonction du type de communication.

Afin de permettre de la souplesse dans la vitesse et le moment d'exécution des threads, nous utiliserons le concept du ***buffer*** qui est un tampon dans la communication. Il est généralement possible d'identifier un sens principal dans la communication : "Qui parle à qui ?".

Communication simple

Le cas le plus simple de communication est en utilisant un mutex ou une sémaphore avec une valeur limite de 1 qui protège un emplacement mémoire [2]. Cet emplacement peut être atteint pour une lecture ou pour une écriture (Algorithme 3 et Algorithme 4). Nous remarquons dans l'Algorithme 3 et l'Algorithme 4 que la

Algorithm 3 Écriture d'une donnée dans *data*

Input : donnée à écrire : IN

1:
2: *Mutex.lock()*
3: $data = IN$
4: *Mutex.unlock()*

Algorithm 4 Lecture d'une donnée depuis *data*

Output : donnée lue : OUT

1:
2: *Mutex.lock()*
3: $OUT = data$
4: *Mutex.unlock()*

lecture et l'écriture sont très semblables. Notons que lors d'une lecture ou écriture toute autre lecture ou écriture est bloquée. La technique sera donc évitée pour les données partagées entre de multiples threads.

Problème des Producteur - consommateur

La situation est la suivante : un ou plusieurs threads (producteurs) produisent des données et les placent dans un buffer partagé avec les consommateurs qui utilisent

2. L'emplacement mémoire peut contenir un ensemble d'informations : une image avec ses informations de temps, un tableau de résultats, ...

les données écrites. Il est désiré que chaque donnée écrite soit lue une seule fois avant qu'une autre ne la remplace. Prenons le cas ou le buffer est un tableau de taille N (Figure 4.1).

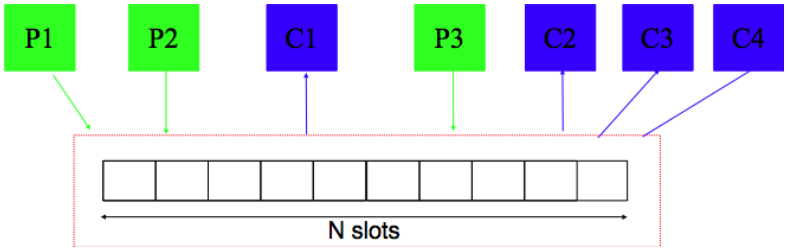


FIGURE 4.1 – Un buffer est le relais dans la communication entre des producteurs "P" qui y écrivent des données et des consommateurs "C" qui utilisent ces données. [42]

L'organisation d'une telle communication se fait à l'aide de deux sémaphores et d'au moins un mutex [42]. Une des sémaphores a le rôle de compteur de places disponibles dans le buffer et l'autre est un compteur de place remplies.

Au début de l'exécution, la première sémaphore que nous appellerons *Empty* est initialisée à N, la taille du buffer. Tandis que la seconde que nous appellerons *Full* est initialisée à 0. En effet, au départ le buffer est sensé être complètement vide.

Deux variables de positions *WritePosition* et *ReadPosition* (initialisées comme étant égales) donnent la prochaine position où écrire ou lire dans le buffer. Le mutex pour protéger l'accès de chaque case du buffer ainsi que la valeur des deux variables *WritePosition* et *ReadPosition*.

Le fonctionnement est présenté dans l'Algorithme 5 pour l'écriture et dans l'Algorithme 6 pour la lecture. En écriture, on attend qu'une case soit libre pour stocker la valeur dans la case *WritePosition* du buffer. Lorsque c'est fait, la prochaine position est actualisée et on prévient qu'une case est remplie. En lecture, la procédure est identique si ce n'est qu'on attend cette fois qu'une case soit remplie et qu'on prévient qu'une case est vide à la fin.

Algorithm 5 Écriture d'une donnée dans *buffer*

Input : donnée à écrire : IN

1:
2: *Empty.wait() //vérifie qu'un emplacement est libre*
3: *Mutex.lock()*
4: *buffer[WritePosition] = IN*
5: *WritePosition = WritePosition + 1*
6: *Mutex.unlock()*
7: *Full.post() //l'emplacement est écrit*

Algorithm 6 Lecture d'une donnée depuis *buffer*

Output : donnée lue : OUT

1:
2: *Full.wait() //attente d'un emplacement écrit*
3: *Mutex.lock()*
4: *OUT = buffer[ReadPosition]*
5: *ReadPosition = ReadPosition + 1*
6: *Mutex.unlock()*
7: *Empty.post() //un emplacement est libre*

Problème de producteur - multi-consommation

Le cas de la communication entre le thread A qui récupère les images puis affiche les résultats et les threads B_i qui recherchent les marqueurs dans des sous-images est particulier. Pour chaque image (+ autres informations) placée dans le buffer, il est attendu que l'entièreté des threads "marqueur" lise l'image et détermine la position de son marqueur. Une fois que tous les threads "marqueur" en ont fini avec l'image, le thread A doit récupérer les résultats avant de placer une nouvelle image dans le buffer. Nous avons donc un cas particulier où le buffer contient à la fois des données à traiter (*input*) et le résultat du traitement (*output*). De plus, le traitement est séparé et se partage sur plusieurs threads. Ceci est représenté à la Figure 4.2.

Pour implémenter cette communication, chaque thread B_i est associé à 2 sémaphores : *Empty* et *Full* qui sont à nouveau le nombre de places du buffer libres et remplies respectivement, mais aux yeux d'un seul thread B_i. Le buffer a pour chacune de ses cases une partie "*input*" et une partie "*output*", chacune protégée par un mutex. Le cas est donc similaire au producteur-consommateur expliqué

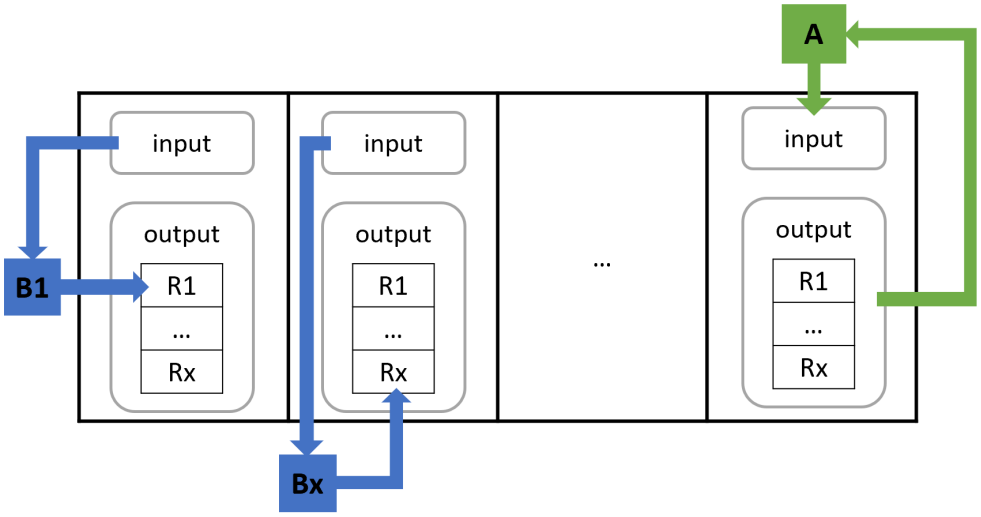


FIGURE 4.2 – Représentation du buffer dans notre cas producteur - multi-consommateur.

précédemment sauf qu'il est multiplié pour être appliqué entre chaque thread B_i et le thread A. La différence étant qu'il y a un seul buffer partagé.

Commençons par expliquer la procédure propre à l'écriture et la lecture de la part du thread A (Algorithme 7). Pour pouvoir lire et remplacer un emplacement du buffer, il faut tout d'abord vérifier que tous les threads B_i ont libéré un emplacement. Ensuite, l'*input* peut être remplacé et l'*output* récupéré en prenant soin d'utiliser les mutexs respectifs. Insistons sur le fait que les résultats récupérés correspondent à l'*input* précédemment présent et qu'il y a donc un délai fonction de la taille x du buffer entre l'écriture des données à traiter et la lecture des résultats.

La procédure pour les threads B_i se décompose en deux parties : d'abord la lecture des données (Algorithme 8) à traiter puis l'écriture des résultats (Algorithme 9). La lecture se fait comme dans le cas producteur-consommateur sauf que la sémaphore *Empty* n'est pas *post* directement.

Une fois les résultats calculés, ceux-ci sont placés dans l'emplacement dédié au tread B_i. Seulement quand le résultat est écrit, la sémaphore *Empty* est *post* pour annoncer que l'emplacement est traité.

Algorithm 7 Thread A : Écriture d'un *input* et lecture d'un *output* dans *buffer*

Input : donnée à écrire dans *input* : IN
Output : donnée lue depuis *output* : OUT

```
1:
2: for tout i jusqu'à x do
3:     Empty[i].wait() //vérifie qu'un emplacement est libre
4: end for
5:
6: buffer[WritePosition].input.Mutex.lock()
7: buffer[WritePosition].input = IN //écrit dans le buffer
8: buffer[WritePosition].input.Mutex.unlock()
9:
10: buffer[WritePosition].output.Mutex.lock()
11: OUT = buffer[WritePosition].output //récupère un résultat précédent
12: buffer[WritePosition].output.Mutex.unlock()
13:
14: WritePosition = WritePosition + 1 //nouvelle position où écrire
15:
16: for tout i jusqu'à x do
17:     Full[i].post() //l'emplacement est écrit
18: end for
```

Algorithm 8 Thread B_i : Lecture d'un *input* depuis le *buffer*

Output : données à traiter : IN

```
1:
2: Full[i].wait() //attente d'un emplacement écrit
3:
4: buffer[Position_i].input.Mutex.lock()
5: IN = buffer[Position_i].input //lis les données à traiter
6: buffer[Position_i].input.Mutex.unlock()
```

Algorithm 9 Thread B_i : Écriture d'un *output* depuis le *buffer*

Input : donnée à écrire : *OUT*

```
1:
2: buffer[Position_i].output.Mutex.lock()
3: buffer[Position_i].output[i] = OUT //écrit le résultat
4: buffer[Position_i].output.Mutex.unlock()
5:
6: Position_i = Position_i + 1 //nouvelle position où lire
7:
8: Empty[i].post() //un emplacement est libre
```

4.2.2 Implémentation et challenges

Comme nous l'avons vu au début de cette section, l'utilisation du multithreading est idéale lorsque différents processus parfaitement indépendants peuvent êtres séparés. Ceci n'est cependant pas notre cas, en séparant la récupération des images à traiter du traitement de chaque sous-image (ROI) pour faire le suivi des marqueurs, la communication des images et résultats entre threads est devenue primordiale. Ce qui implique l'utilisation des outils que nous avons présenté précédemment. Nous verrons malgré tout que le jeu en valait la chandelle et qu'un réel gain en vitesse est obtenu.

Si dans un premier temps l'implémentation des différents threads a été faite en utilisant une librairie générique, l'utilisation de Qt pour l'interface utilisateur nous a poussé à utiliser le multithreading faisant partie de Qt. Les threads sont alors implémentés comme des classes. L'avantage des QThread (tels qu'ils sont appelés) est de pouvoir aisément pouvoir jouer sur la priorité des processus liés au traitement et ainsi les rendre plus importants que la gestion de l'interface utilisateur.

Notre application peut être utilisée dans deux situations différentes :

- Le traitement d'un fichier vidéo pré-enregistré.
- Le traitement "*on-the-fligh*" des images venant d'une caméra (notre caméra stéréo par exemple).

En effet, l'objectif dans ces deux cas n'est pas le même. Dans le premier cas, un fichier résultat des positions des marqueurs pour **toutes les images** est désiré. Le temps de traitement nous importe moins. Dans le second cas, le fonctionnement temps réel est une nécessité. Quitte à "sauter" des images dans le traitement. Par contre, si la vidéo (et l'audio) doit être enregistrée, celle-ci doit autant que possible être complète pour permettre un traitement correct et complet à posteriori. Ces différences d'objectifs se répercutent sur l'architecture du code (et essentiellement des threads).

Quelle que soit la situation, l'interface et le traitement sont séparés. Le traitement, comme nous l'avons vu, est partagé en un thread A qui récupère les images à traiter puis affiche et enregistre les positions des marqueurs trouvés précédemment. Le traitement d'images qui calcule ces positions est fait dans un thread B_i séparé qui traite une sous-image (ROI) pour chaque marqueur.

Traitement d'un fichier pré-enregistré

Lorsqu'un fichier vidéo est traité, toutes les images de celui-ci doivent être traitées. Les choses sont donc systématiques : chaque image de la vidéo est partagée via un buffer "producteur - multi-consommateur" avec les threads B_i et les résultats d'une image précédente sont récupérés pour être affichés et enregistrés.

L'affichage se faisant dans l'interface utilisateur, un signal permet d'envoyer l'image annotée des marqueurs détectés pour actualiser l'interface. Lorsqu'un marqueur nécessite d'être re-sélectionné (voir section 2.2.2) un signal est aussi envoyé à l'interface afin d'y afficher un message.

La communication des paramètres du traitement qui sont sélectionnés dans l'interface est protégée par des mutex.
L'organisation entre threads dans le cas du traitement d'un fichier vidéo est schématisée dans la Figure 4.3.

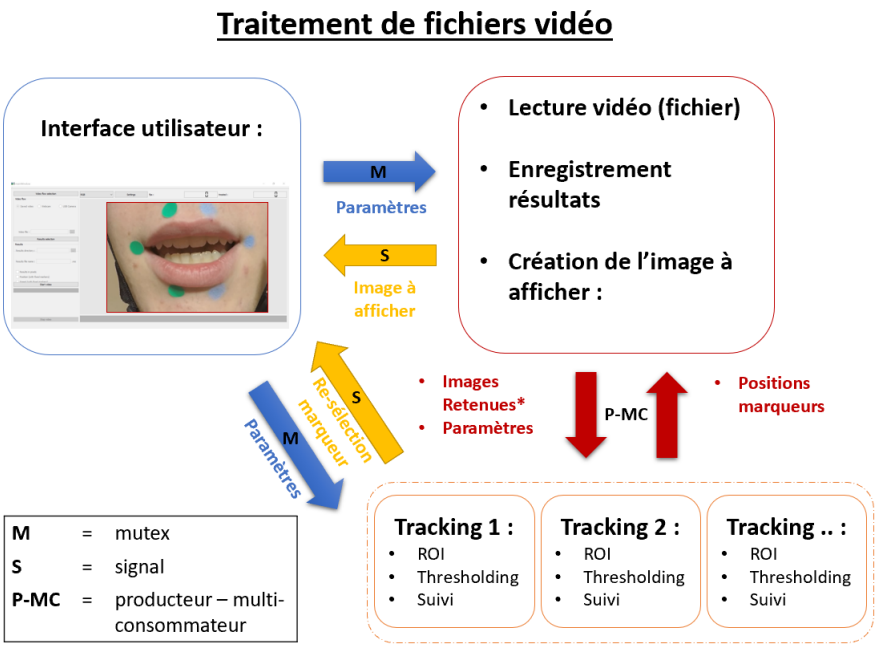


FIGURE 4.3 – Interaction entre les différents threads dans le cas du traitement d'un fichier vidéo. Les threads sont représentés par des rectangles et les buffers et autres communications par des flèches.

Traitement en temps réel des images d'une caméra

Dans le cas du traitement en temps réel des images venant d'une caméra (qui peut être une webcam ou la caméra stéréo), l'utilisation du multithreading est plus critique. Le traitement est le même que lors de la lecture d'une vidéo si ce n'est que les images ne sont pas lues d'un fichier mais d'un buffer. La récupération des images de la caméra se fait en effet dans un processus différent. Ceci permet, si désiré, d'enregistrer la totalité des images sous la forme d'un fichier vidéo tout en permettant que toutes les images ne soient pas traitées si la vitesse de calcul ne le permet pas. Il est préférable d'avoir un fichier vidéo complet qui peut être post-traité.

À la suite de tests avec la caméra stéréo, nous avons remarqué que de meilleures performances sont obtenues si nous enregistrons le fichier vidéo dans un thread séparé avec un buffer d'une certaine taille afin de servir de tampon. Ce buffer est une sorte de queue d'images implémentées par un "producteur-consommateur". Afin d'interpréter les résultats, les logopèdes peuvent avoir besoin d'enregistrer le son également. Un tel enregistrement se fait à une fréquence bien supérieure à celle de l'aquisition des images et fonctionne donc de manière très indépendante sur

base de paramètres choisis dans l'interface avant son commencement.

