MASTER EN SCIENCES INFORMATIQUES

Etude de la faisabilité d'un support automatisé à l'annotation de vidéos en langue des signes: Cas du Corpus LSFB

Lebutte, Jérémy; Smal, Anne

Award date:
2017

Awarding institution:
Universite de Namur

Université de Namur
Faculté d'informatique
Année académique 2016 - 2017

Étude de la faisabilité d'un support automatisé à l'annotation de vidéos en langue des signes :
Cas du Corpus LSFB

Jérémy LEBUTTE Anne SMAL

Maître de stage : Laurence MEURANT

Promoteur : _____ (Signature pour approbation du dépôt - REE art. 40)
Professeur Anthony CLEVE

Mémoire présenté en vue de l'obtention du grade de
Master en Sciences Informatiques.

Résumé

Avec l'évolution de l'informatique dans les années 2000, les corpus électroniques se développent ce qui permet aux chercheurs de disposer de meilleures bases pour la recherche en langue des signes. Cependant, les vidéos composant ces corpus doivent être annotées à la main, ce qui nécessite un temps conséquent. Dès lors, il serait intéressant d'avoir une solution permettant d'accélérer ce processus d'annotation en l'automatisant. Le but de ce mémoire est d'évaluer la faisabilité d'une telle solution.

Dans un premier temps, nous avons travaillé quatre mois au sein du Laboratoire de Langue des Signes de Belgique Francophone (LSFB-lab) de l'Université de Namur, où nous avons développé un système d'annotation automatique des vidéos composant leur corpus. Ensuite, nous avons testé notre solution sur un échantillon de vidéos avant de comparer nos résultats avec les annotations manuelles.

Les résultats obtenus semblent montrer qu'une solution visant à automatiser le processus d'annotation des vidéos composant le corpus du LSFB-lab n'est pas réalisable dans les conditions actuelles. Cependant, d'autres solutions sont envisageables pour accélerer ce processus d'annotation : l'une se basant toujours sur l'automatisation mais nécessitant d'enregistrer de nouvelles vidéos en tenant compte des limites des techniques d'automatisation, l'autre se basant sur un système collaboratif permettant de garder les vidéos actuelles.

Mots-clés : langue des signes - reconnaissance - automatisation

Abstract

Thanks to the evolution of computer science in the 2000s, electronic corpus are developed, allowing researchers to have better bases for sign language research. However, the videos composing these corpus must be annotated by hand. Thenceforward, it is interesting to have a solution to accelerate this annotation process by automation. In this memoir, we will evaluate the feasibility of such a solution.

We first worked for four months at the Laboratory of French Belgian Sign Language (LSFB-lab) of the University of Namur, where we developed a system to automate annotation of the videos that compose its corpus. Then we tested our solution on a sample of videos before comparing our results with the manual annotations.

The results obtained seem to show that a solution which aim to automate the annotation process of the videos composing the LSFB-lab corpus is not possible under the current conditions. However, other solutions are possible to accelerate this annotation process : one based on automation but requiring to record new videos taking into account the limitations of automation techniques, the other based on collaborative system to keep current videos.

Keywords : sign language - recognition - automation

Remerciements

Nous tenons à remercier les personnes qui nous ont aidés dans la réalisation de ce mémoire.

En premier lieu, nous voudrions remercier Anthony Cleve qui, en tant que promoteur, nous a suivis et guidés tout au long de notre stage et de notre mémoire.

Nous remercions également Laurence Meurant, notre maître de stage, pour nous avoir permis de réaliser ce stage, ainsi que les membres du LSFB-lab pour leur disponibilité quand nous avons eu besoin d'eux.

Merci à la faculté d'informatique et plus particulièrement au bureau 435 composé de Nessrine, Adrien, Loup et Maxime pour nous avoir accueillis pendant les quatre mois de notre stage.

Merci à Bruno Dumas pour son cours sur la reconnaissance gestuelle qui nous aura bien aidé. Merci à Delphine Nicolay qui nous a aidés à comprendre les aspects un peu trop mathématiques. Merci à Pierre Rousseau pour nous avoir dépannés face à certains problèmes techniques.

Merci à toutes les personnes qui nous ont soutenus dans l'écriture de ce mémoire et surtout aux personnes qui ont passé de nombreuses heures à la relecture de celui-ci.

Pour finir, un merci tout particulier à tous les professeurs dont nous avons croisé le chemin durant nos années d'études et qui nous ont tant appris.

Table des matières

Glossaire

Dans ce glossaire, nous définirons les termes importants nécessaires à la compréhension de ce mémoire. En outre, certains mots peuvent avoir plusieurs sens en fonction du contexte dans lequel ils sont utilisés. Nous donnerons ici le sens dans le contexte de ce mémoire.

Annotation : élément d'un fichier ELAN décrivant le tag du signe réalisé à un certain moment d'une vidéo.

Blob : groupe de pixels connectés dans une image binaire. Dans le cas d'étude, l'image binaire représentant les pixels de couleur peau, il s'agit d'un groupe de pixels connectés de couleur peau.

Corpus : collection de morceaux de langage qui sont sélectionnés et triés selon des critères linguistiques explicites afin d'être utilisés comme un échantillon du langage.

Coupure : moment dans une vidéo où les modérateurs interviennent. Ces moments sont représentés par un plan de couleur unie.

Espace de couleur : représentation tri-dimensionnelle de la couleur.

Format Bayer : format d'image obtenu en appliquant un filtre sur l'appareil de capture de sorte que chaque pixel ne capte que l'une des trois composantes du format RGB.

Fusion : évènement ayant lieu lorsque deux parties du corps se confondent visuellement sur l'image.

Histogramme : représentation de la distribution des couleurs au sein d'une image.

HSV : espace de couleurs représentant celles-ci à l'aide d'une composante de teinte (H), de saturation (S) et de luminosité (V).

Image binaire : image numérique pour laquelle chaque pixel n'a que deux valeurs possibles (généralement noir (0) et blanc(255)).

Langue des signes : langue gestuelle naturelle des sourds. Elle assure toutes les fonctions remplies par les langues orales.

LSFB : Langue des Signes de Belgique Francophone.

LSFB-lab : Laboratoire de Langue des Signes de Belgique Francophone.

Occlusion : phénomène survenant lorsqu'une partie du corps passe devant une autre masquant alors cette dernière.

RGB : espace de couleurs le plus répandu dans le monde numérique. Il permet de représenter une couleur à l'aide des composantes rouge (R), verte (G) et bleu (B).

rgb : espace de couleurs RGB normalisé.

RGBA : espace de couleurs RGB auquel une composante de transparence (A) a été ajoutée.

Signature d'un signe : manière de réaliser un signe. Certains signes peuvent avoir plusieurs signatures.

Signe : geste équivalent d'un mot en langue des signes. La relation entre mot et signe n'est cependant pas une relation *un-à-un*. En effet, il peut arriver que plusieurs signes représentent un même mot.

Signer : action de réaliser un signe.

Signeur : personne qui signe.

Tag : représentation textuelle d'un signe utilisée dans les fichiers d'annotations et définie par le LSFB-lab.

YCbCr : espace de couleurs représentant celles-ci à l'aide d'une composante de luminance (Y), d'une différence de bleu (Cb) et d'une différence de rouge (Cr).

Première partie

Introduction

Introduction

La langue des signes de Belgique francophone (LSFB) est la langue naturelle des sourds francophones mais elle n'est officiellement reconnue comme une langue que depuis octobre 2003. De par son statut de langue minoritaire et le manque de forme écrite, cette langue n'a que très peu de ressources à sa disposition, ce qui constitue un vrai handicap pour la recherche. C'est avec l'évolution des technologies au début des années 2000 que la recherche peut enfin s'épanouir. En effet, c'est l'occasion de développer des corpus linguistiques sur la langue des signes. L'informatique permet de récolter des données sous forme de vidéos, de les archiver, de les annoter et de les documenter. Cependant, l'annotation de ces données reste une tâche longue et manuelle mais indispensable pour pouvoir développer des corpus et pour élargir les données disponibles dans le but d'automatiser un jour cette annotation.

À Namur, le laboratoire de Langue des signes de Belgique francophone (LSFB-lab) a été inauguré en 2013. L'un des projets menés en son sein est la création du Corpus LSFB, premier large corpus informatisé illustrant l'usage actuel de la LSFB et disponible en accès libre. Pour concevoir ce corpus, de nombreux sourds et malentendants ont été filmés alors qu'ils discutaient entre eux. Pas moins de 150 heures de vidéos ont ainsi été produites. Parmi celles-ci, 12h ont déjà été annotées et 2h ont été traduites en français. Le travail d'annotation consiste à segmenter la vidéo pour définir l'instant de début et de fin de chaque signe et à attribuer à l'intervalle défini un tag identifiant ce geste. Actuellement, les annotations de ces vidéos sont réalisées manuellement par des annotateurs bilingues (LSFB-français). L'un des problèmes rencontrés par le laboratoire est la lenteur de ce processus d'annotation. En effet, environ huit heures de travail sont nécessaires pour annoter deux minutes de conversation vidéo. Pour pallier ce problème, une solution envisageable est d'automatiser le processus d'annotation.

Cependant, une telle solution n'est pas évidente à mettre en place. C'est pourquoi le but de ce mémoire est d'évaluer la faisabilité d'une solution visant à accélérer l'annotation des vidéos du LSFB-lab grâce à l'automatisation de celle-ci.

Pour évaluer cette faisabilité, nous avons intégré le LSFB-lab durant quatre mois dans le cadre de notre stage. Dans un premier temps, nous y avons observé les annotateurs et les vidéos à traiter afin de définir clairement les objectifs à atteindre. Ensuite, une solution de traitement semi-automatique des vidéos y a été développée. L'objectif de cette solution était de reconnaitre les signes réalisés et de produire les fichiers d'annotations correspondants. La mise en place de cette solution a débuté par une phase de documentation sur les techniques

de reconnaissance gestuelle. Certaines de ces techniques ont alors été implémentées et légèrement modifiées afin de tenir compte des contraintes particulières des vidéos à traiter. Nous nous sommes également renseignés sur la langue des signes afin de définir les critères de comparaison entre deux signes. Pour terminer, nous avons évalué notre solution en comparant les résultats de l'annotation automatique avec ceux de l'annotation manuelle. Grâce à cette évaluation, nous avons apporté une réponse à la question de la faisabilité d'une automatisation du processus d'annotation des vidéos du laboratoire.

La suite de ce mémoire est divisée en trois parties : le contexte et l'état de l'art, la contribution que nous avons apportée et une brève conclusion.

La première partie présente le contexte et l'état de l'art. Dans un premier temps, la langue des signes, son histoire et ses particularités sont expliqués. Ensuite, un état de l'art concernant l'informatique pour la langue des signes est établi, il présente l'état de la recherche en langue des signes ainsi que les techniques de reconnaissance gestuelle. Enfin, un background technologique nécessaire à la bonne compréhension du mémoire est abordé.

Comme son nom l'indique, la partie contribution présente la contribution de ce mémoire. L'analyse des besoins sera présentée et suivie par l'explication du fonctionnement de la solution proposée. L'implémentation de cette solution sera ensuite expliquée en présentant notamment les choix techniques qui ont été faits. Cette partie se terminera avec l'évaluation de cette solution.

Finalement, la partie conclusion propose un bref résumé des parties précédentes. Un point de vue sur les perspectives futures pour le laboratoire au vu des résultats obtenus est également présenté.

Deuxième partie

Contexte et état de l'art

Chapitre 1

Langue des signes

Dans ce chapitre, nous aborderons l'histoire de la langue des signes ainsi que ces caractéristiques et particularités. Les différentes informations présentées par la suite ont été extraites des ouvrages : [8], [9], [18], [12].

1.1 Histoire

Il existe quelques traces écrites concernant des sourds et des langages gestuels dans l'Antiquité, notamment dans la Bible ou dans la philosophie. À cette époque, les sociétés dominantes, comme la Grèce et Rome, ne considèrent pas les sourds comme des individus bien que les avis à ce sujet divergent. Platon observe que "si nous n'avions point de voix ni de langue et que nous voulussions nous montrer les choses les uns aux autres, n'essaierions-nous pas, comme le font en effet les muets, de les indiquer avec les mains, la tête et le reste du corps ?"[25]. Aristote n'est pas du même avis. Pour lui, l'ouïe est nécessaire pour s'instruire et être pleinement humain. "Les animaux, qui, tout en étant intelligents, ne peuvent rien apprendre, sont en général ceux à qui la nature a refusé un organe pour percevoir les sons, comme l'abeille et les autres espèces, s'il y en a qui soient à cet égard dénuées comme elle. Au contraire, ceux des animaux qui, à la mémoire, peuvent ajouter le sens de l'ouïe sont en état de s'instruire"[3]. Porphyre de Tyr, philosophe du IIIème siècle PCN, estime que la raison n'a aucun lien avec l'ouïe ou la parole. Il écrit : "N'est-il pas absurde de déterminer qu'un être est doué de raison ou ne l'est pas selon que son parler est intelligible ou non, qu'il reste muet ou qu'il a un langage ? On refuserait ainsi la raison au Dieu qui est au-dessus de tout et aux autres Dieux parce qu'ils sont muets ?"[26].