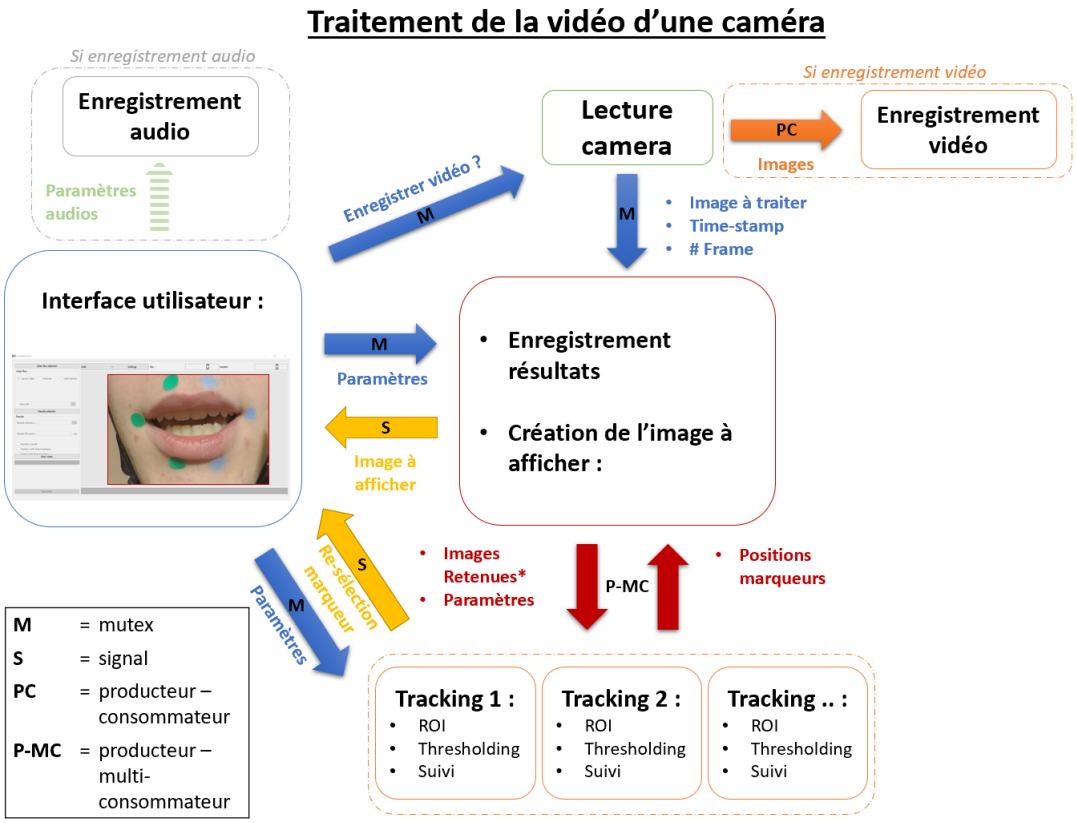


FIGURE 4.4 – Interaction entre les différents processus formant le LMTS Studio 2.0 dans le cas du traitement en temps réel des images de la caméra stéréo.

Problématique de synchronisation entre les threads Les threads fonctionnent généralement de manière libre temporellement. Ceux-ci doivent cependant savoir quand commencer à travailler et surtout quand s'arrêter. Sans cela, un thread pourrait ne jamais être arrêté et continuer à fonctionner même après fermeture de l'application. Pour éviter ce problème, l'information de la fin du traitement est systématiquement passée dans les différents buffers. À chaque lecture du buffer cette information est vérifiée en premier lieu et le processus arrêté le cas échéant.

Si le démarrage des threads se fait à leur création, celui-ci est réellement effectif quand la lecture d'un emplacement du buffer donne une image non-vide et avec un time-stamp (temps lié à l'image) positif. À chaque nouvelle vidéo, les buffers sont ré-initialisés pour ne pas y mélanger des images de la vidéo traitée précédemment.

Test de vitesse Comme nous l'avons vu en début de section, le multithreading permet de réduire le temps total nécessaire au traitement en permettant de faire des opérations simultanément. Si nous parlons en temps nécessaire par image T_{ROI}, faire le traitement de manière séquentielle revient à multiplier le temps moyen pour un marqueur par le nombre de marqueurs à suivre. Comme nous pouvons le voir sur la Figure 4.5, le multithreading a un intérêt qu'à partir de plusieurs marqueurs suivis. Le test montré a été effectué avec un ordinateur à 4 processeurs logiques ($\approx$ 4 coeurs). Ainsi pour 7 marqueurs, les 4 premiers peuvent être faits en même temps et les 3 restants ensuite ce qui fait environ 2 fois le temps T_{ROI}. Pour 10 marqueurs : $10/4 \approx 3$ si arrondis vers le haut.

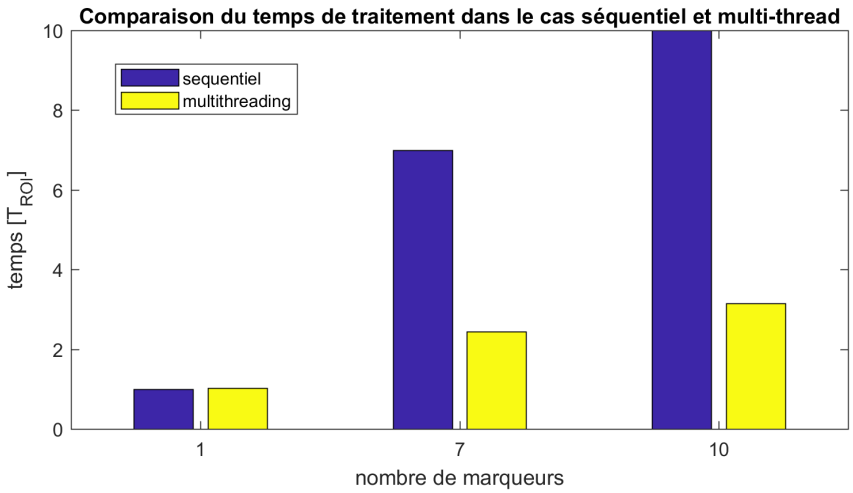


FIGURE 4.5 – Prenant le temps de traitement moyen d'un marqueur T_{ROI} comme référence. Si le traitement était fait de manière séquentielle, il faudrait N fois ce temps pour rechercher N marqueurs dans chaque image. Grâce au multithreading, ce temps est largement réduit (test fait sur un ordinateur avec 4 unités virtuelles de calcul, pour une même vidéo).

Les nombres de marqueurs utilisés dans le test ci-dessus ne sont pas choisis au hasard. Il s'agit des nombres de marqueurs à suivre dans le cas d'une mono-vision (5 marqueurs sur les lèvres et 2 fixes comme repère de distance) et dans le cas stéréo (2 fois les 5 marqueurs sur les lèvres).

Notons que le pré-requis pour que le traitement de N marqueurs dans un flux d'images à $f[image/s]$ se fasse en temps réel est que le temps de traitement d'un marqueur seul soit inférieur à :

$$T_{ROI} < \frac{\frac{1}{f}}{\lceil \frac{N}{\#\text{coeurs}} \rceil}$$

Où $\lceil x \rceil$ est l'arrondi à la valeur supérieure de x. Si le traitement était séquentiel, la condition serait $T_{ROI} < \frac{1}{f \cdot N}$. Les performances nécessaires pour effectuer le traitement seront moindres à partir du moment où #coeurs > 1.

Sur un ordinateur avec des performances modestes [3], la vitesse de traitement d'une vidéo GOPRO avec 7 marqueurs atteint 30 fps (avec ou sans enregistrement des résultats). Une vitesse bien supérieure à 60 fps peut cependant être atteinte sur un ordinateur plus performant.

4.3 Interface caméra

Nous discuterons dans cette section de l'interfaçage d'une caméra industrielle et plus particulièrement dans notre cas de la caméra stéréo. L'utilisation d'une webcam, très semblable à la lecture d'un fichier vidéo grâce à OpenCV ne sera pas discutée.

La caméra que nous avons choisie répond aux standards USB3.0 Vision et Gen<i>Cam ce qui permet de l'utiliser avec des logiciels d'autres fabricants, mais aussi, à l'inverse, d'utiliser la caméra d'un autre fabricant pour la remplacer. Le plus gros de l'interface a été réalisé sur base des exemples de code (C++) et du guide fourni par le fabricant *GetCameras* [44]. La caméra est dans un premier temps paramétrée et démarrée. La récupération des images se fait par la suite sous forme d'évènements : lorsqu'une image est transférée depuis la caméra, une fonction est automatiquement exécutée. Le déroulé de cette fonction est le suivant :

- l'image (formée des deux images des deux capteurs) est obtenue,
- l'image est convertie dans le format utilisé par OpenCV,
- les sous-images correspondant aux deux capteurs sont extraites, ré-orientées l'une par rapport à l'autre et ré-assemblées,
- l'image ré-assemblée est placée dans le buffer pour être traitée
- si enregistrement vidéo : l'image ré-assemblée ou l'image supérieure (choix de l'utilisateur) est placée dans une queue pour être enregistrée dans un fichier vidéo,
- si gain automatique : toute les 20 images, le gain de la caméra est ajusté.

L'utilisation de l'application *Daheng galaxy viewer* fournie par le fabricant est très pratique : elle permet de paramétrer la caméra et de faire des essais, puis de générer les lignes de codes à utiliser pour reproduire les paramètres voulus depuis notre interface C++. Parmi les paramètres qui peuvent être choisis (et donc modifiés par l'utilisateur depuis notre application) il y a :

FrameRate : Nombre d'images capturées par seconde (fps). La caméra est

3. Processeur *intel core i5* avec une vitesse de base de $1.8GHz$ et 8 processeurs logiques. Mémoire vive de $8Gb$ et disque SDD.

triguée par le software et peut atteindre un taux d'images par seconde libre jusqu'à 90 fps.

Temps d'exposition : Temps durant lequel le capteur reste "ouvert" pour chaque image. Celui-ci doit être choisi de sorte de faire un compromis entre la luminosité et la netteté de l'image. Un temps court donne une image nette mais sombre tandis qu'un temps long donnera une image lumineuse (donc pas besoin de gain), mais le moindre mouvement donnera du flou. [4]

Gain : Le gain permet de contrebalancer un temps d'exposition court en augmentant artificiellement la luminosité de l'image.

Contrairement à de nombreuses caméras industrielles, celle que nous avons choisi ne comporte pas de fonction intégrée adaptant automatiquement le gain et la balance des blancs. Si la balance des blancs est relativement peu importante dans notre cas, il est intéressant de ne pas devoir ajuster manuellement le gain. En fonction de l'endroit et du moment dans la journée où une vidéo est prise, la luminosité des images obtenues peut être très variable.

Adaptation automatique du gain L'adaptation automatique du gain que nous avons implémentée se base sur la luminance du centre de l'image de la caméra supérieure (Figure 4.6). Pour ce faire, la procédure suivante est faite toutes les 20 images :

1. la sous-image comprise entre $\frac{1}{3}$ et $\frac{2}{3}$ de la largeur/hauteur de l'image de la caméra supérieure est récupérée.
2. la sous-image est transformée en nuance de gris (nuance de gris = luminance).
3. la moyenne de la luminance de tous les pixels est calculée.
4. la luminance moyenne est comparée à une valeur cible (paramétrable par l'utilisateur).
5. l'erreur observée (divisée par un coefficient d'amortissement) est la modification à apporter au gain de la caméra.
6. le gain de la caméra est modifié.

Le facteur d'amortissement permet d'éviter des changements brusques de luminosité qui perturberaient le traitement d'images. Une image sur 20 est testée pour obtenir une adaptation du gain efficace sans pour autant plomber la vitesse de fonctionnement.

L'utilisation du gain automatique se combine bien avec l'éclairage indépendant. La Figure 4.7 présente deux situations d'acquisition très différentes : l'une avec une

4. Si le capteur reste ouvert pendant un temps significatif face à la vitesse des mouvements observés, un marqueur aura accompli un déplacement pendant l'acquisition de l'image. Il en résulte un marqueur flou dont la position en un temps donné est incertaine.

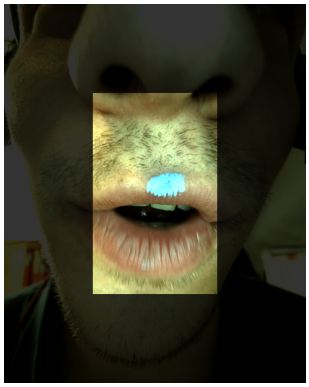
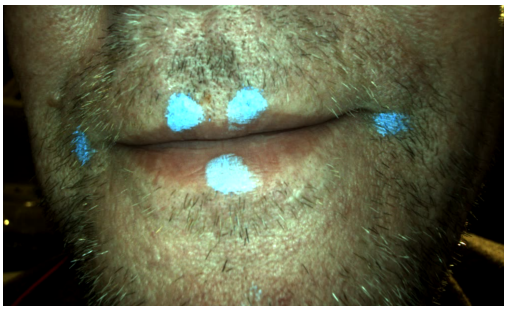


FIGURE 4.6 – L'adaptation du gain est faite sur base de la luminance du centre de l'image d'une des caméras.

forte illumination environnante et l'autre dans une quasi obscurité. Malgré cela, les deux vidéos sont parfaitement exploitables par notre algorithme. Remarquons aussi que les deux sujets ont une barbe légère qui n'entrave pas pour autant le placement des marqueurs au maquillage.



(a) Situation de forte sur-exposition.



(b) Situation de faible luminosité

FIGURE 4.7 – Images prises avec une des caméras, avec éclairage et réglage du gain automatique.

4.4 Enregistrement du son

Lors de l'analyse des résultats obtenus avec le LMTS 2.0, il peut être utile de mettre en relation les courbes du mouvement des lèvres avec les sons ou mots prononcés par le patient. L'utilisation de la vidéo enregistrée est déjà un premier point. Cependant, tout le monde n'a évidemment pas la capacité de lire sur les lèvres. L'enregistrement d'un fichier audio synchronisé avec la vidéo permet donc une interprétation plus aisée des résultats obtenus.

La fréquence d'acquisition du son étant plusieurs ordres de grandeurs au dessus de la fréquence d'acquisition des images (plusieurs kHz contre une dizaine de Hz), l'enregistrement du son doit se faire de manière séparée.

Une implémentation présente dans les exemples de *QtCreator* a été utilisée [45]. Celle-ci a simplement été interfacée à notre application et certaines de ses options superflues ont été enlevées. La fenêtre présentée Figure J.12 fait partie de l'interface modifiée depuis lors [45]. L'enregistrement sonore est démarré et puis arrêté au même moment que l'enregistrement de la vidéo. Ce qui permet, sauf dans le cas où des images sont perdues dans l'enregistrement vidéo, d'obtenir des enregistrements faisant la même durée.

L'utilisation de l'enregistreur de son et les quelques réglages qui lui sont propres est présenté dans l'Annexe J.2.

4.5 Interface graphique

L'interface graphique peut être décomposée en 8 parties distinctes représentées sur la Figure 4.8 :

1. La fenêtre vidéo
2. Le champ "choix du flux vidéo"
3. Le champ "résultats"
4. Le champ "lancement/arrêt vidéo"
5. Le bouton "choix de l'espace de couleur"
6. Le bouton "options du traitement d'images"
7. Le champ "vitesse du flux vidéo et du traitement d'images"
8. Le champ "informations générales"

Chacune de ces parties sont décrites plus en détail dans l'Annexe J.2.

Cette interface a été pensée afin que l'utilisateur effectue le traitement d'images de manière séquentielle. La colonne gauche à de la fenêtre vidéo est dédiée aux principales étapes que doit accomplir l'utilisateur.

En haut de cette colonne, il est demandé de dans le champ "choix du flux vidéo" de sélectionner le type de vidéo à traiter. Il y a trois possibilités : une vidéo déjà enregistrée, une vidéo issue de la webcam de l'ordinateur ou une vidéo issue de la caméra stéréo. Selon le type de flux sélectionné, certaines options comme les paramètres de la caméra stéréo ou l'enregistrement du son sont proposées.

Ensuite, dans le champ "résultats" situé juste en dessous, il est demandé à l'utilisateur de choisir le dossier dans lequel les résultats seront enregistrés ainsi que le nom du fichier résultats. Il est également demandé si la vidéo traitée est en stéréovision ou bien en monovision. Enfin, le type de résultats désirés doit être choisi parmi : les positions en pixels, les positions en mm et les vitesses en mm/s. Si aucune de ces propositions n'a été sélectionnée, aucun fichier résultats ne sera créé.

En bas de la colonne de gauche, le champ "lancement/arrêt vidéo" regroupe toutes les actions que peut réaliser l'utilisateur durant le traitement de la vidéo. Cela inclut le démarrage de la vidéo et son arrêt, la mise en pause de la vidéo ainsi que l'initialisation et la resélection des marqueurs.

Nous avons également souhaité laisser à l'utilisateur la liberté de modifier l'intégralité des paramètres du traitement d'images. Pour ce faire, il suffit de cliquer sur le bouton "options du traitement d'images" au dessus de la fenêtre vidéo et de modifier les paramètres dans la fenêtre affichée (Figure J.22).

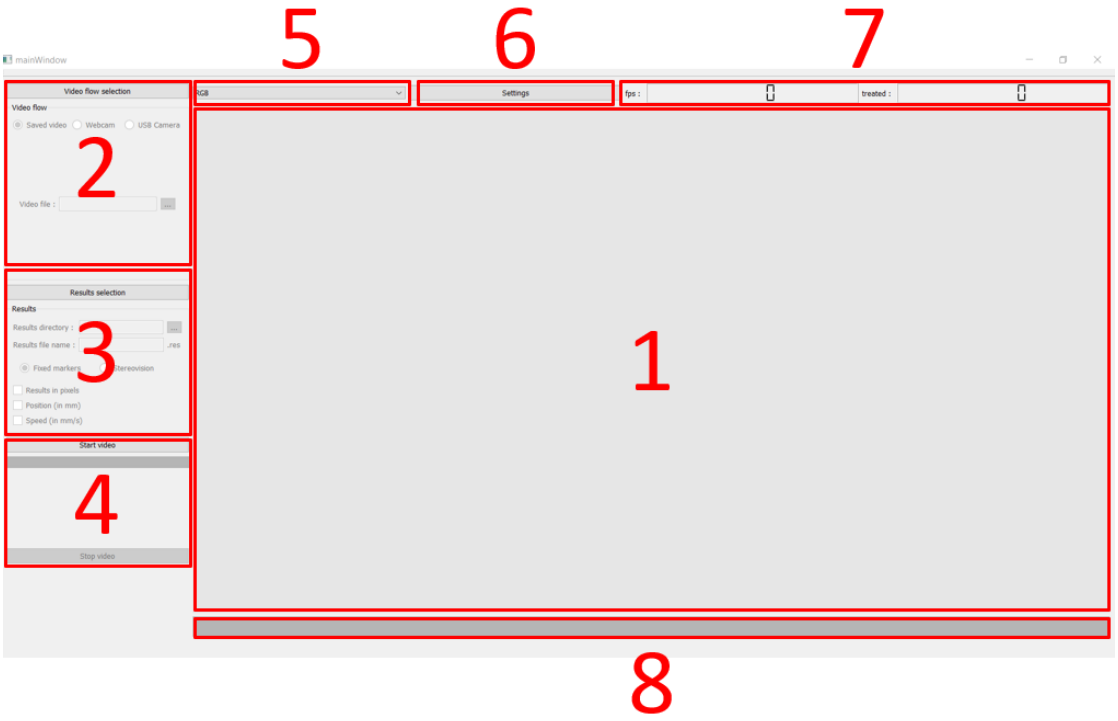


FIGURE 4.8 – Vue globale de l'interface graphique

4.6 Conclusions sur l'application complète

L'application LMTS 2.0 est notre principale réalisation. Le code constituant celle-ci a été écrit et pensé pour pouvoir être repris et amélioré dans le futur. Les feedbacks réguliers de la part de la mémorante en logopédie (qui a utilisé notre application avec des vidéos prises avec la GOPRO) nous ont permis d'améliorer la prise en main comparée à nos premières versions. Nous avons également pu corriger des problèmes de compatibilités entre ordinateurs et réfléchir à la manière dont l'installation se fait sur un nouvel ordinateur.

N'ayant pas d'ordinateur suffisamment performant, nous n'avons pas pu faire de tests de suivi du mouvement des lèvres en temps réel à proprement parler. L'enregistrement des vidéos venant de la caméra stéréo n'a pas été possible pour nous au delà de 10 fps (20 fps si seulement une caméra) sans que nous observions des pertes d'images.

Les performances minimales requises pour utiliser la caméra stéréo sont annoncées par le fabricant comme d'avoir minimum un "intel core duo" fonctionnant à 2.4 GHz. Afin d'utiliser un débit d'images suffisant pour suivre le mouvement des lèvres (40 à 50 fps minimum), un processeur supérieur à un "intel core i5" est requis.

Une amélioration future très utile pour l'utilisation du LMTS Studio 2.0 serait de pouvoir enregistrer les différents paramètres choisis pour le traitement et la caméra. En effet, dans la version actuelle, à chaque démarrage de l'application, les paramètres reviennent à ceux choisis par défaut (dans le code). Dans le cas où ces paramètres ne sont pas adaptés il est nécessaire de re-modifier ceux-ci à chaque ouverture de l'application.

De manière générale, le code composant le LMTS Studio 2.0 a été écrit dans l'optique de pouvoir être amélioré et ré-utilisé par d'autres personnes dans le futur. En quelques chiffres, le projet Visual Studio est composé de 29 fichiers de code "C++" (fichiers sources et headers compris), 4 fenêtres Qt et implémente 28 classes différentes. Le lecteur intéressé trouvera dans l' Annexe I un diagramme de classe expliqué reprenant la quasi-totalité de celles-ci. Ce type de graphique (Figure I.1) donne un bel aperçu de l'inter-connexion entre l'interface utilisateur et le traitement, de manière plus complète que ce que nous avons présenté Figure 4.3 et Figure 4.4.

Chapitre 5

Discussion et Conclusions

Bien que les trois chapitres de ce mémoire aient été présentés de manière indépendante, ceux-ci s'inter-connectent pour former ce que nous avons appelé le LMTS 2.0. Une belle illustration de ces interactions est donnée Figure 5.1 où la vidéo de la caméra stéréo (placée sur son casque) est traitée en temps réel. L'algorithme y suit 5 marqueurs (dans ce cas-ci des gommettes bleues) dans les deux images affichées sur l'écran.

Le système complet prouve ainsi être entièrement fonctionnel bien que nous n'ayons pas été en mesure d'utiliser ses pleines capacités, n'ayant pas d'ordinateur suffisamment performant à notre disposition. Le système complet permet de :

- Faire le suivi, après initialisation manuelle, de tout marqueur de couleur posé sur la peau et dont le contraste est suffisant : gommette de couleur, tâche faite au maquillage, ...
- Enregistrer les positions des marqueurs recherchés en pixel ou en millimètre ainsi que les vitesses correspondantes en mm/s.
- Avoir au minimum un feedback du traitement en temps réel, et de traiter l'entièreté du flux vidéo en temps-réel si les performances de l'ordinateur sont suffisantes.
- Enregistrer une vidéo depuis une caméra industrielle (caméra stéréo) ou une webcam.
- Enregistrer le son (de manière synchronisée à la vidéo) dans un ficher séparé.
- Post-traiter une vidéo enregistrée avec la caméra stéréo ou la GOPRO (dans ce cas, la vidéo doit être convertie en ".avi").

Les tests qui ont été effectués montrent de bonnes performances du système en terme de suivi et de robustesse de l'algorithme. Tout d'abord, la détection des marqueurs dans les images est précise (de l'ordre du pixel) et fait face à peu de

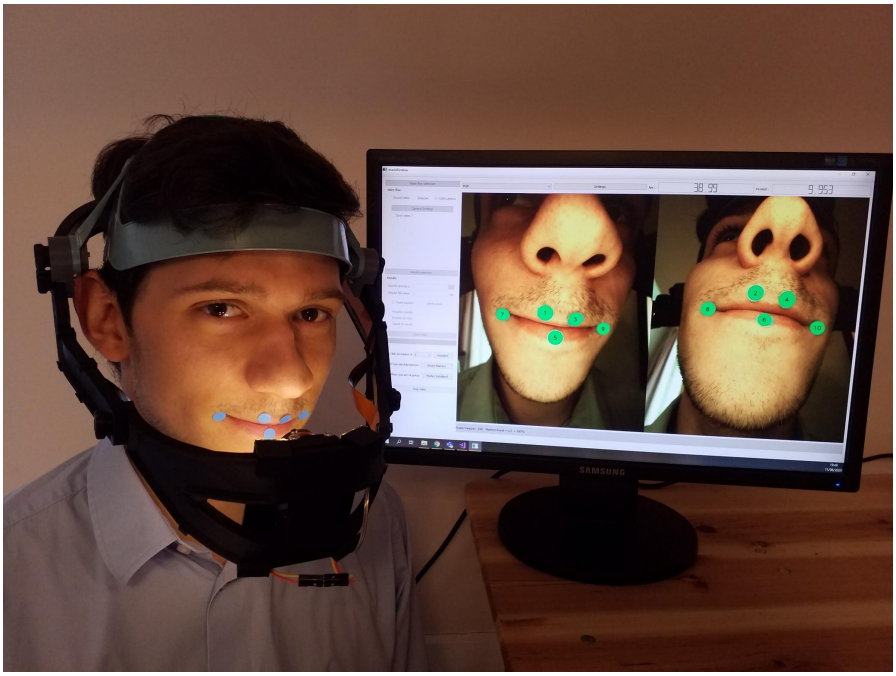


FIGURE 5.1 – Traitement en temps réel de la vidéo de la caméra stéréo via le LMTS Studio 2.0.

pertes (99.7% de réussite). Ensuite, la transformation en coordonnées spatiales via la stéréoscopie donne :

- une erreur sur la profondeur inférieure à 1.0% dans la plage d'intérêt autour de $80mm$ (avec un fit de l'erreur).
- une erreur sur les distances mesurées entre marqueurs de moins de 5% (avec un fit de l'erreur).

Malgré le travail présenté dans ce mémoire, le dispositif réalisé nécessite encore de nombreux tests et quelques améliorations avant de pouvoir envisager une utilisation hors d'un contexte de recherche. En effet, des essais du nouveau casque, à plus grande échelle et dans des conditions réelles, devraient également être réalisés afin de détecter d'éventuels défauts. De plus, un usage en tant que dispositif médical nécessiterait une certification comme tel.

La réalisation de ce système de suivi du mouvement des lèvres a mis en évidence le besoin d'une caractérisation rigoureuse des mouvements labiaux :

- Vitesse maximale observable,
- Amplitudes minimales et maximales des mouvements,
- Fréquence des mouvements volontaires mais aussi des mouvements non-contrôlés.

Ces différents points devraient former les bases du cahier des charges d'un dispositif ultérieur car ils impactent directement la précision et le nombre de fps nécessaire. À défaut, les performances atteignables avec la caméra stéréo choisie sont suffisamment larges pour permettre cette caractérisation via une campagne d'acquisition sur un grand nombre de personnes.

Enfin, comme nous en avons déjà fait la remarque dans le chapitre 2, notons que l'algorithme mis en place permet le suivi de marqueurs de couleurs dans bien d'autre situations que le suivi du mouvement des lèvres. Tout en utilisant le casque fabriqué, les marqueurs peuvent êtres placés sur tout le visage pour, par exemple, faire du suivi après une opération de chirurgie plastique [46]. De manière générale, des points de couleurs peuvent êtres placés sur n'importe quel support imaginable et la vidéo obtenue pourra donc donner des résultats de positions pour une multitude de situations.

En bref, l'application "LMTS Studio 2.0" présente un algorithme de traitement robuste et performant (même avec des marqueurs maquillés). Son utilisation, par une mémorante en neuropsychologie, nous a valu jusqu'à présent des retours positifs.

Bibliographie

[1] Dysarthrie. `https://orthophonie.ooreka.fr/comprendre/dysarthrie`.

[2] Lise COLMANT. Lip motion tracking system. Master's thesis, EPL (UCLouvain), 2015.

[3] Sophie LAURENT. Influence de l'observation d'actions motrices langagières sur la qualité articulatoire de mots chez des enfants présentant des troubles articulatoires d'origine neurologique. Master's thesis, UCLouvain, 2015.

[4] Amélie Van Thorre. Impact de la stimulation des neurones miroirs, via l'observation d'actions motrices gestuelles, sur la qualité articulatoire de mots chez un patient présentant une apraxie de la parole. Master's thesis, UCLouvain, 2016.

[5] Amrelle MERMOZ. Influence de l'observation d'actions motrices langagières sur la production de phrases de patients aphasiques transcorticaux moteurs. Master's thesis, UCLouvain, 2016.

[6] Diaria Melnik. Comparaison du codamotion et du lip motion tracking system sur base de résultats obtenus sur des sujets adultes tout-venant. Master's thesis, UCLouvain, 2016.

[7] Laura Gilles. Influence de la stimulation des neurones miroirs, par le biais de l'observation d'actions motrices langagières, sur la qualité articulatoire de mots chez des sujets sains. Master's thesis, UCLouvain, 2017.

[8] Céline Hermant Margot Périn. Lip motion tracking system (lmts) : étude exploratoire de la stabilité des mouvements labiaux au cours du temps chez des adultes tout-venant. Master's thesis, UCLouvain, 2017.

[9] Abdelilah Sellami. Rapport de stage, 2016.

[10] Richard Szeliski. *Computer Vision : Algorithms and applications*. Springer, 2011.

[11] Nicolas Eveno, Patrice Delmas, Pierre-yves Coulon, and Laboratoire Des. Vers L' Extraction Automatique Des Lèvres D' Un Visage Parlant. *GRETSI 01*, pages 193–1196, septembre 2001.

[12] Pierre-Yves COULON Nicolas EVENO, Alice CAPLIER. Automatic and accurate lip tracking. *IEEE Transactions on Circuits and Systems*, 2003.

[13] H. Seyedarabi, W. Lee, and A. Aghagolzadeh. Automatic lip tracking and action units classification using two-step active contours and probabilistic neural networks. pages 2021–2024, 2006.

[14] Sébastien Stillittano, Vincent Girondel, and Alice Caplier. Lip contour segmentation and tracking compliant with lip-reading application constraints. *Machine Vision and Applications*, 24(1) :1–18, 2013.

[15] Isabelles L. HERLIN Bertrand LEROY. Un modèle déformable paramétrique pour la reconnaissance de visages et le suivi du mouvement des lèvres. *Proceedings GRETSI'95*, pages 701–704, septembre 1995.

[16] G. BAILLY P. GACON, P.-Y. COULON. Modèle statistique et description locale d'apparence pour la détection des contours des lèvres. *Noûs*, pages 1–4, 2005.

[17] Jamal Ahmad Dargham, Ali Chekima, and Sigeru Omatu. Lip detection by the use of neural networks. *Artificial Life and Robotics*, 12(1-2) :301–306, 2008.

[18] Ahmad B.A. Hassanat, Mouhammd Alkasassbeh, Mouhammd Al-Awadi, and Esra'A A.A. Alhasanat. Colour-based lips segmentation method using artificial neural networks. In *2015 6th International Conference on Information and Communication Systems, ICICS 2015*, pages 188–193. Institute of Electrical and Electronics Engineers Inc., may 2015.

[19] Zign Creations. Reference manual : Zign track 2. `https://zigncreations.com/reference_manual_zt2.pdf`.

[20] Klaus Dorfmüller. Robust tracking for augmented reality using retroreflective markers. *Computers and Graphics (Pergamon)*, 23(6) :795–800, 1999.

[21] `https://www.qualisys.com/hardware/accessories/passive-markers/super-spherical-markers/`.

[22] Omid Haji Maghsoudi, A. Vahedipour Tabrizi, B. Robertson, and Andrew Spence. Superpixels based marker tracking vs. hue thresholding in rodent biomechanics application. *Conference Record of 51st Asilomar Conference on Signals, Systems and Computers, ACSSC 2017*, 2017-October :209–213, 2018.

[23] Jesse S. Jin Richard Y.D. Xu, John G. Allen. Robust Real-Time Tracking of Non-rigid Objects. *Conferences in Research and Practice in Information Technology, Vol. 36*, 2004.

[24] Fan Yang Ming-Hsuan Yang Shu Wang, Huchuan Lu. Superpixel Tracking. *International Conferences on Computer Vision*, 2011.

[25] Zhiyuan Liang Yu Liu Wenguan Wang Ling Shao Jianbing Shen, Xiaopeng Hao. Real-Time Superpixel Segmentation by DBSCAN Clustering Algorithm. *IEEE Transactions on image processing, Vol. 25, No. 12*, 2016.