Au Moyen Age, la situation des sourds n'est pas aussi mauvaise. Les sourds exercent divers métiers : bouchers, laboureurs, servantes, portiers ou encore moines. Cette période connaît quelques tentatives d'instruction isolées mais les sourds restent des marginaux.

La Renaissance voit apparaître quelques tentatives concrètes pour instruire les sourds grâce à la langue des signes. Certaines célébrités comme Léonard de Vinci, Montaigne, Diderot, Descartes, Rousseau ou Condillac estimaient que la parole gestuelle était autant capable et digne que les langues vocales. À Amiens vers 1720, Etienne de Fays, un sourd érudit, éduque un petit groupe d'enfants en utilisant les signes.

La première école pour sourds est ouverte à Paris en 1760 par l'abbé Michel de l'Épée. Il utilise alors un mélange de signes couramment utilisés par les sourds (l'alphabet manuel) et de gestes de son invention (les signes méthodiques). Toute l'Europe entend parler de son enseignement et les gens se pressent à ses leçons pour admirer l'intelligence des jeunes sourds. Grâce à des démonstrations publiques, et à l'intérêt que lui porte à l'époque l'opinion publique, il réussit à obtenir des subsides.

Durant le début du 19e siècle, environ 200 écoles pour sourds sont créées à travers l'Europe par d'anciens élèves de l'Abbé de l'Epée devenus professeurs. La première école belge pour sourds est fondée à Liège par Jean-Baptiste Pouplin, école où enseigne le premier professeur belge, Joseph Henrion, lui aussi ancien élève de l'institution de l'Abbé de l'Épée. Ensuite, d'autres écoles ouvrent leurs portes dans tout le pays.

En 1816, Thomas Gallaudet, pasteur américain, et Laurent Clerc, professeur sourd et ancien de l'institut de l'Abbé de l'Epée, se rendent aux Etats-Unis pour y fonder la première école pour sourds. En 1864, le fils de Thomas Gallaudet, Edward, fonde l'université Gallaudet destinée aux Sourds et malentendants. Cette université est, à ce jour, la seule université pour sourds au monde.

À Saint-Jacques, Auguste Bébian, ami de Laurent Clerc, est le premier professeur entendant à enseigner en langue des signes et non en français signé. Comme il a fait une analyse approfondie du langage des signes, il corrige ce qui lui paraît inexact ou ambigu. Ceci permet aux gestes de mieux répondre à tous les besoins de l'intelligence des sourds. Auguste Bébian est licencié en 1821 mais, neuf ans plus tard, en décembre 1830, une révolte éclate car les élèves veulent que Bébian revienne et soit nommé directeur. Finalement, le calme revient après dix jours, suite à l'exclusion de trois émeutiers. C'est la première affirmation collective d'une identité sourde.

En 1834, Ferdinand Berthier met en place la tradition des banquets sourds où sont invités des journalistes qui en font des comptes rendus admiratifs. En 1838 à Paris, Berthier se penche sur la question des droits des sourds et crée la Société centrale des sourds-muets, qui permet aux sourds de se rendre compte de leurs droits.

En 1844, le premier mariage entre sourds est célébré, ce qui, à l'époque, deviendra la norme. La langue des signes est alors transmise en même temps que la surdité.

Fin du XIXeme siècle, la langue des signes est de moins en moins bien vue. En 1880 a lieu le Congrès de Milan qui rassemble les professeurs de l'éducation des sourds, à majorité entendants, qui veulent l'interdiction de la langue des signes dans les écoles. On y décide d'interdire son utilisation dans le but de la remplacer par l'oralisation. Plusieurs choses peuvent expliquer ce choix : les préjugés religieux qui donnaient la parole supérieure, la confiance en la médecine qui, on l'imaginait, allait rendre l'audition aux sourds, ainsi que la méconnaissance de la langue des signes que les gens pensaient incapable d'abstraction.

La décision du Congrès de Milan fut lourde de conséquences. Elle est appliquée partout sauf aux Etats-Unis. Les professeurs qui enseignent en langue des signes sont licenciés et les nouveaux élèves sont séparés des anciens. Pendant les 100 prochaines années, les sourds sont alors vus comme des anormaux, infirmes ou limités. Ils sont cantonnés à des métiers manuels en bas de l'échelle sociale. Bien que la langue des signes soit encore pratiquée lors des rencontres dans les associations, elle n'est plus officiellement parlée. Cela a pour effet

d'appauvrir fortement la langue des signes qui se diversifie progressivement en dialectes propres aux communautés isolées les unes des autres.

En 1922, Emile Cornet fonde la Fédération Sportive Belge pour les sourds. Et en 1924 ont lieu les premiers Jeux Olympiques pour sourds à Paris.

Dans les années 70, beaucoup d'associations sont créées, notamment la Fédération Francophone des Sourds de Belgique (FFSB) en 1977. En 1981, Année Internationale des Handicapés, la traduction du journal télévisé de la RTBF en langue des signes apparait.

En 1982, un peu plus de cent ans après le décret de Milan a lieu le "Réveil Sourd", un décret autorisant de nouveau l'utilisation de la langue des signes dans l'enseignement a vu le jour. Un an plus tard est publié le premier lexique LSFB-français. Les cours de langue des signes de l'époque sont simplement des cours de français signé et les enseignants ne sont pas spécifiquement formés à la langue des signes. Par la suite, en 1988, le Parlement Européen dépose une résolution pour encourager les États membres à reconnaitre leur langue des signes.

Au début des années 90, les associations consultent les ministres concernés dans le but qu'un décret de reconnaissance de la langue des signes soit voté. Cependant, deux choses sont nécessaires pour qu'un tel décret voie le jour : l'existence de cours de LSFB et l'existence d'interprètes LSFB-français. En 1994, les premiers cours en promotion sociale sont créés à Liège et à Bruxelles et, en 1995, le premier service d'interprètes pour sourds est ouvert à Namur. Suite à cela, un premier projet de décret de reconnaissance de la LSFB est déposé au gouvernement de la Communauté Française qui demande alors une étude de faisabilité. Celle-ci ne sera réalisée qu'en 2002 pour que les subsides nécessaires à la réalisation de cette étude soient dégagés. Le rapport est réalisé en quatre mois (délai imposé) par l'Université Libre de Bruxelles, le laboratoire de recherche PROFILS (Programme de recherche, d'orientation et de formation d'interprètes en langue des signes) et l'Institut Libre Marie Haps de Bruxelles. Ce rapport dégage plusieurs priorités : encourager les initiatives en matière d'enseignement bilingue, soutenir la professionnalisation du métier d'interprète, favoriser l'émergence de cadres sourds au sein de la communauté et de professionnels sourds dans les instances liées à la surdité et promouvoir la recherche en LSFB. Nicole Maréchal, ministre de l'Aide à la jeunesse et de la Santé de 1999 à 2004, déclare [1] :

> "Ce rapport a aussi permis de définir quelle langue des signes reconnaitre. Ce qui était indispensable vu la variété de la langue des signes, des variantes régionales, des tournures,... La langue des signes était reconnue comme une langue vivante, avec de grandes potentialités créatives, disposant d'une grammaire, d'une syntaxe, d'une conjugaison, faisant naître des signes nouveaux, en adaptant d'autres... Ils ont ensuite fait un état des lieux, des usages de la langue des signes dans différents domaines : éducation, enseignement, formation, accès aux services, aux emplois, aux soins de santé, à l'information,... Ils ont émis des recommandations, mais mieux encore, ils les ont hiérarchisées en leur donnant des priorités différentes. Cela permettait très pratiquement aux politiques d'élaborer un calendrier. Enfin, ils ont aussi évalué les coûts de cette reconnaissance, coût qui pouvait lui aussi s'étaler dans le temps selon les priorités définies."

La langue des signes de Belgique francophone (LSFB) est finalement reconnue en octobre 2003 par le parlement de la Communauté Française par un vote à l'unanimité. Nicole Maréchal raconte que "ce décret est un texte cadre, une étape ouvrant sa porte à des actions à réaliser pour permettre l'épanouissement personnel et social des personnes sourdes. La reconnaissance de la langue des signes était un acte symbolique extrêmement important". En même temps que ce décret naît la Commission Consultative de la langue des signes.

Aujourd'hui, les Sourds avec un grand "S" se considèrent comme une vraie communauté avec sa langue, ses règles d'échange et ses valeurs propres. Pour les Sourds, la différence entre entendants et sourds ne se définit pas en terme de déficit mais en mode de communication : les uns communiquent avec la voix et les autres avec les gestes. Un sourd sera handicapé chez les entendants tout autant qu'un entendant le sera parmi les sourds.

Les Sourds sont en général des personnes nées sourdes ou très malentendantes ou qui le sont devenues très tôt. Pour ces personnes, le français oral n'est pas un moyen de communication naturel. Les autres préfèrent se qualifier de malentendants et rejette le qualificatif de "sourds". Ceci les démarque des Sourds qui, eux, sont fiers de l'être. Ces Sourds ne savent pas faire le lien entre la parole et l'écrit, qui est une simple retranscription de la parole, car la langue orale et la langue des signes fonctionnent sur des modes très différents. Ainsi, un Sourd qui lit doit faire un double effort : il doit d'abord décoder l'écrit sur lequel il met du sens et ensuite traduire ce sens dans sa propre langue. C'est pour cette raison qu'il existe un illettrisme particulier chez les Sourds. Ceux-ci sont allés à l'école et connaissent l'alphabet mais certains sont incapables de donner du sens à leur lecture ni de transmettre du sens par l'écrit.

1.2 La langue des signes

La Langue des Signes de Belgique Francophone (LSFB) est la langue naturelle des sourds francophones. C'est une langue visuelle et non verbale avec sa propre syntaxe. La LSFB, comme toutes les langues des signes, permet d'exprimer toutes les nuances possibles et il existe des styles différents de communication comme la poésie, l'humour, l'argot, les injures ou encore les jeux de mots (ou plutôt jeux de signes).

Comme toutes les langues, la langue des signes possède un vocabulaire et une grammaire. Cependant, il existe beaucoup de différences entre la langue des signes et la langue orale. Tout d'abord, la langue des signes n'utilise pas le canal audio-vocal mais le canal visuo-gestuel. De plus, la langue orale ne permet pas d'exprimer deux choses en même temps. La langue des signes, quant à elle, utilise les mains, le visage et le mouvement des épaules. Il est donc possible de désigner plusieurs choses et plusieurs actions en même temps. La langue des signes possède une grammaire en partie spatialisée : l'espace de signation situé devant le signeur est comme une scène de théâtre. Les décors, les personnages et les objets y sont placés. Ils sont ensuite désignés et déplacés en fonction des besoins.

La langue des signes n'est pas internationale. Les langues des signes de chaque pays diffèrent par leur vocabulaire. En effet, chaque signe n'est pas créé arbitrairement mais correspond aux besoins des signeurs. Les signes évoluent avec le temps et en fonction des

communautés qui l'utilisent. Cependant, deux sourds étrangers sont, en général, capables de se comprendre. En effet, la grammaire des langues des signes est fort semblable dans tous les pays. De plus, la langue des signes est très imagée. Chaque signe a une certaine iconicité, ce qui le rend plus ou moins compréhensible même pour quelqu'un qui ne le connaît pas. On utilise par exemple les transferts de taille et de forme pour présenter les contours ou un détail significatif de l'objet que l'on veut désigner. Il existe d'ailleurs une langue des signes internationale (LSI) qui utilise énormément l'iconicité.

1.2.1 L'iconicité et la pensée visuelle

Les langues des signes nécessitent de faire une abstraction totale de la langue orale. Il ne faut pas penser ce que l'on va dire en mots mais en images. C'est ce que l'on appelle la pensée visuelle.