[26] Pierre Venet. Portage d'un algorithme de traitement d'image, 2015.

[27] Documentation opencv. `https://docs.opencv.org/`, 31/12/2019.

[28] Optique pour l'ingénieur. Modélisation et calibrage d'une caméra. `http://www.optique-ingenieur.org/fr/cours/OPI_fr_M04_C01/co/Grain_OPI_fr_M04_C01_2.html`, juin 2008.

[29] Zhengyou Zhang. Flexible Camera Calibration By Viewing a Plane From Unknow Orientations. *IEEE*, 1999.

[30] Zhengyou Zhang. Camera Calibration with One-Dimentional Objects. *IEEE*, 2004.

[31] Wilhelm Burger. Zhang ' s Camera Calibration Algorithm : In-Depth Tutorial and Zhang ' s Camera Calibration Algorithm : In-Depth Tutorial and Implementation by. *techreport (BurgerCalibration2016)*, 2019.

[32] Huijie Zhao, Lunbao Xu, Shaoguang Shi, Hongzhi Jiang, and Da Chen. A high throughput integrated hyperspectral imaging and 3D measurement system. *Sensors (Switzerland)*, 18(4), 2018.

[33] Richard Hartley and Andrew Zisserman. *Multiple View Geometry in computer vision*. Cambridge University Press, 2003.

[34] A. Paolillo G. Di Leo, C. Liguori. Propagation of uncertainty through stereo triangulation. *IEEE*, 2010.

[35] Sanja Fidler. Stereo epipolar geometry for general cameras. `http://www.cs.toronto.edu/~fidler/slides/CSC420/lecture13.pdf`.

[36] Daniel Scharstein. *View Synthesis Using Stereo Vision*. Springer, 1999.

[37] Joshua Gluckman and Shree K. Nayar. Rectified catadioptric stereo sensors. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24(2) :224–236, 2002.

[38] Doo Hyun Lee and Inso Kweon. A novel stereo camera system by a biprism. *IEEE Transactions on Robotics and Automation*, 16(5) :528–541, 2000.

[39] Ting Xue, Liqun Qu, Zhaofeng Cao, and Tao Zhang. Three-dimensional feature parameters measurement of bubbles in gas – liquid two-phase flow based on virtual stereo vision. *Flow Measurement and Instrumentation*, 27 :29–36, 2012.

[40] Pascal Chevalier. Comment calculer la focale de l'objectif de votre caméra ?

[41] Qt. Threading Basics.

[42] Bonaventure Olivier. Systèmes informatiques site du cours sinf1252 de l'uclouvain.

[43] Jim Cooling. *Real-time Operating Systems Book 1, The Theory*. Lindentree Associates, 2018.

[44] GetCameras. C++ Programmer's guide.

[45] Qt Company. Audio Recorder Example.

[46] G. K. Essick C.-A. Trotman, J. J. Faraway. Three-dimensional nasolabial displacement during movement in repaired cleft lip and palate patients," Plastic and reconstructive surgery. *Plastic and reconstructive surgery, Vol. 105, No. 4, pp. 1273-1283*, 2000.

[47] Rouge vert bleu - wikipedia. `https://fr.wikipedia.org/wiki/Rouge_vert_bleu`, 31/07/2019.

[48] L'espace cie lab. `http://www.tsi.enst.fr/pages/enseignement/ressources/beti/correl_couleur/lab.html`.

[49] Fei-Fei Li. Dimensionality ReducBon Machine (3D to 2D) 3D world 2D image.

[50] Oussama Khatib Bruno Siciliano. *Springer Handbook of Robotics*. Springer, 2016.

Annexe A

Espaces colorimétriques

Les différents espaces de couleurs utilisés sont décrits ci-après en utilisant les définitions implémentées dans la librairie *OpenCV* utilisée :

Espace RGB : L'espace **RGB** - pour *Red Green Blue* - ou *RVB* en français est l'espace "de référence" pour décrire une image. En effet, toute couleur peut être obtenue par le mélange de trois couleurs primaire : le rouge, le vert et le bleu. On parle de synthèse additive (voir Figure A.1). L'écran d'un ordinateur reproduira une image en ayant pour chacun de ses pixels une intensité pour ces 3 couleurs.

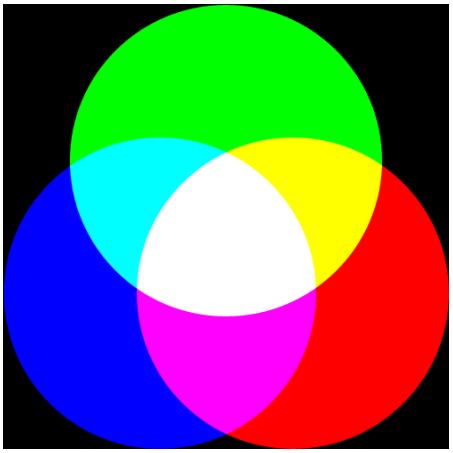


FIGURE A.1 – Synthèse additive des différentes couleurs possibles à base des couleurs primaire : le rouge, le vert et le bleu.

L'intensité de chacune de ces couleurs est stockée dans un octet qui peut donc avoir $2^8 = 256$ valeurs différentes. Un pixel complet peut donc avoir $256^3 = 16\,777\,216$

couleurs différentes soit trente fois le nombre de couleurs différentiables par l'humain [47] ! Un représentation de ces couleurs possibles est donnée dans la Figure A.2.

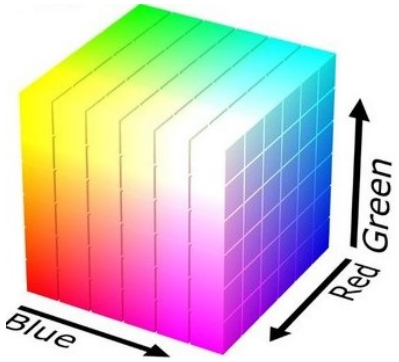


FIGURE A.2 – Espace de couleurs RGB. *Images by Michael Horvath, available under Creative Commons.*

Espace HSV : L'espace **HSV** - pour Hue Saturation Value - est une représentation très utile en traitement d'image qui s'obtient par une transformation de l'espace RGB. La teinte (Hue) y est découplée de la saturation (ou intensité) et de la valeur (ou brillance) de la couleur.

$$\begin{cases} M &= max(R,G,B) \\ m &= min(R,G,B) \\ C &= M - m \end{cases}$$

$$H = \begin{cases} (60^o \cdot \dfrac{G-B}{C}) mod 360^o & \text{si R = max} \\ (60^o \cdot \dfrac{B-R}{C} + 120^o) & \text{si G = max} \\ (60^o \cdot \dfrac{R-G}{C} + 240^o) & \text{si B = max} \end{cases}$$

$$S = 1 - \frac{m}{M}$$
$$V = M$$

Cet espace est très utile pour la recherche de couleurs car la valeur de teinte H sera indépendante des variations de luminosité et d'éventuels reflets. Une représentation de l'espace de couleur est donnée Figure A.3.

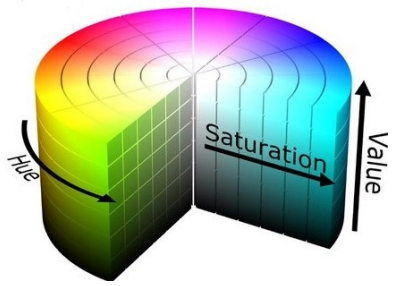


FIGURE A.3 – Espace de couleurs HSV. *Images by Michael Horvath, available under Creative Commons.*

Espace LAB : Tout comme l'espace précédent, le **LAB** est dérivé du RGB via une transformation :

$$\begin{pmatrix} X_0 \\ Y \\ Z_0 \end{pmatrix} = \begin{bmatrix} 0,412453 & 0,357580 & 0,180423 \\ 0,212671 & 0,715160 & 0,072169 \\ 0,019334 & 0,119193 & 0,950227 \end{bmatrix} \cdot \begin{pmatrix} R \\ G \\ B \end{pmatrix}$$

$$\begin{aligned}
X &= \frac{X_0}{X_n} && \text{où } X_n = 0,950456 \\
Z &= \frac{Z_0}{Z_n} && \text{où } Z_n = 1,088754 \\
L &= 116\Big(\frac{Y}{Y_n}\Big)^{\frac{1}{3}} - 16 && \text{pour } \tfrac{Y}{Y_n} > 0,008856 \\
L &= 903,3\frac{Y}{Y_n} && \text{pour } \tfrac{Y}{Y_n} \leq 0,008856 \\
a &= 500\Big(f\big(\frac{X}{X_n}\big) - f\big(\frac{Y}{Y_n}\big)\Big) + \delta \\
b &= 200\Big(f\big(\frac{Y}{Y_n}\big) - f\big(\frac{Z}{Z_n}\big)\Big) + \delta
\end{aligned}$$

Avec :

$$\begin{aligned}
\delta &= 128 && \text{pour une image 8 bit} \\
f(t) &= \sqrt[3]{t} && \text{pour } t > 0,008856 \\
f(t) &= 7,7787t + \frac{16}{116} && \text{pour } t > 0,008856
\end{aligned}$$

La valeur L représente la luminance (en pourcentage) tandis que a et b donnent l'information de couleur. La couleur est définie à partir d'un mélange de deux teintes : une première (a) qui va du vert au magenta et la seconde (b) qui va du bleu au jaune [48]. Une représentation de l'espace LAB est donnée Figure A.4.

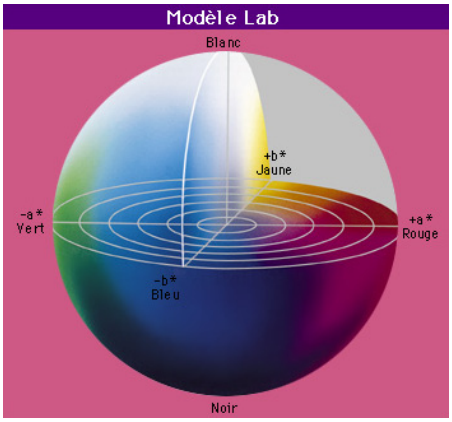


FIGURE A.4 – Représentation de l'espace LAB [48].

Espace YUV ou YCbCr L'espace **YUV** - aussi appelé **YCbCr** - a été à l'origine créé pour résoudre le problème du passage de la télévision noir et blanc à la télévision couleur. Afin que les postes noir et blanc continue à fonctionner il était nécessaire de continuer à envoyer un signal de luminance auquel il fallait rajouter les informations de couleurs.
Les composantes choisies sont les suivantes :

- $Y \propto R + G + B$ est la luminance (image N-B),
- $U = Cb \propto B - Y$ est une image du niveau de bleu,
- $V = Cr \propto R - Y$ est une image du niveau de rouge.

La transformation exacte à partir de l'espace RGB est la suivante :

$$\begin{aligned} Y &= 0,299 \cdot R + 0,587 \cdot G + 0,114 \cdot B \\ Cb &= (B - Y) \cdot 0,564 + \delta \\ Cr &= (R - Y) \cdot 0,713 + \delta \end{aligned}$$

$$\delta = 128 \quad \text{pour une image 8 bit}$$

La Figure A.5 représente le plan *Cb*-*Cr* pour une luminosité constante de $Y = 0,5$ (toutes les valeurs étant exprimées de -1 à 1). Notons que le rouge et le bleu y sont opposés.

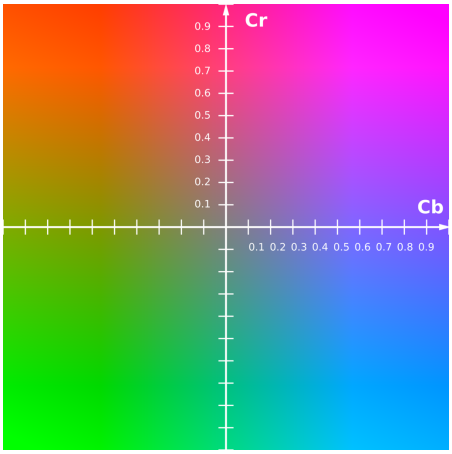


FIGURE A.5 – Plan Cb-Cr pour une luminosité constante de $Y = 0,5$. Image de Simon A. Eugster.

Annexe B

Algorithmes du traitement d'image

Une explication en pseudo-code des algorithmes présentés sous-section 2.2.2 est donnée ci-après pour permettre une meilleure compréhension de la part du lecteur.

Algorithm 10 GetRoiGom

Input : Image complète : $ImageOriginal$
Input : Position précédente du marqueur : $x,\ y$
Input : Largeur et hauteur de la ROI : $dx,\ dy$
Output : Image ROI : $ImageROI$

```
1:
2: // Si dx ou dy trop grand
3: if dx > largeur then
4:     dx = largeur
5: end if
6: if dy > hauteur then
7:     dy = hauteur
8: end if
9: // Calcul du coin supérieur gauche
10: x0 = x − dx/2.0
11: if x0 < 0 then
12:     // Si marqueur proche du bord gauche
13:     x0 = 0
14: else if x0 > (largeur(ImageOriginale) − dx) then
15:     // Si marqueur proche du bord droit
16:     x0 = (largeur(ImageOriginale) − dx)
17: end if
18: y0 = (y − dy)/2.0
19: if y0 < 0 then
20:     // Si marqueur proche du bord supérieur
21:     y0 = 0
22: else if y0 > (hauteur(ImageOriginale) − dy) then
23:     // Si marqueur proche du bord inférieur
24:     y0 = (largeur(ImageOriginale) − dy)
25: end if
26: // Récupération de la sous-image
27: ImageROI = ImageOriginal.Rect(x0, y0, dx, dy)
```

Algorithm 11 ThresholdGom

Input : Image ROI : $ImageROI$
Input : Couleur moyenne marqueur : $couleur$
Input : Espace de couleur à utiliser : $Espace_Couleur$
Input : Seuils de couleur à utiliser : $seuil_couleur$
Output : Marqueur détouré : $ImageROI_Thresholded$
Output : Marqueur détouré érodé : $ImageROI_Eroded$
Output : Marqueur trouvé ? : $marker_found$

1:
2:
3: // Transformation espace de couleur
4: $ImageROI_Transformed = cvtColor(ImageROI,\ Espace_Couleur)$
5:
6: // Thresholding de la ROI : Les pixels dans la plage
7: $(couleur - seuil_couleur)$ à $(couleur + seuil_couleur)$ sont conservés
8: $ImgThresholded = inRange(couleur,\ seuil_haut_couleur)$
9:
10: // Ouverture et fermeture
11: $ImgTresholded = erode(ImgThresholded)$
12: $ImageROI_Eroded = ImgThresholded$
13: $ImgTresholded = dilate(ImgThresholded)$
14: $ImgTresholded = closing(ImgThresholded)$
15: $ImageROI_Thresholded = ImgThresholded$
16:
17: $marker_found =!(CurentSize < InitialSize/VarSize \ || \ CurentSize > InitialSize \cdot VarSize)$

Algorithm 12 TrackGom

Input : Anciennes valeurs de couleur et position : old_color, $old_position$
Input : Marqueur détouré : $ImageROI_Thresholded$
Input : Marqueur détouré érodé : $ImageROI_Eroded$
Output : Couleur à suivre et position retenue : $couleur$, (x, y)

1:
2: // Moyenne couleur du marqueur détouré érodé
3: $new_color = mean(ImageROI_Eroded)$
4:
5: $couleur = new_color \cdot \alpha + old_color \cdot (1 - \alpha)$
6:
7: // Calcul des contours de l'image détourée
8: $contours = findContours(ImageROI_Thresholded)$
9:
10: // Détermination de la position du contour le plus proche
11: **for** *chaque contour* **do**
12: *Calcul distance par rapport au centre précédent du marqueur*
13: **if** *Distance plus petite que pour les autres contours* **then**
14: $new_position = meanPosition(contour)$
15: **end if**
16: **end for**
17:
18: // Nouvelle position du marqueur
19: $(x, y) = new_position \cdot \beta + old_position \cdot (1 - \beta)$

Annexe C

Histogrammes de couleur :