On ne peut pas traduire mot à mot du français en signes. Ce serait alors du français signé et non de la langue des signes. Le français signé est compliqué à comprendre pour un sourd qui ne maitrise pas le français oral. De la même façon que la lecture, le français signé demande un double travail pour le sourd : il devra d'abord décoder les mots puis donner du sens. Prenons des expressions telles que "elle est tombée enceinte" ou "tu veux un coup de main", traduites mot à mot, le sens de ces expressions laisse à désirer.

La pensée visuelle permet de se représenter les choses en "images" plutôt qu'en "mots", ce qui est très intéressant car cela permet de se représenter des phénomènes qui sont très compliqués à exprimer par des mots alors qu'un schéma suffit. C'est souvent le cas dans les sciences. De ce fait, la langue des signes permet parfois d'exprimer bien plus simplement que la langue orale des concepts scientifiques.

L'iconicité joue un très grand rôle dans la langue des signes. Par exemple, pour exprimer le temps, le corps représentera le présent. Le passé sera vers l'arrière et le futur vers l'avant. Un autre exemple est l'utilisation d'acteurs. Comme expliqué précédemment, l'espace devant le signeur est une scène. Le signeur présentera ses acteurs en les décrivant une fois (ou en le pointant du doigt directement) et les placera dans l'espace. Il les réutilisera ensuite au moment voulu et ne devra plus en refaire la description.

L'iconicité peut intervenir à plusieurs niveaux. Ce peut être le fait de choisir un signe plutôt qu'un autre parce qu'il est plus imagé. L'iconicité intervient également dans l'agencement des phrases : la disposition des signes dans l'espace de signation permet de reconstruire des représentations scéniques. L'iconicité peut aussi traduire une logique de la pensée en représentant un terme par un détail visuel plutôt que par le mot standard.

1.2.2 Paramètres de la langue des signes

Un signe en langue des signes peut être décomposé en éléments de base qui peuvent être combinés à l'infini. Ces éléments, aussi appelés paramètres du langage, sont au nombre

de cinq [1].

— La configuration de la main ;
— L'orientation de la paume ;
— Le mouvement ;
— L'emplacement où le signe est articulé ;
— L'expression du visage.

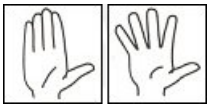
La configuration

La configuration de la main est en fait la forme de celle-ci. Elle est souvent désignée par la lettre de l'alphabet dactylographique à laquelle elle correspond ou par le chiffre qui la représente. Elle peut être étudiée selon les traits suivants :

— La position du pouce : croisé, contre, étendu, avant (avec ou sans contact) :

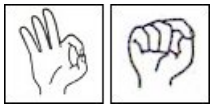
— Pour les autres doigts : leur nombre (1 à 4) et lesquels (index, majeur, annulaire, auriculaire) :

— L'écartement des doigts : serrés ou écartés :



— La position : droit, angle, courbe, plié ou compact :

— Le contraste d'un doigt avec les autres :



1. Le fait qu'il y ait cinq paramètres est parfois discuté. En effet, certains auteurs (comme Stokoe) considèrent que l'orientation peut être prise en considération dans la configuration de la main et que les expressions du visage sont à part.

Statistiquement, il y a une dizaine de configurations très fréquentes (appelées "non marquées") et environ vingt-cinq moins fréquentes qui forment la plupart des signes. Les configurations non marquées sont les plus faciles à reconnaitre et reproduire.

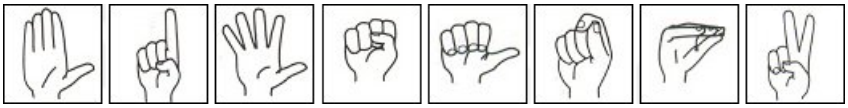


FIGURE 1.1 – Exemple de configurations non marquées

On peut également observer que certains signes se font à une main et d'autre à deux mains. On peut en réalité distinguer trois cas de figure différents :

— Une seule main active ;
— Deux mains actives ;
— Un main active et une passive.

L'orientation

L'orientation de la main est parfois associée directement à la configuration ou au mouvement. Nous la considérerons comme un paramètre à part entière car elle permet dans de nombreux cas de différencier deux signes. On dénombre six orientations différentes (vers le haut, vers le bas, vers la gauche, vers la droite, vers soi et vers l'avant) bien que certains se posent la question de savoir s'il ne faudrait pas en compter plus. Une main possède une double orientation : celle de la paume et celle des doigts.

Le mouvement

Le mouvement se compose de plusieurs caractéristiques :

— Le nombre de mains actives. Dans le cas où les deux mains sont actives, le mouvement de la seconde peut être symétrique, parallèle, alterné ou conjoint au mouvement de la première main.
— Le type de mouvement : rectiligne, courbe, circulaire, rotatif, pianotage des doigts ou une composition ou répétition de ces éléments.
— La direction dont le rôle est généralement syntaxique. Par exemple, pour dire "donne-moi", le signeur fera le signe "donner" de l'interlocuteur vers lui-même. Inversement, pour dire "te donner", le signeur fera le signe "donner" de lui-même vers l'interlocuteur.
— La vitesse et l'intensité.
— Le changement de configuration comme une ouverture ou une fermeture de la main.
— Le contact avec l'autre main ou le corps. Ce contact peut survenir lors de la phase initiale ou finale et peut être prolongé ou bref.

L'emplacement

Il y a entre douze et trente emplacements possibles selon les chercheurs. Tous ces emplacements se regroupent en catégories :

— sur le corps :
  — devant le signeur : au niveau du menton, du front, de la joue, de l'estomac, de l'épaule, etc.
  — sur le bras, le poignet, les doigts ou la main de base (main passive)
— dans l'espace :
  — au niveau du torse ;
  — au niveau du visage ;
  — au niveau du cou.

Comme pour la plupart des paramètres, seules certaines possibilités d'emplacement sont retenues car considérées comme pertinentes. En effet, les interlocuteurs se regardent dans les yeux lors d'une conversation. Les signes seront donc mieux distingués s'ils se font au niveau du visage ou du cou. Pour cette raison, les signes ayant une configuration plus compliquée seront faits plus près du visage alors que les signes plus éloignés auront une tendance à être assez simples.

L'expression du visage

Les expressions du visage peuvent apporter beaucoup de nuances à un signe. Par exemple, pour marquer le fait que quelque chose est très gros, les joues seront gonflées alors que pour montrer que quelque chose en minuscule, la bouche sera pincée et les yeux plissés.

Les éléments à prendre en compte pour analyser l'expression du visage sont :
— les yeux : la direction du regard et le degré d'ouverture ;
— les sourcils : levés ou froncés ;
— les joues et la bouche : joues gonflées, bouche pincée, etc. ;
— les mouvements de la tête, des épaules et du tronc ;
— l'expression générale du visage : sévère, joyeux, triste, étonné, etc.

Chapitre 2

Etat de l'art : l'informatique pour la langue des signes

2.1 Etat de la recherche en langue des signes

La langue des signes est une des langues qui possède le moins de ressources. Cela est du à plusieurs choses. Tout d'abord, la langue des signes est une langue minoritaire qui n'est reconnue que depuis quelques années. De plus, il n'existe pas officiellement de forme écrite pour cette langue bien que certains aient essayé en inventant le signwriting [1]. Mais cette écriture reste très compliquée.

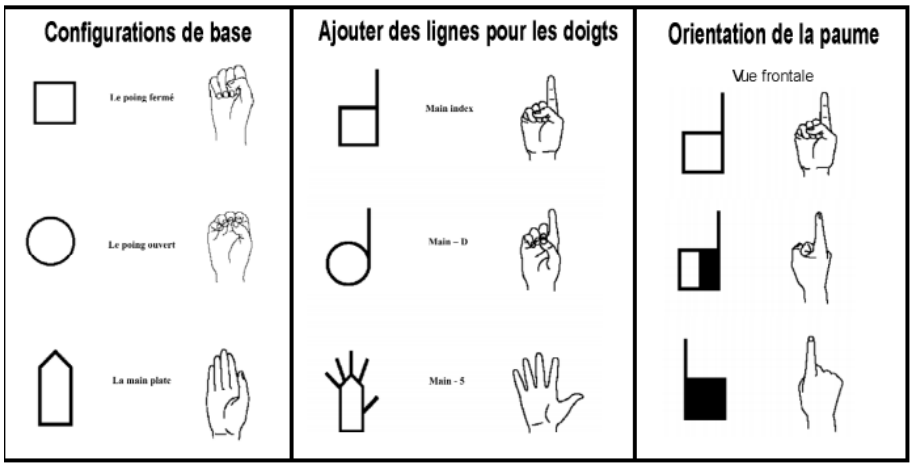


FIGURE 2.1 – Principes de base du signwriting. Image reproduite de : `http://www.signwriting.org/`.

1. http ://www.signwriting.org/

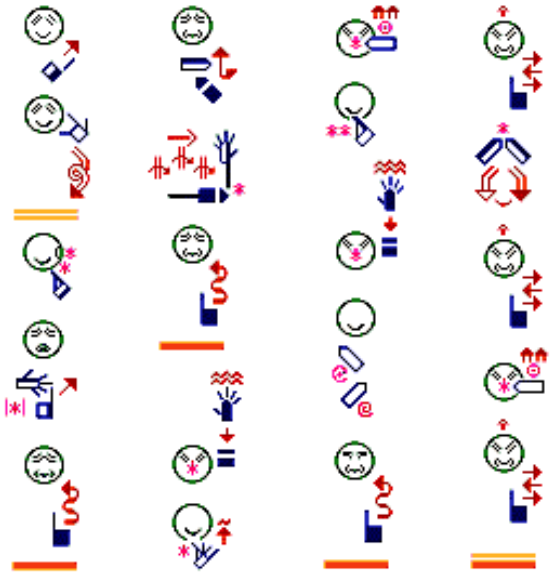


FIGURE 2.2 – Exemle d'écriture signwriting. Image reproduite de : `http://www.omniglot.com/writing/signwriting.htm`.

2.1.1 Utilité de l'informatique

La recherche en langue des signes a commencé avec les travaux de Tervoort en 1953 et Stokoe en 1960. Mais c'est avec la création en 2004 de la Sign Language Linguistics Society (SLLS) que la recherche sur la langue des signes devient un effort mondial. L'évolution des technologies au début des années 2000 a eu un impact important pour la recherche : c'était la possibilité de développer des corpus linguistiques informatisés sur la langue des signes. En effet, un dictionnaire papier est très limité pour représenter des signes, des mouvements. L'informatique permet de récolter des données sous forme de vidéo et de les archiver, les annoter et les documenter. Cependant, l'annotation de ces données reste une tâche longue et manuelle qui est inévitable pour élargir les données disponibles dans le but d'automatiser un jour ce processus d'annotation.

Sinclair définit en 1996 un corpus comme étant une collection de morceaux de langage qui sont sélectionnés et triés selon des critères linguistiques explicites afin d'être utilisés comme un échantillon du langage. Un corpus informatique est un corpus encodé de manière homogène et standardisée pour des tâches ouvertes. Les corpus ont une très grande importance pour la recherche en langue des signes. En effet, les langues sans forme écrite (en particulier les langues des signes) n'ont même pas l'entrée textuelle la plus élémentaire pour pouvoir analyser ce langage. À ce niveau, les corpus non informatisés et les techniques statistiques ne sont pas disponibles. L'établissement de larges corpus électroniques à partir desquels il est possible de dériver des informations pourraient considérablement améliorer la productivité des chercheurs en langage gestuel.

Les corpus ont un rôle important dans la recherche. Cependant, la création de corpus informatiques étant une "discipline" assez jeune, elle souffre d'un manque de conventions et de standards. En effet, il existe énormément d'outils d'annotation : AnCoLin, Anvil, ELAN, iLex, SignStream et syncWRITER. Chaque outil possède sa propre norme. Cette situation empêche la possibilité d'échanger des données entre projets ce qui implique qu'un travail similaire doit être répété pour de nouveaux projets, ralentissant les progrès.

La technologie a également permis d'inventer quelques systèmes pour aider les sourds à communiquer. Outre Kinect et les méthodes de reconnaissance gestuelle présentées à la section suivante, des outils pour traduire la langue des signes en temps réel ont été créés. C'est le cas notamment des gants Enable Talk [2], du Nexus Project [3] ou encore d'un bracelet et de bagues pour traduire la langue des signes [4].