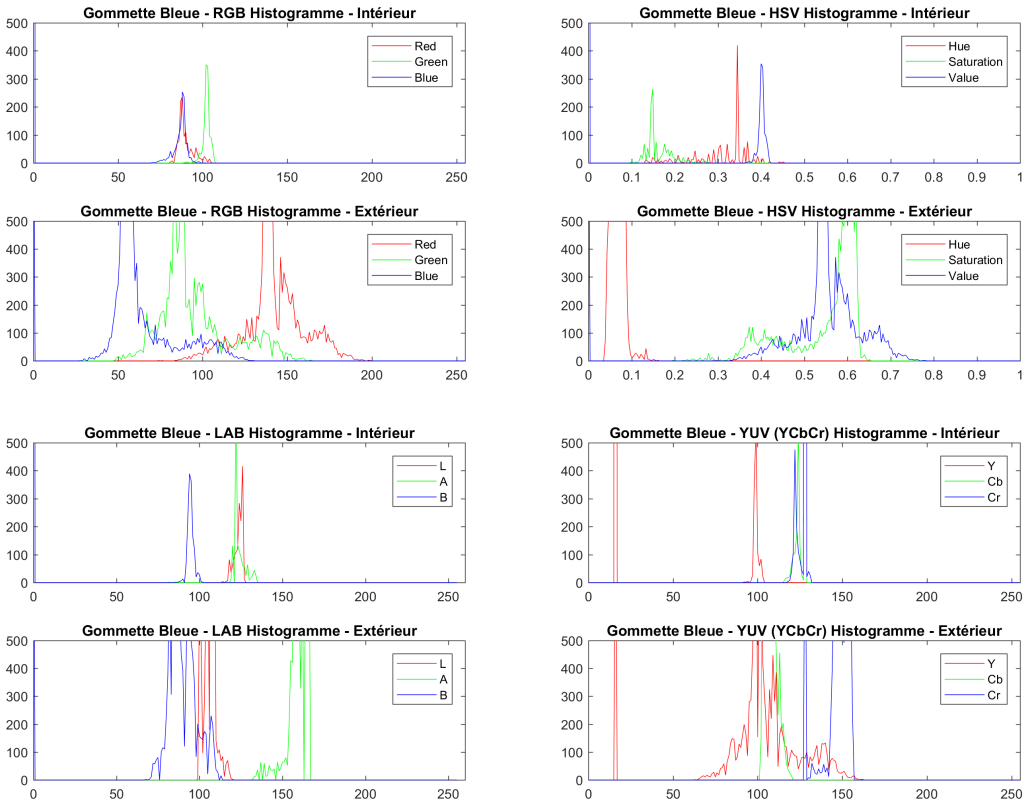


FIGURE C.1 – Gommette bleue

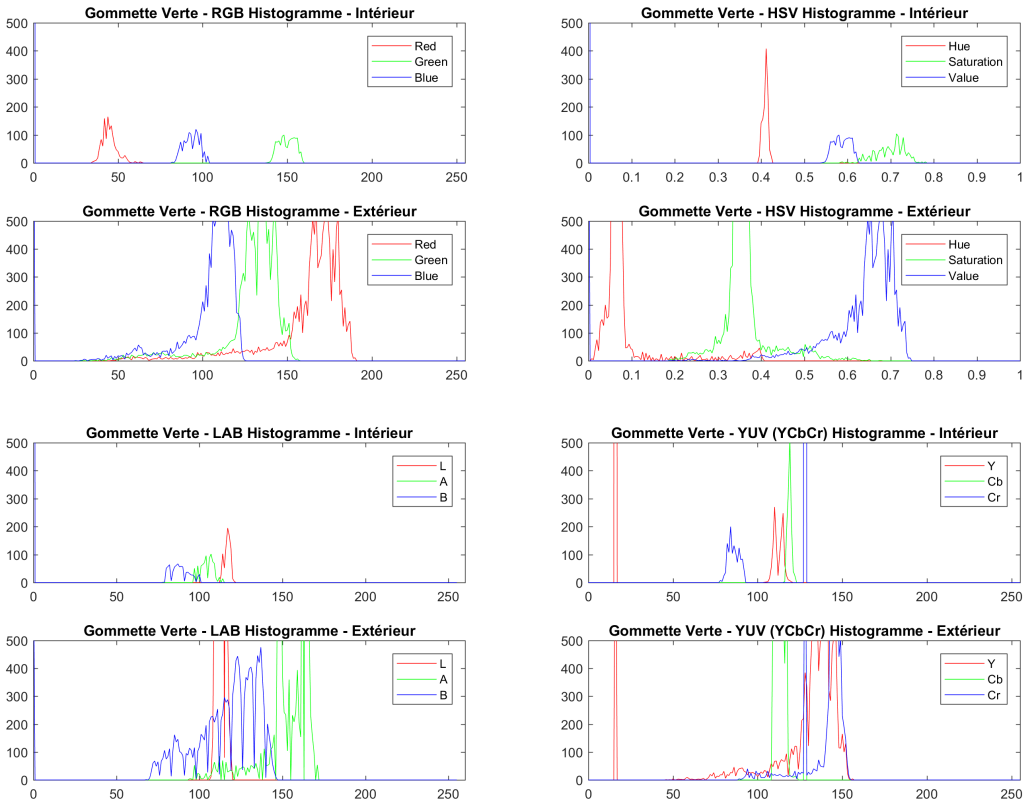


FIGURE C.2 – Gommette verte

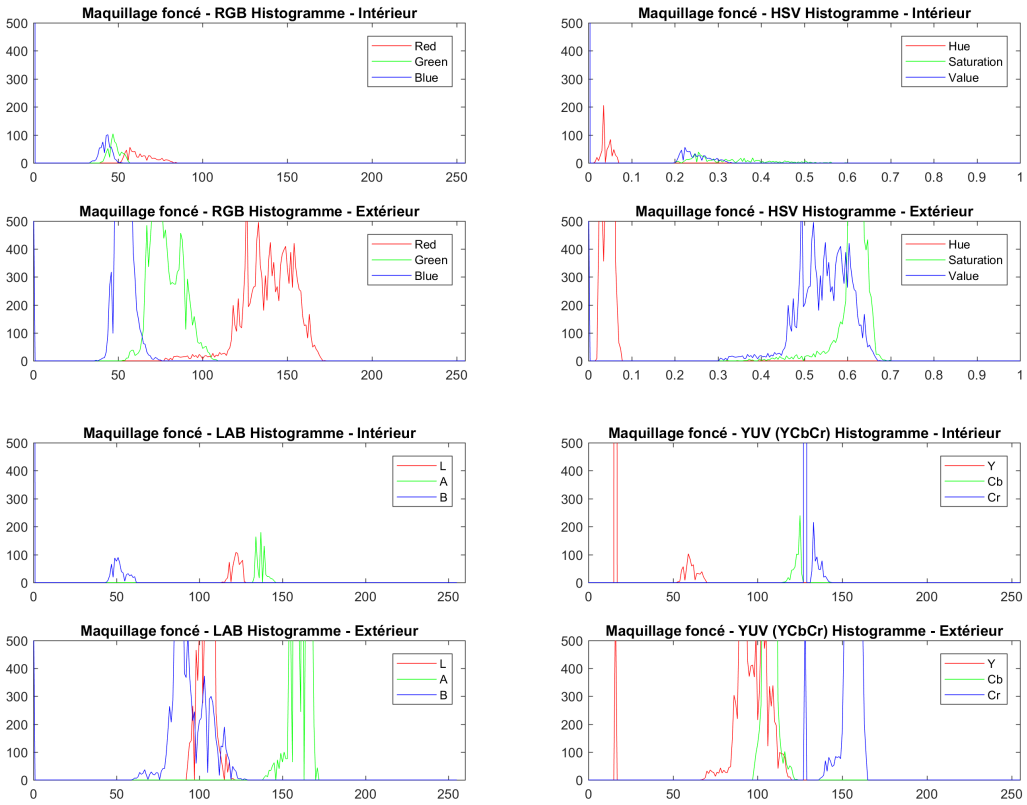


FIGURE C.3 – Maquillage bleu foncé

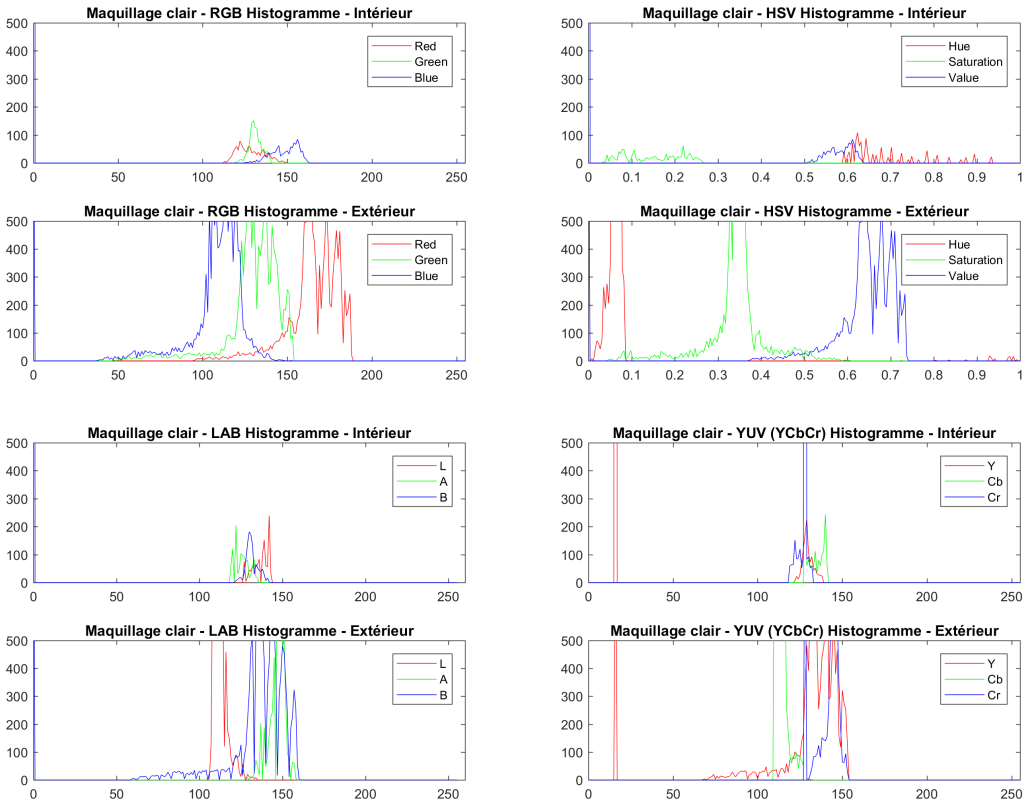


FIGURE C.4 – Maquillage clair

Annexe D

Stéréoscopie avec des caméras parallèles

Afin d'expliquer le fonctionnement de la stéréoscopie, nous pouvons nous intéresser au cas de deux caméras parallèles et co-planaires (dans le même plan). Cette configuration est présentée dans la Figure D.1. [49, 50]

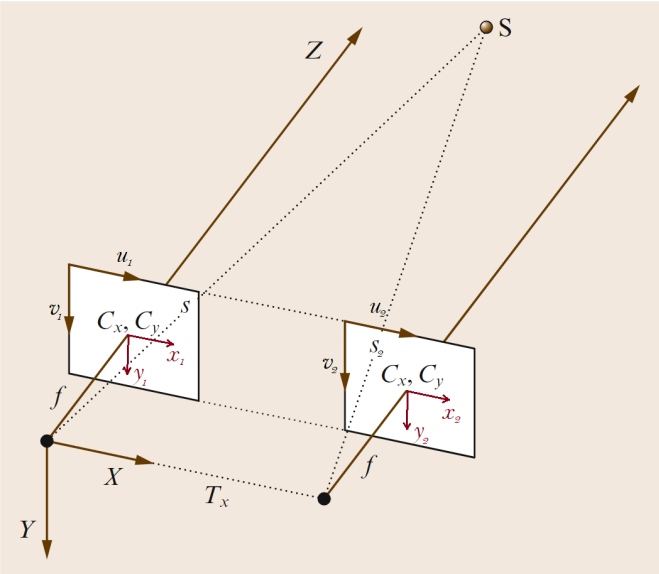


FIGURE D.1 – Stéréovision à caméras parallèles (image remaniée de [50])

Dans cette disposition :

— Les plans "images" des caméras sont parallèles

— Les centres optiques des caméras sont situés au même niveau (décalage uniquement selon l'axe $\overrightarrow{X}$)
— Les longueurs focales des caméras sont identiques (f)

Pour déterminer la profondeur de l'image, on utilise les triangles semblables (voir Figure D.2). Ce qui donne :

$$\frac{B}{Z} = \frac{B + x_1 - x_2}{Z - f}$$

Où B est la distance entre les centres optiques des caméras, z la profondeur de l'image, f la longueur focale des caméras et x_1 et x_2 la position de la projection du point P sur les deux plans image.

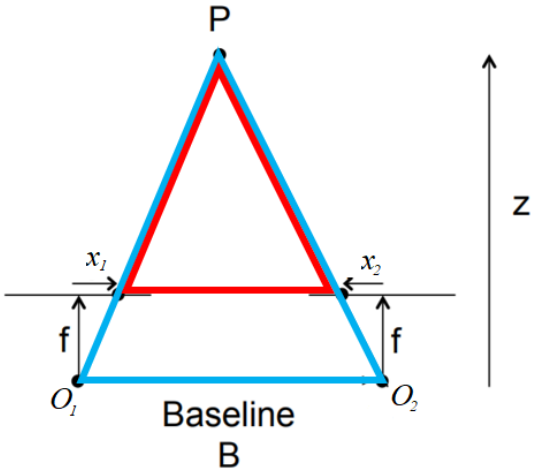


FIGURE D.2 – Cas de caméras parallèles : vue du dessus. On utilise le fait que les triangles rouge et bleu soient semblables pour déterminer la profondeur Z du point P. (image remaniée de [49])

En développant cette dernière équation et en réutilisant la règles de triangles semblable pour la première caméra, on trouve les relations suivantes dépendant de la différence $D = x_1 - x_2$ que l'on appelle la disparité :

$$Z = \frac{f_x \cdot B}{x_1 - x_2}$$
$$X = \frac{x_1 \cdot Z}{f_x}$$
$$Y = \frac{y_1 \cdot Z}{f_y}$$

Ce qui montre bien que nous retrouvons une coordonnée 3D grâce à la deuxième image. Notons que $\frac{Z}{f_x}$ et $\frac{Z}{f_y}$ jouent le rôle des coefficients d'agrandissement k_x et k_y respectivement.

Annexe E

Spécifications de la caméra stéréo

Specifications	VEN-134-90U3M-D
Resolution	1280 × 1024 × 2
Sensor Type	Onsemi PYTHON 1300, Global shutter CMOS
Optical Size	1/2 inch
Pixel Size	4.8μm × 4.8μm
Frame Rate	90fps @ 1280 × 1024 × 2
ADC Bit Depth	10bit
Pixel Bit Depth	8bit, 10bit
Shutter Time	5μs~1s
Gain	0dB~16dB
Pixel Data Formats	Mono 8/ Mono10
Signal to Noise Ratio	39dB
Definition	>951 lines
Synchronization	External trigger, Software trigger
I/O	2 GPIOs
Operating Temp.	0°C~45°C
Storage Temp.	-20°C~70°C
Operating Humidity	10%~80%
Power Consumption	<3.5W@5V
Data Interface	USB3.0
Regulations	CE, RoHS, USB3.0 Vision, GenICam

FIGURE E.1 – Spécifications de la caméra VEN-134-90U3M-D du fabricant GetCameras [44]

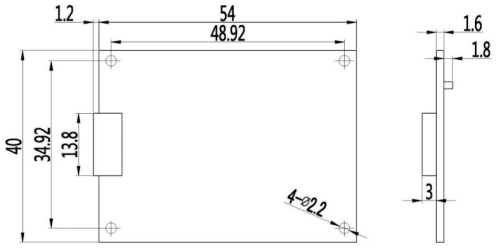


FIGURE E.2 – Dimensions de la carte de contrôle de la caméra VEN-134-90U3M-D [44]

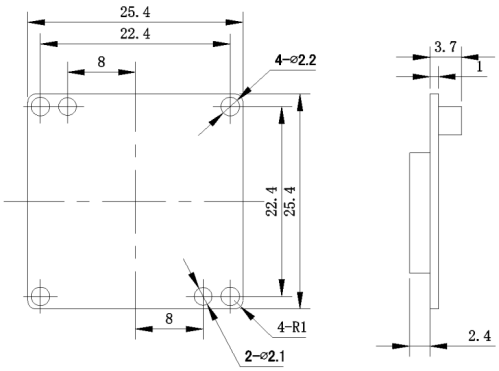


FIGURE E.3 – Dimensions des capteurs d'image de la caméra VEN-134-90U3M-D [44]

Annexe F

Résultat de la calibration des caméras stéréos

Pour rappel :

$$\mathbf{C} = \begin{bmatrix} f_x & 0.0 & c_x \\ 0.0 & f_y & c_y \\ 0.0 & 0.0 & 1.0 \end{bmatrix} \quad \mathbf{D} = \begin{bmatrix} k_1 & k_2 & p_1 & p_2 & k_3 \end{bmatrix}$$

Où f_x et f_y sont appelées distances focales, c_x et c_y sont la coordonnée du centre optique dans l'image, k_i et p_i sont les coefficients de distorsion (voir sous-section 3.1.2). La calibration des caméras stéréos placées dans leur emplacement sur le casque donne :

$$\mathbf{C_1} = \begin{bmatrix} 6.92e+02 & 0.0 & 4.96e+02 \\ 0.0 & 7.10e+02 & 6.25e+02 \\ 0.0 & 0.0 & 1.0 \end{bmatrix} \quad \mathbf{C_2} = \begin{bmatrix} 6.92e+02 & 0.0 & 5.40e+02 \\ 0.0 & 7.10e+02 & 6.33e+02 \\ 0.0 & 0.0 & 1.0 \end{bmatrix}$$

$$\mathbf{D_1} = \begin{bmatrix} 4.88e-02 & -1.25e-01 & 0.0 & 0.0 & 1.03e-01 \end{bmatrix}$$

$$\mathbf{D_2} = \begin{bmatrix} 3.27e-02 & -2.46e-02 & 0.0 & 0.0 & -3.37e-02 \end{bmatrix}$$

Remarquons que les coefficients de distorsions dans $\mathbf{D}_1$ et $\mathbf{D}_2$ sont très faibles. Ceux-ci pour rappel multiplient $r = \sqrt{x^2 + y^2}$ où x et y sont normés par la profondeur Z_c, ce qui donnera généralement des valeur $r < 1$. Nous en concluons la quasi absence de distorsion dans l'image grâce aux optiques "sans distorsion" utilisés.

Les matrices de rotation et translation obtenues entre les caméras sont les suivantes :

$$\mathbf{R} = \begin{bmatrix} 9.99e-01 & -5.70e-03 & -5.79e-03 \\ 7.75e-03 & 8.84e-01 & 4.67e-01 \\ 2.46e-03 & -4.67e-01 & 8.84e-01 \end{bmatrix}$$

$$\mathbf{T} = \begin{bmatrix} 3.42e-01 & -3.77e+01 & 1.02e+01 \end{bmatrix} [mm]$$

Remarquons que $\mathbf{T}$ donne une translation entre les deux capteurs très proche de l'écartement théorique entre ceux-ci. Si nous considérons que la matrice de rotation $\mathbf{R}$ est se rapproche d'une rotation pure selon l'axe $\overrightarrow{X}$ d'un angle θ :

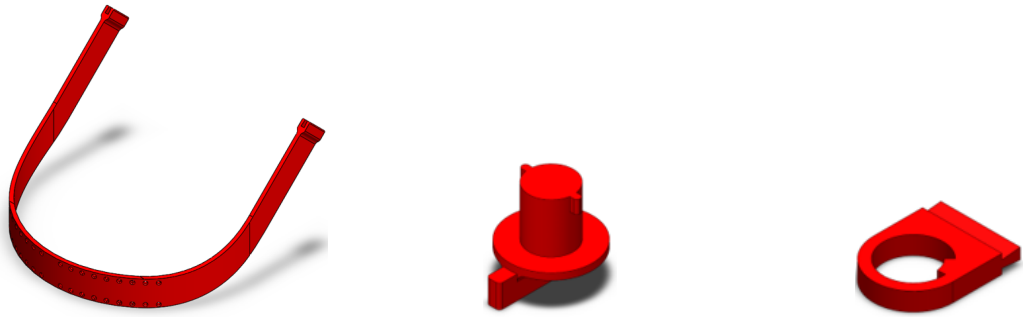
$$\mathbf{R} \approx \begin{bmatrix} 1 & 0 & 0 \\ 0 & cos\theta & sin\theta \\ 0 & -sin\theta & cos\theta \end{bmatrix}$$

Nous avons donc $cos\theta \approx 8.84e-01$ et donc $\theta \approx 27.87°$, ce qui est relativement proche des 32° théoriques.

Annexe G

Pièces 3D du premier prototype du casque du LMTS 2.0

Ces pièces font références au casque présenté dans la Section 3.5.3.



(a) Support caméra (b) Vis casque (c) Plug casque

FIGURE G.1 – Pièces imprimées en 3D de la première version

La "vis" permet d'attacher le "plug" au casque comme montré sur la Figure G.2. Cette dernière pièce comporte une queue d'aronde permettant d'y attacher le "support caméra" (FigureG.1a). Elle comporte également une butée permettant de bloquer l'angle de la pièce par rapport au casque dans une "position base" lorsque le dispositif est en cours d'utilisation (voir Figure G.3a) ainsi qu'une "position haute" qui relève le "support caméra" (voir Figure G.3b), ce qui est utile lors de l'installation ainsi que du rangement du dispositif. Le "support caméra" possède une série de trous permettant de tester plusieures dispositions horizontales et verticales pour nos capteurs d'image. Dans cette version, le boîtier protégeant la carte de contrôle est scotché sur le casque.

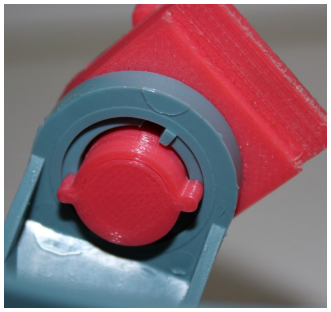
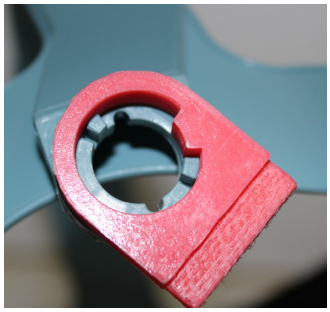
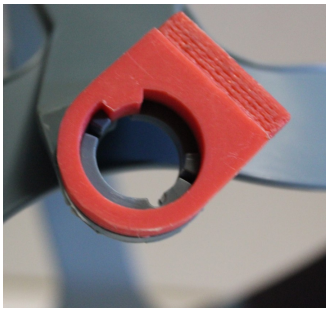


FIGURE G.2 – Vis maintenant le plug attaché au casque



(a) Position base



(b) Position haute

FIGURE G.3 – Positions du plug

Annexe H

Listing des pièces composant le casque du LMTS 2.0

Nom	Origine	Description	Quantité
Caméra stéréo	GetCameras	référence : VEN-134-90U3C-D. Options : câbles de 460mm, boîtier C-mount.	1
Lentille	GetCameras	référence : LM12-5MP-03MM-F2.8-2-ND1.	2
Adaptateur C mount vers M12	GetCameras	référence : LADAP-C-TO-M12-V2. Permet de monter les lentilles sur les boitiers C-mount.	2
Câble usb 3.0	GetCameras	référence : CABLE-D-USB3-5M. Câble usb 3 de 4.60 m pour brancher la caméra stéréo sur un ordinateur.	1
Ruban LED	"Action"	Ruban LED de 1M de long avec un boitier pour 3 piles AA.	1
vis M3x5	"Brico"	Permet de fixer les capteurs d'image sur le casque.	4 ou 8
vis M4x20 tête hexagonale	"Brico"	Permet de fixer les différentes parties du casque ensemble - à combiner avec "vis casque".	4
vis M4x30 tête hexagonale	"Brico"	Permet de fixer le cerclage au casque Honeywell.	2
casque Honeywell	RS	Référence : Honeywell 1002297 Face Shield Helmet. Casque à placer sur la tête du patient.	1

Nom	Origine	Description	Quantité
écrou M4	"Brico"	S'emboîtent dans les pièces "vis bout", "attache casque" et "pièce bouche".	6
cerclage	3D	Pièce faisant le tour de la tête du patient.	1
pièce bouche	3D	Support des capteurs d'image.	1
fpga box	3D	Protège et fixe la carte de contrôle de la caméra stéréo.	1
Attache casque G	3D	Plug gauche du casque.	1
Attache caque D	3D	Plug droit du casque.	1
Bout vis	3D	Accueille un écrou M4 pour y fixer les plug's du casque.	2
vis casque	3D	Transforme les vis M4 à tête hexagonales à vis moletées.	6
support ruban led	3D	Se clipse sur la pièce bouche pour positionner l'éclairage LED.	2
clip 27x8 mm	3D	Se clipse sur le cerclage et la pièce bouche pour fixer les câbles proprement.	10
capuchon caméra	3D	Protège les optiques des caméras lorsque le dispositif n'est pas utilisé.	2

Annexe I

Structure du code

La structure du code formant le LMTS Studio 2.0 est réprésentée à la page suivante sous la forme d'un diagramme de classe (Figure I.1). Celui-ci comprend à la fois la partie "traitement d'image" et la partie "interface graphique".

L'interface graphique est principalement définie par la classe ***mainWindow*** qui correspond à la fenêtre principale de l'interface graphique. La structure contient une instance des trois différentes fenêtres de paramètres qui peuvent être ouvertes à partir de la fenêtre principale :

cameraSettingsDialog pour les réglages de la caméra stéréo.

AudioRecorder pour l'enregistrement audio.

settingsDialog pour les paramètres liés au traitement d'image.

Le traitement d'image tourne quant à lui principalement autour de la structure ***ImageData*** qui contient tous les paramètres du traitement d'image (en général). Cette structure de donnée contient également un tableau de ***MarkerPlace*** qui rassemble toutes les informations de position, couleur et paramètres de traitement propre à chaque marqueur suivi.

TraitementVideoThread est la classe qui implémente le traitement lorsqu'il s'agit d'une vidéo pré-enregistrée. La communication avec les threads "marqueurs" ***GomThread*** se fait via le buffer ***GomBuffer*** qui implémente le producteur - multi-consommateur. Celui-ci contient un tableau de 10 ***BufferItem*** et un tableau de maximum 20 ***GomItem*** (un pour chaque ***GomThread***). Chaque ***BufferItem*** se compose d'un ***ImgInput*** (image à traiter et informations liées à cette image) et d'un ***ImgOutput*** (résultat du traitement).

Lorsque l'image d'une caméra est utilisée, ***TraitementThread*** remplace ***TraitementVideoThread***. Celui-ci récupére les images à traiter via le buffer ***ImgBuffer***

qui est rempli soit par ***WebcamThread*** dans le cas d'une webcam, soit par ***GetCameraHandler*** dans le cas de la caméra stéréo.

La queue permetant de stocker les images de la caméra dans un thread séparé (***StoreThread***) est implémentée par ***ImgQueue***.

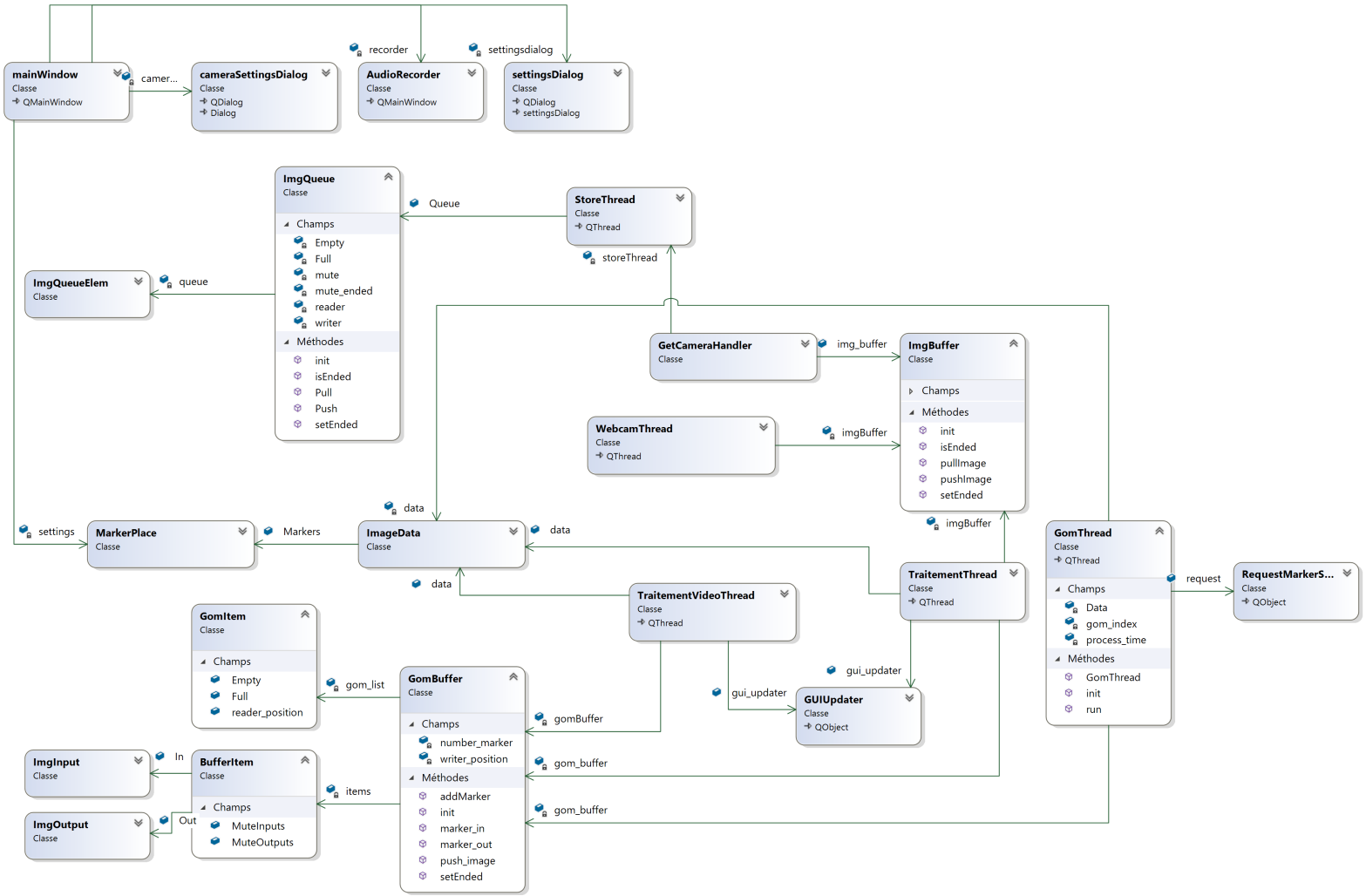


FIGURE I.1 – Diagramme de classe de l'application LMTS Studio 2.0

Annexe J

Mode d'emploi du LMTS Studio 2.0

J.1 Installation

L'application LMTS Studio 2.0 en tant que tel n'a pas besoin d'une installation particulière une fois le dossier de setup téléchargé sur l'ordinateur (Figure J.1). Tout au plus, un raccourci vers l'application peut être fait et copié à l'endroit désiré [1].

Si la caméra stéréo doit être utilisée, il est par contre nécessaire d'installer les pilotes de celle-ci.

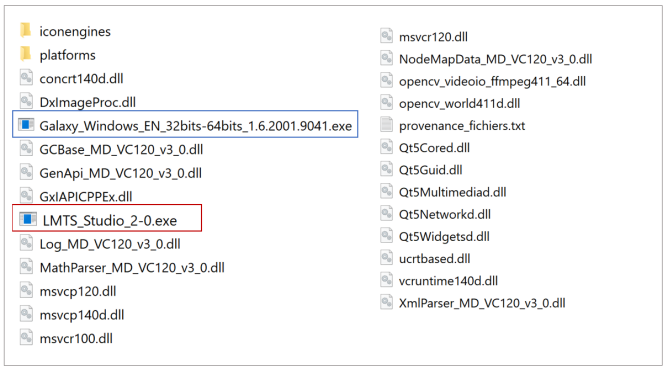


FIGURE J.1 – Dossier de setup du LMTS Studio 2.0. Encadré en rouge : l'application du LMTS Studio 2.0. Encadré en bleu : l'installateur de la caméra stéréo.

1. Pour faire un raccourci : clic droit sur l'application (cadre rouge dans la Figure J.1) puis "créer un raccourci".

Note sur le type de fichier vidéo à traiter : Le type de fichier normalement pris en compte par le LMTS Studio 2.0 reprend tous les fichiers dont l'extension est ".avi". Les fichiers enregistrés par la GOPRO ("MP4") doivent donc être au minimum convertis en ".avi" (et la distorsion corrigée si possible). Dans le cas où un fichier ".MP4" ou autre doit être traité et provoque une erreur dans le LMTS Studio 2.0, supprimer ou renommer provisoirement le fichier "opencv_videoio_ffmepeg411_644.dll" peut corriger le problème. Celui-ci doit cependant être remis en ordre après afin de pouvoir enregistrer ou traiter une vidéo en ".avi".