2.1.2 Le laboratoire LSFB

En Belgique, et plus précisément dans la fédération Wallonie-Bruxelles, les premières recherches sur la langue des signes de Belgique francophone (LSFB) ont commencé à l'Université de Namur (UNamur) en 2000. Ces recherches ont mené à une première thèse de doctorat en 2006 par Laurence Meurant. En février 2013, le laboratoire de Langue des signes de Belgique francophone (LSFB-lab) est inauguré. C'est un laboratoire de l'UNamur consacré à la recherche sur la langue des signes de Belgique francophone. L'équipe du laboratoire, qui travaille sous la supervision de Laurence Meurant, comprend des sourds et des entendants et se compose d'une quinzaine de personnes : des chercheurs, un responsable vidéo, des enseignants et des experts de langue des signes.

le projet Corpus LSFB

Un des projets du laboratoire est le Corpus LSFB. L'objectif de ce corpus est de constituer un large ensemble de données qui serait disponible pour la recherche linguistique sur la langue des signes de Belgique francophone. Ce projet a démarré en 2012 grâce à un Mandat d'Impulsion Scientifique octroyé par le F.N.R.S. Bien que ce corpus ait été conçu pour la recherche linguistique dans un premier temps, c'est également un outil au service des enseignants, des formateurs, des étudiants et des interprètes. Le corpus permet aussi une sauvegarde de l'héritage linguistique et culturel de la communauté des Sourds. Le corpus est également libre pour tous les utilisateurs et leur permet de trouver des vidéos pour le plaisir ou pour enrichir ou revoir un cours.

Ce projet a eu l'avantage de ne pas être le premier à vouloir rassembler des données sur la langue des signes. D'autres équipes, dans d'autres pays (notamment en Australie, aux Pays-Bas, en Grande-Bretagne et en Allemagne), ont déjà tenté l'expérience. Le projet du Corpus-LSFB a donc pu compter sur l'expérience de ces équipes et surtout du Dr. Onno

2. http ://www.enabletalk.com/
3. http ://www.nexusproject.reaco.fr
4. http ://www.educadis.fr/cours-de-langue/news-langues-elearning/un-bracelet-et-des-bagues-pour-traduire-la-langue-des-signes

Crasborn, responsable du Corpus néerlandais à la Radboud Universiteit Nijmegen, qui a joué le rôle de conseiller.

Le Corpus LSFB est le premier large corpus informatisé illustrant l'usage actuel de la LSFB et disponible en accès libre. Cent signeurs de profils variés âgés de 18 à 95 ans ont été filmés entre 2013 et 2015 en train d'utiliser la LSFB pour un total de 150h de vidéo. Une partie de ces vidéos a été annotée (10h/150h) et traduite en français (2h/150h). L'annotation et la traduction des vidéos est réalisée avec l'outil ELAN. Pour chaque signe réalisé dans la vidéo, les annotateurs doivent spécifier le temps de début et de fin de celui-ci et associer un "tag" à l'intervalle de temps défini. En outre, l'annotation ainsi produite doit être associée à l'une des deux mains afin de disposer d'annotations distinctes pour les mains gauche et droite (pouvant réaliser des signes différents). Les traducteurs, quant à eux, doivent traduire les phrases signées en français. De la même manière que pour les annotations, ils vont devoir définir le temps de début et de fin de chaque phrase et ensuite associer la traduction française à cet intervalle de temps.

Le corpus contient trois profils de signeur différents :
- Les signeurs natifs : la LSFB était la langue parlée par leurs parents ;
- Les signeurs quasi-natifs : ils ont acquis la LSFB très jeunes et ont été à l'école avec d'autres Sourds ;
- Les signeurs tardifs : ils ont acquis la LSFB tardivement.

Le corpus a été organisé en cinquante sessions et dix-neuf tâches. Une session correspond à une journée d'enregistrement et contient les vidéos de deux signeurs qui discutent en fonction des tâches proposées par le modérateur. Ils ont du se présenter et se raconter des histoires, des souvenirs ou des blagues. Ils ont du décrire des plans, des itinéraires et des portraits. Ils ont également du argumenter, comparer et classer des choses. Finalement, ils ont discuté de la langue des signes et de la communauté sourde.

Utilité du corpus

Le corpus peut être utilisé dans plusieurs domaines différents : pour la recherche, pour l'enseignement ou par simple curiosité. Il peut aider à répondre à diverses questions comme "que pensent les sourds de la relation entre sourds et entendants ?", "quelle est la différence entre les signeurs de Liège ou du Hainaut ?", "quel genre de blague se racontent-ils ?", "quelle vidéo pourrait être utile pour le cours que je vais donner aujourd'hui ?", "de quelles manières différentes peut être exprimé ce mot ?", etc.

Le corpus est accessible via le site web `http://www.corpus-lsfb.be/`. Bien que le site permette de faire une "simple visite", cette dernière ne permet pas d'avoir accès au corpus mais uniquement au lexique. Pour pouvoir consulter le corpus, il faut demander un accès. Trois types d'accès peuvent être demandés : public, professionnel ou chercheur. L'accès public permet d'accéder aux vidéos du corpus et, contrairement aux deux autres types d'accès, ne demande pas d'être validé par l'équipe du LSFB-lab. L'accès professionnel est prévu pour les enseignants, interprètes ou formateurs et permet d'avoir accès à plus de vidéos. L'accès chercheur est prévu pour les personnes faisant de la recherche comme, entre

autres, les linguistes, sociologues ou anthropologues. Cet accès permet d'obtenir d'autres vidéos ainsi que des métadonnées potentiellement utiles pour la recherche.

Le site contient un moteur de recherche qui permet de naviguer de plusieurs manières dans les vidéos : en choisissant un type de signeur (âge, région, profil linguistique), un signeur en particulier, une tâche d'une session particulière, etc. Ces critères peuvent bien sûr être combinés.

2.2 Reconnaissance gestuelle

La reconnaissance gestuelle est la capacité d'un système à reconnaitre les gestes effectués par l'utilisateur et de réaliser des actions en conséquence. La première chose avant d'aborder les techniques de reconnaissance est de bien définir ce qu'est un geste.