Note en cas de problème de mise à l'échelle : Malgré le soin apporté pour rendre l'application LMTS Studio 2.0 compatible avec tous les ordinateurs Windows, des problèmes de mise à l'échelle peuvent toute fois survenir. Si c'est le cas, faites un clic droit sur l'application. Dans propriétés, allez dans compatibilité et cliquez sur "modifier les paramètres de PPP élevés". Dans la fenêtre qui apparaît, cochez la case "Remplacer le comportement de mise à l'échelle PPP élevée. Mise à l'échelle effectuée par :" et sélectionnez "Système" avant de valider.

Installation des pilotes de la caméra stéréo

Les étapes suivantes ne sont nécessaires que si le LMTS Studio 2.0 doit être utilisé en combinaison de la caméra stéréo.

Pour ce faire, il faut exécuter le programme d'installation encadré en bleu dans la Figure J.1. **Notons que pour que l'installation réussisse, la caméra stéréo doit être branchée via son câble à un port USB3 de l'ordinateur !**

La fenêtre Figure J.2 apparaît : cliquez sur "Next". Ensuite acceptez la licence et cliquez sur "Next" (Figure J.3).

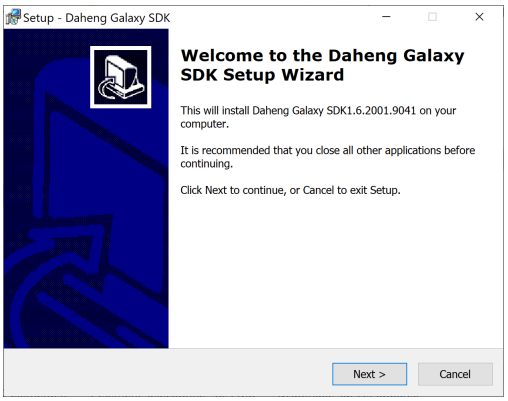


FIGURE J.2 – Programme d'installation de la caméra.

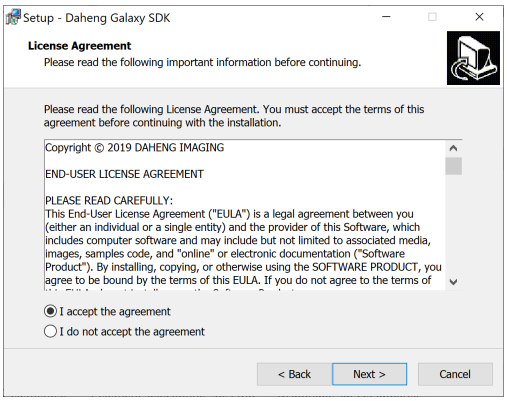


FIGURE J.3 – Acceptation de la licence.

Si désiré, l'emplacement d'installation peut être personnalisé (Figure J.4), puis cliquez sur "Next". Uniquement la composante "USB3 Vision Camera" est nécessaire (Figure J.5), les autres cases peuvent être dé-cochées avant de cliquer sur "Next".

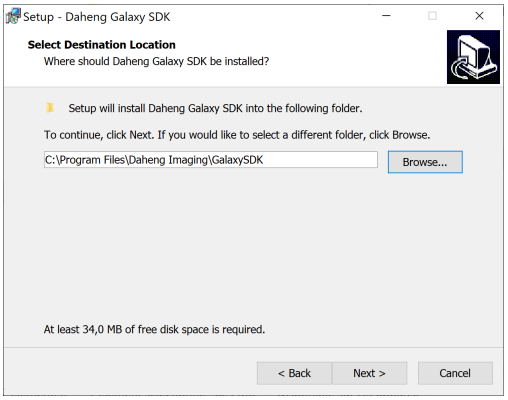


FIGURE J.4 – Choix de l'emplacement d'installation.

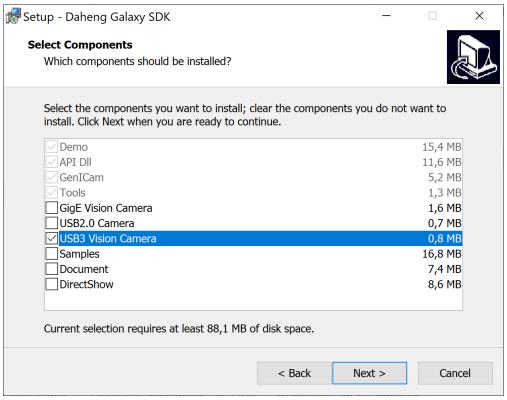


FIGURE J.5 – Choix des composante : seule "USB3" est nécessaire.

Cliquez deux fois sur "Next" (Figure J.6 et J.7).

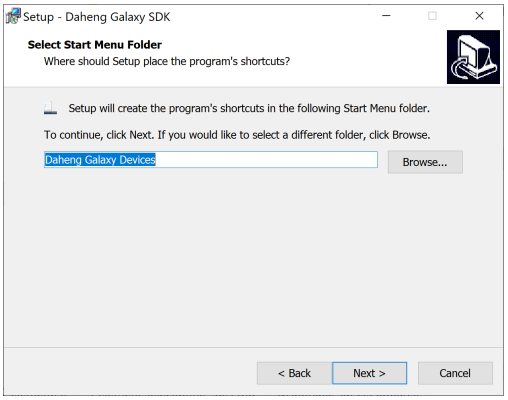


FIGURE J.6

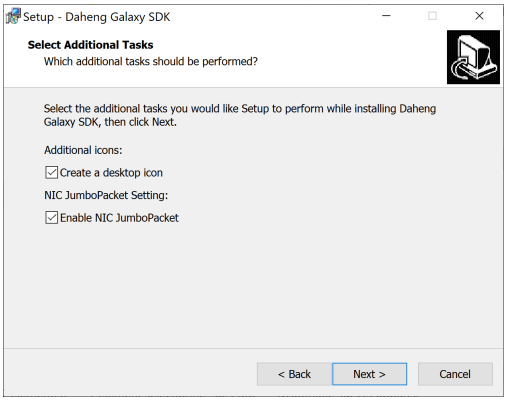


FIGURE J.7 – Les deux cases peuvent être dé-cochée si pas nécessaire.

Enfin, les mêmes composantes devraient être affichées (Figure J.8). Cliquez sur "Install", l'opération ne devrait pas durer plus de quelques secondes. Enfin, il sera nécessaire de redémarrer l'ordinateur pour compléter l'installation et pouvoir utiliser la caméra. Cliquer sur "Finish" (Figure J.9).

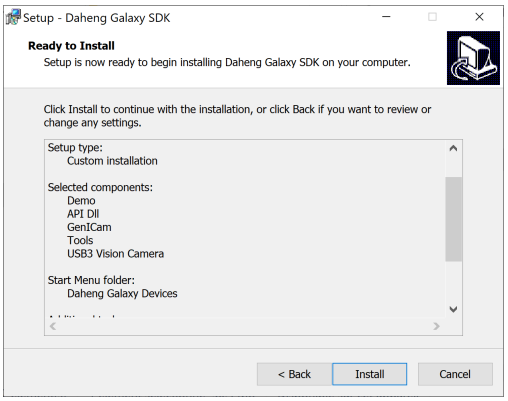


FIGURE J.8 – Vérification finale.

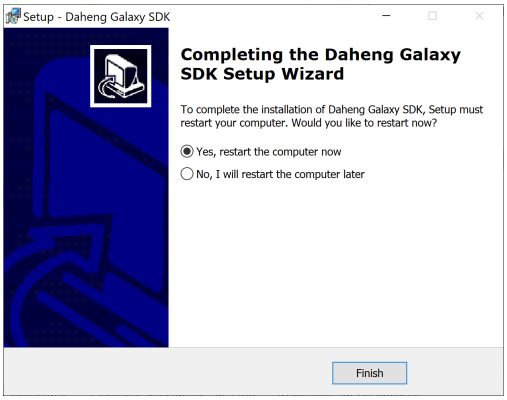


FIGURE J.9 – L'installation nécessite un redémarrage.

J.2 Description de l'interface graphique

Cette section décrit les différents éléments de l'interface graphique présentée dans la Figure 4.8.

Fenêtre vidéo

Cette fenêtre permet de visualiser le flux vidéo que l'utilisateur souhaite traiter. Elle est également cliquable, ce qui est utile dans certaines étapes du traitement de la vidéo. Par exemple, pour indiquer l'emplacement initial des différents marqueurs.

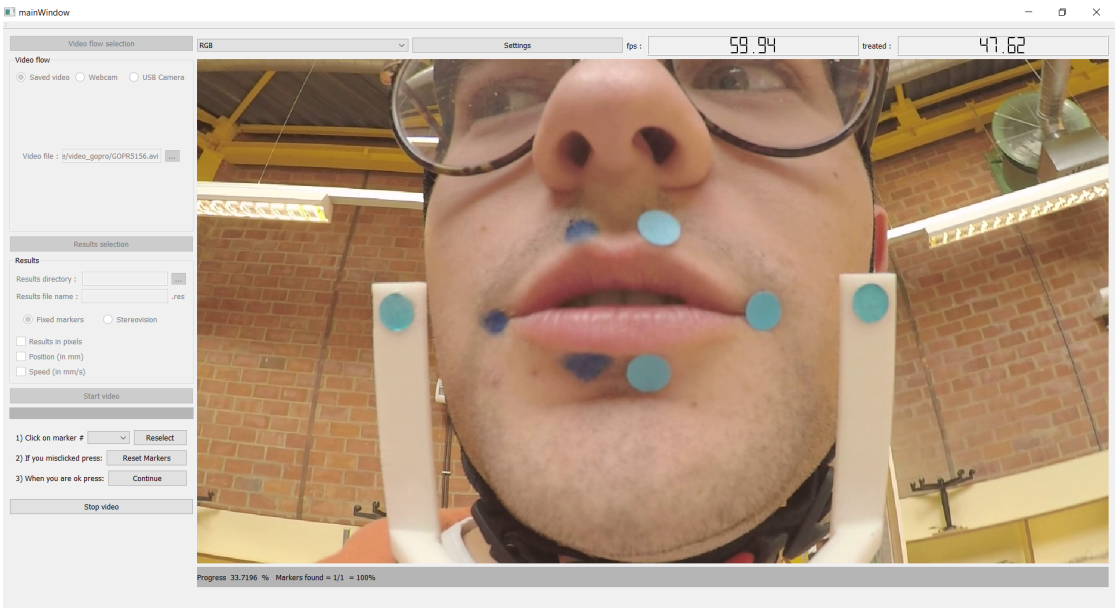
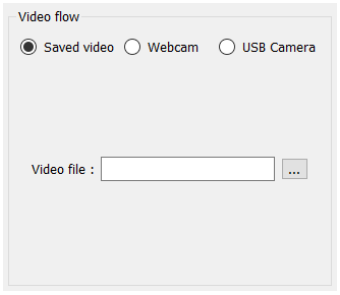


FIGURE J.10 – Fenêtre vidéo

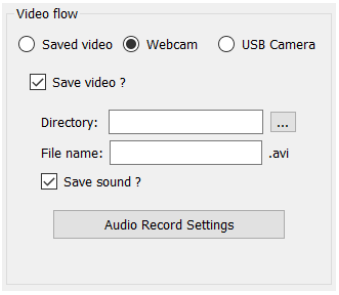
Choix du flux vidéo

Notre application peut traiter 3 types de flux vidéo différents :

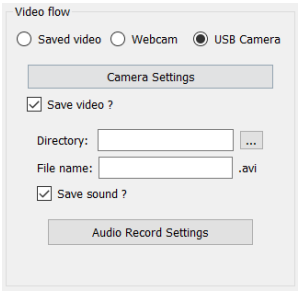
- Des vidéos enregistrées au format ".avi" (ou ".mp4" voir la note dans la section précédente)
- Un flux vidéo venant d'une webcam
- Un flux vidéo venant de la caméra stéréo (branché par USB 3.0) de notre dispositif



(a) Vidéo enregistrée sur l'ordinateur



(b) Webcam de l'ordinateur



(c) Caméra USB de notre dispositif actuel

FIGURE J.11 – Champ "choix du flux vidéo"

Vidéo enregistrée : Lorsque l'on sélectionne le bouton "Saved video", le champ "choix du flux vidéo" aura la forme de la Figure J.11a. Il suffit alors d'appuyer sur le bouton [...] afin de naviguer dans les fichiers de l'ordinateur et sélectionner la vidéo à traiter. A noter que celles-ci doivent être au format ".avi" ou ".mp4".

Un flux vidéo venant d'une webcam : Lorsque le bouton "Webcam" est sélectionné, le champ affichera la Figure J.11b. Nous offrons la possibilité d'enregistrer la vidéo issue de la webcam ainsi que le son de celle-ci. Il est possible de choisir le dossier dans lequel le fichier sera enregistré ainsi que le nom de celui-ci. Le fichier vidéo ainsi obtenu sera sous format .avi. Si l'option enregistrer le son est cochée, il est possible de changer les options liées au son en cliquant sur le bouton "Audio Record Settings". Une fenêtre (voir Figure J.12) s'ouvrira alors afin de pouvoir changer les paramètres suivants :

— Le choix du micro
— Le choix du codec audio
— Le choix du format conteneur
— Le choix de la fréquence d'échantillonage
— Le choix du nombre de canaux
— Le choix du mode d'encodage

Ce choix de flux vidéo n'est pas applicable pour effectuer un suivi du mouvement des lèvres étant donné que la position webcam n'est pas solidaire de la tête. Cependant, cela reste intéressant afin réaliser des tests de maquillage sans devoir installer le dispositif complet sur la tête d'un patient.

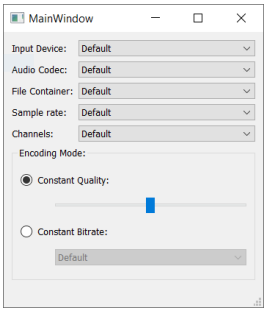


FIGURE J.12 – Fenêtre des options pour l'enregistrement du son

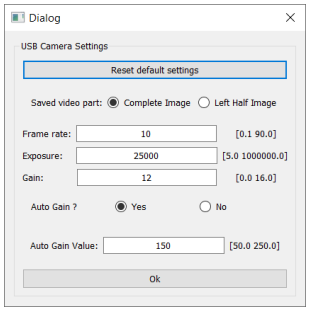


FIGURE J.13 – Fenêtre des options de l'image des caméras USB du dispositif

Un flux vidéo venant de la caméra stéréo de notre dispositif de stéréovision : En sélectionnant ce flux vidéo (voir Figure J.11c), les possibilités

d'enregistrer la vidéo et le son fonctionnent de la même façon que lorsque l'on choisit un flux vidéo venant d'une webcam. De plus, il est possible de modifier les paramètres des caméras USB en appuyant sur le bouton "Camera Settings". La fenêtre représentée sur la Figure J.13 s'ouvrira, offrant la possibilité de sélectionner :

— Si l'on enregistre les vidéos des deux caméras ou bien que d'une seule (celle devant la bouche)
— La vitesse d'acquisition des images
— Le gain de l'image
— Si l'on applique un gain automatique sur l'image
— La valeur du gain automatique

Résultats

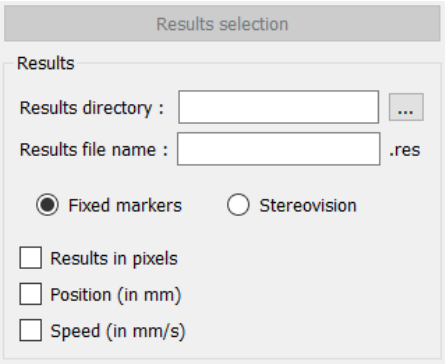


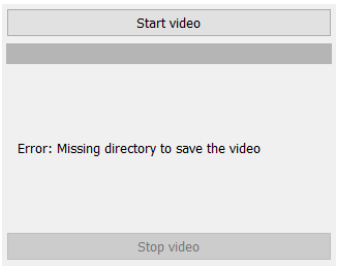
FIGURE J.14 – Champ "résultats"

Comme montré sur la Figure J.14, il est possible de sélectionner le dossier dans lequel les résultats seront enregistrés ainsi que le nom du fichier résultat. Puis, il faut sélectionner si le flux vidéo est en monovision avec des marqueurs fixes visibles à l'image (afin de faire la conversion des positions en pixels en millimètres) ou alors si le flux est en stéréovision et donc ne nécessite pas d'avoir de marqueurs fixes afin de faire une conversion des pixels en millimètres. Ensuite, il suffit de cocher une ou plusieurs des cases situées dans la partie inférieure du champ. Les choix, "Results in pixels", "Position (in mm)" et "Speed (in mm)" permettent respectivement d'enregistrer les positions des marqueurs sur l'image vidéo en pixels, d'enregistrer les positions en millimètre et enfin d'enregistrer les vitesses des marqueurs. Il est à noter que lorsqu'aucune de ces cases ne sont cochées, aucun fichier résultat ne sera créé.

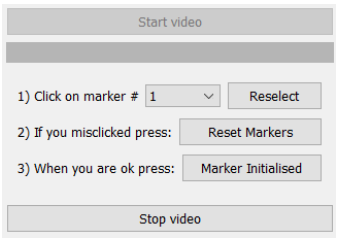
Note importante : Si la position en *mm* des marqueurs doit être enregistrée sur base de la stéréovision, la sélection des marqueurs doit être faite de manière

séquentielle : pour chaque marqueur, il faut le sélectionner dans l'image de gauche puis dans l'image de droite de sorte que tous les numéros impairs soient dans l'image gauche et tous les numéros pairs dans l'image de droite. Ceci est essentiel pour que la transformation de coordonnée se fasse correctement.

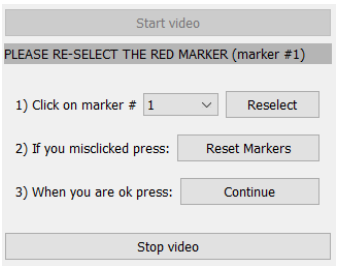
Lancement/arrêt vidéo



(a) Champ bloqué en présence d'erreur



(b) Champ lors de l'initialisation des marqueurs



(c) Champ en cas de marqueur perdu

FIGURE J.15 – Champ de "lancement/arrêt" de la vidéo à traiter

Lancement erroné de la vidéo : Pour pouvoir lancer le traitement de la vidéo, il suffit d'appuyer sur le bouton "Start video". Si toutes les informations nécessaires au lancement n'ont pas été remplies, un message d'erreur apparaîtra (voir Figure J.15a), interrompant le lancement du traitement.

Initialisation des marqueurs : Une fois toutes ces informations fournies, il est possible d'initialiser les marqueurs dans la fenêtre vidéo. Pour ce faire, il suffit de cliquer sur les différents marqueurs (voir Figure J.16). Il n'est pas nécessaire de sélectionner précisément le centre du marqueur car cela sera réglé automatiquement. Les différents marqueurs sont affichés avec un disque vert de rayon proportionnel à leur taille à l'image et numérotés dans l'ordre de leur initialisation comme illustré sur la Figure J.17.

Dans le cas où un marqueur n'a pas été correctement initialisé, il suffit de sélectionner le numéro de celui-ci dans la liste déroulante (voir Figure J.15b), d'appuyer sur le bouton "Reselect" et de resélectionner le marqueur. Il est aussi possible de réinitialiser tous les marqueurs en une fois en appuyant sur "Reset Markers". Une fois l'intégralité des marqueurs initialisés, il suffit d'enclencher le bouton "Marker initialized" pour lancer le traitement.

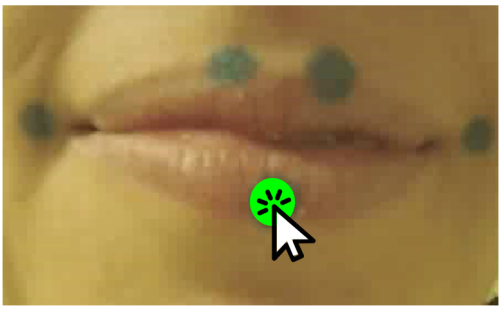


FIGURE J.16 – Sélection de marqueur

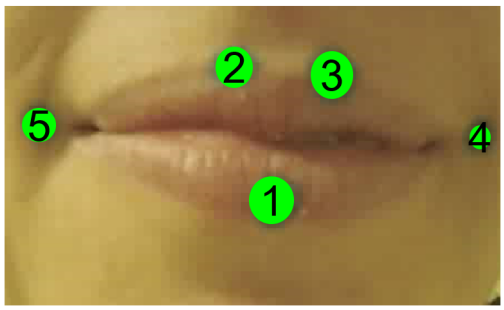


FIGURE J.17 – Marqueurs initialisés

Marqueur perdu : Au cours du traitement de la vidéo, il est possible qu'un marqueur soit perdu en cours de route. Celui-ci sera affiché en bleu (voir Figure J.18). Dans le cas où le le marqueur est perdu pendant 60 images consécutives, celui-ci sera affiché en rouge (comme montré sur la Figure J.19). Le traitement sera alors mis en pause et un message demandant de resélectionner le marqueur s'affichera juste en dessous du bouton "Start video" (comme illustré sur la Figure J.15c). Une fois cela fait, il suffit d'appuyer sur le bouton "Continue" pour poursuivre le traitement.

Il est également possible de de resélectionner des marqueurs en cours de traitement. Pour ce faire, il suffit d'appuyer sur le bouton "Pause" (le même que le bouton "Continue" de la Figure J.15c dont le nom change en cours de traitement de la vidéo). Ensuite, il faut choisir le marqueur à resélectionner dans la liste déroulante, appuyer sur le bouton "Reselect" et appuyer sur "Continue" pour poursuivre le traitement vidéo.

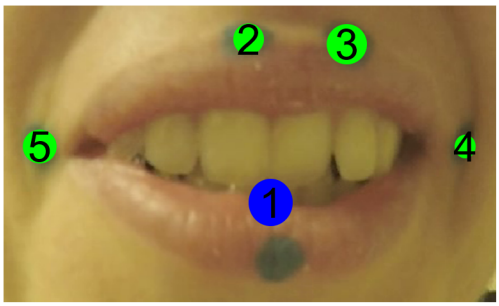


FIGURE J.18 – Marqueur perdu (en bleu)

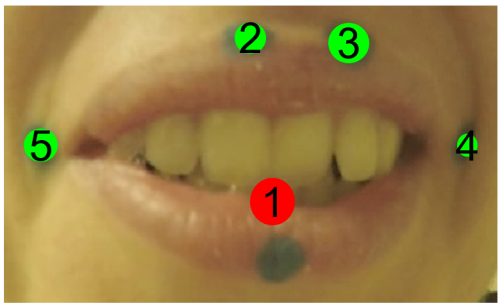


FIGURE J.19 – Marqueur perdu pendant plus de 60 frames (en rouge)

Fonction secondaire de l'option "pause" et "continue" Dans le cas du traitement du flux vidéo venant d'une caméra, le bouton "pause" / "continue"

permet de stopper momentanément le stockage des résultats du traitement. Le flux vidéo et le suivi continuant malgré tout. Cela peut être une option intéressante pour limiter la taille des fichier résultats et/ou faciliter la séparation des sous-expériences. Le praticien peut ainsi, si il le souhaite, interrompre l'enregistrement et le reprendre uniquement pendant les courts moments d'intérêt. Notons que l'option "pause" n'interrompt pas l'enregistrement vidéo et audio. Les informations de temps et de numéro d'image stockées dans chaque fichier résultat permettra de localiser aisément les "coupures" faites.

Choix de l'espace de couleur

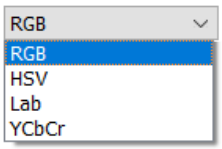
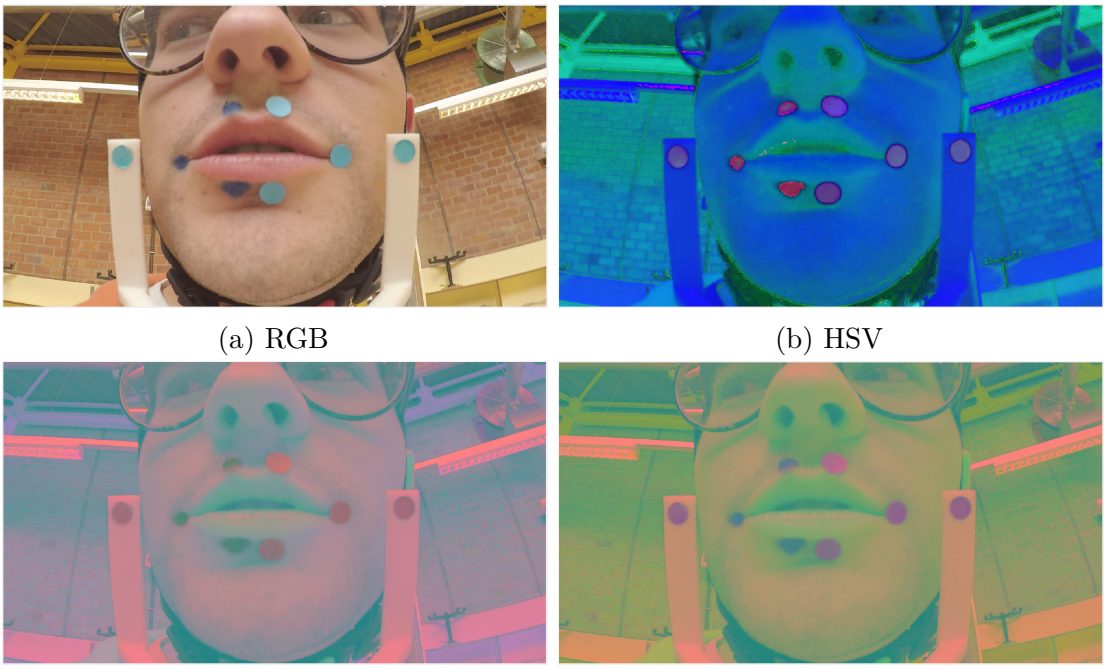


FIGURE J.20 – Liste déroulante du choix de l'espace de couleur

La liste déroulante de la Figure J.20 permet de changer l'espace de couleur dans lequel sera montré les images dans la fenêtre vidéo. La Figure J.21 permet de voir la même image dans les différents espaces de couleur disponibles dans notre application. Cette fonctionnalité n'est là que pour se rendre compte du contraste entre la peau et le maquillage choisi pour dessiner les marqueurs sur le visage. En effet, celui-ci sera différent en fonction de l'espace de couleur choisi. Il est à noter que cet outil n'est présent qu'à des fins de visualisation et ne change pas l'espace de couleur pour le traitement. Pour ce faire, il faut se rendre dans la fenêtre des options du traitement d'images que nous détaillerons au point suivant.



(a) RGB (b) HSV

(c) Lab (d) YCbCr

FIGURE J.21 – Visualisation des différents espaces de couleur

Options du traitement d'image

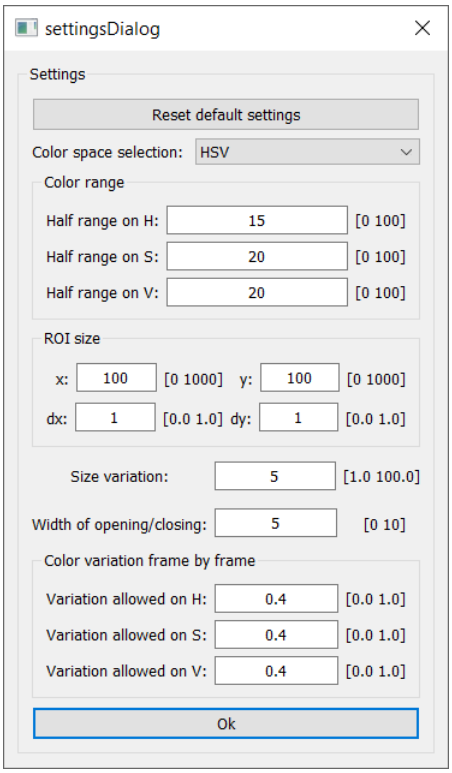


FIGURE J.22 – Fenêtre des options pour le traitement d'images

La Figure J.22 illustre la fenêtre des options du traitement des images. Nous avons fait en sorte que tous les paramètres intervenant dans le traitement puissent être modifiés. Ceux-ci seront détaillés ci-après.

Choix de l'espace de couleur : Comme expliqué au point précédant, choisir un espace de couleur permettant un grand contraste entre le marqueur et la peau est très important. Plus celui-ci sera important, plus il sera facile pour l'algorithme de discriminer les marqueurs par rapport à la peau.

L'intervalle de couleur : Les différents pixels formant les marqueurs dans l'image n'auront pas exactement la même couleur. Cela est dû à divers facteurs comme les ombres à l'image, un maquillage pas totalement homogène, etc. C'est pour cela que pour discriminer les marqueurs, il faut sélectionner les pixels d'un certain intervalle de couleur.

La taille de la région d'intérêt : Afin de réduire le temps calcul nécessaire à retrouver les différents marqueurs, chacun de ceux-ci seront recherchés dans des zone réduites autour de chaque marquer appelée région d'intérêt (ou ROI en anglais). Celles-ci ont une taille initiale $[x, y]$ qui peut être augmentée momentanément durant le traitement. Les ROI's se déplacent afin de suivre le mouvement des marqueurs. dx et dy sont les coefficients de la moyenne mobile faite entre la nouvelle et l'ancienne position pour filtrer les déplacements les plus rapides ($dx = dy = 1$: la position retenue est égale à la nouvelle position trouvée. $dx = dy = 0$: la position du marqueur reste fixe).

La variation autorisée sur la taille du marqueur : La principale façon de déterminer si un marqueur est perdu est de se baser sur la variation de sa taille. Si la partie détectée devient très grande, c'est sûrement dû au fait que ce n'est plus le marqueur qui est détecté mais bien la peau. Il faut alors choisir une valeur autorisant les variations de taille venant du mouvement des lèvres tout en limitant celles venant d'une erreur de traitement.

La largeur de l'ouverture/fermeture : Comme expliqué dans la section parlant des outils de traitement d'images, l'opération d'ouverture/fermeture permet de filtrer un marqueur afin d'obtenir des bords plus nets. Si ce paramètre est trop petit, le marqueur comportera des trous et si ce paramètre est trop grand, la forme détectée du marqueur ne sera plus exactement la même que celle du marqueur réel.

La variation de couleur autorisée : Différents éléments extérieurs peuvent changer la couleur d'un marqueur à l'image comme les ombres ou la variation de luminosité par exemple. C'est pour cela qu'il faut que notre algorithme adapte la couleur de marqueur qu'il recherche. Cependant, il se peut que l'on se mette à détecter la peau au lieu des marqueurs par exemple. C'est pour cela qu'il faut limiter la variation de la couleur recherchée. Ceci est fait par une moyenne mobile entre les valeurs précédentes et nouvelles (dont l'utilisateur peut choisir le coefficient α qui multiplie la nouvelle valeur).

$$\text{couleur} = \alpha \cdot \text{nouvelle couleur} + (1 - \alpha) \cdot \text{ancienne couleur}$$

Vitesse du flux vidéo et du traitement d'image

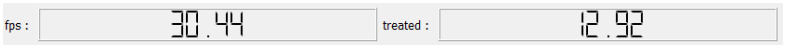


FIGURE J.23 – Champ "vitesse du flux vidéo et du traitement d'image"

La Figure J.23 présente deux informations du traitement vidéo. La partie de gauche, "fps", indique de nombre d'images par secondes délivré par le flux vidéo tandis que la partie de droite, "treated", indique la vitesse du traitement de celui-ci en nombre d'images traitées par seconde (ou le nombre d'images qui pourrait être traitée en une seconde lorsque le traitement est plus rapide que le flux vidéo).

Informations générales

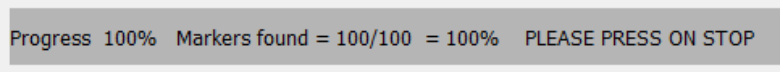


FIGURE J.24 – Champ "informations générales"

Le champ montré sur la Figure J.24 donne des informations durant le traitement des images. Plus précisément, des informations sur l'avancement du traitement de la vidéo, le pourcentage de marqueurs trouvés et des indications sur quel bouton appuyer en fin de traitement.

UNIVERSITÉ CATHOLIQUE DE LOUVAIN
École polytechnique de Louvain
Rue Archimède, 1 bte L6.11.01, 1348 Louvain-la-Neuve, Belgique | **www.uclouvain.be/epl**