Le dictionnaire Larousse définit le geste comme un "mouvement du corps porteur ou non de signification". Adam Kendon [17] propose une définition plus formelle :

> "*Un geste est une forme non-verbale de communication dans laquelle des actions visibles du corps communiquent des messages particuliers, soit à la place de la parole ou avec et en parallèle des mots. Les gestes incluent les mouvements des mains, du visage ou de toutes autres parties du corps. Les gestes diffèrent des autres formes de communications physiques non-verbales qui ne communiquent pas de messages spécifiques comme les attitudes purement expressives, les proxémiques ou les manifestations d'attention commune.*"

Les différents gestes existants peuvent être classifiés selon certaines caractéristiques majeures et de nombreuses taxonomies ont été proposées dans la littérature. Nous retiendrons toutefois celle proposée par Mohamed Kaâniche [16] en raison de sa complétude. Selon cette taxonomie, les gestes peuvent d'abord être classés en fonction des parties du corps impliquées dans leur réalisation et de leur aspect dynamique ou statique. Les gestes dynamiques peuvent en outre être décomposés en gestes inconscients et conscients. Enfin, les gestes peuvent être décomposés, selon leur objectif, en cinq types :

- Les emblèmes traduisant une courte communication verbale ;
- Les illustrateurs décrivant ce que la personne dit verbalement. Ce type de gestes peut encore être décomposé en cinq sous-catégories (battement, déictique, iconique, métaphorique et cohésif) ;
- Les gestes émotionnels véhiculant l'émotion de la personne ;
- Les régulateurs permettant de contrôler l'interaction ;
- Les adaptateurs servant à relâcher la pression du corps. Il s'agit du seul des cinq sous types à être réalisé inconsciemment.

La figure 2.3 permet de visualiser cette taxonomie à l'exception toutefois du critère concernant les parties du corps.

Dans le cadre de la reconnaissance automatique de la langue des signes, il est nécessaire que le système soit capable de reconnaitre les gestes de la main dynamiques et statiques.

Le processus général de reconnaissance gestuelle à l'aide des technologies orientées vision se compose de trois grandes phases : la détection des parties du corps d'intérêt, le suivi de celles-ci durant la séquence vidéo et la reconnaissance des gestes réalisés. En outre, afin de réaliser la phase de reconnaissance, il est nécessaire d'extraire certaines caractéristiques telles que la forme ou la position durant les phases de détection et de suivi. Dans la suite de cette section, nous aborderons les méthodes existantes afin de mener à bien chacune des trois étapes du processus.

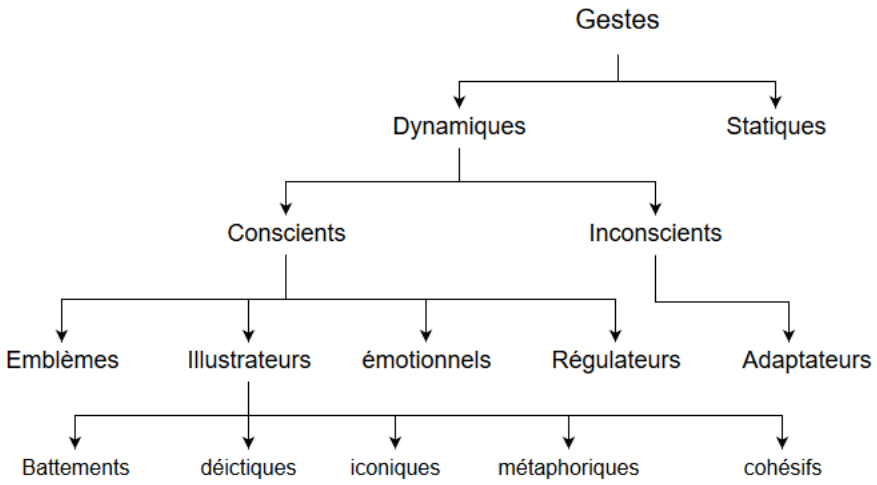


FIGURE 2.3 – Taxonomie des gestes proposée par Mohamed Kaâniche [16].

2.2.1 Détection

La détection représente la première étape du processus de reconnaissance gestuelle. Cette étape consiste à repérer et à isoler les zones de l'image comprenant les parties du corps d'intérêt. Un certain nombre de méthodes ont été mises au point en se basant sur les caractéristiques de couleurs, de formes, de mouvements et sur les modèles anatomiques des mains. Bien que ces caractéristiques puissent être utilisées seules pour détecter les mains, il n'est pas rare que les systèmes de détection les combinent [28].

Détection par couleur

Le principe de la détection par couleur est de classifier les pixels comme appartenant à une région de peau ou non en fonction des valeurs de leurs composantes de couleur.

D'abord, il est nécessaire de choisir une représentation de la couleur (un espace de couleur) adéquate parmi les nombreuses utilisées dans le traitement d'images (voir section 3.2).

Ensuite, la distribution des couleurs de peau au sein de l'espace de couleur choisi doit être modélisée. Grâce à cette distribution, il est finalement nécessaire de classifier les pixels des images observées. Trois grandes catégories de méthodes existent pour réaliser la modélisation et la classification : les régions explicitement définies, les méthodes non paramétriques et les méthodes paramétriques.

Les régions explicitement définies constituent la méthode la plus simple. Le principe est de définir un intervalle pour l'ensemble ou une partie des composantes de couleur. La combinaison des intervalles délimite alors une région dans l'espace contenant les différentes couleurs de peau possibles. Étant donné un pixel, celui-ci sera classifié comme pixel de peau si sa valeur de couleur appartient à la région définie. La difficulté de ces méthodes réside dans le choix des intervalles. En effet, des intervalles mal définis induiront une région couvrant trop ou trop peu de couleurs impliquant une détection de mauvaise qualité. Différents ensembles d'intervalles applicables aux espaces de couleur les plus répandus ont été définis au cours de précédents travaux [30].

Les méthodes non paramétriques consistent à attribuer à chaque point de l'espace de couleur une valeur déterminant son appartenance à la classe peau. Ces valeurs peuvent être attribuées à l'aide d'**histogrammes** ou d'un **réseau de neurones artificiels de type SOM** (Self-Organizing Map) et d'un ensemble d'images d'entraînement dont les pixels ont été manuellement classifiés.

Les méthode basées sur les **histogrammes** nécessitent dans un premier temps de construire un histogramme de peau et un histogramme de "non peau". Chaque composante de couleur considérée est donc divisée en zones (ou bins) représentant un intervalle de valeurs. Ensuite, pour chaque n-uplet possible d'intervalles, le nombre de pixels des images d'entraînement représentant une couleur de peau et dont les valeurs de couleur appartiennent aux intervalles définissant le n-uplet est calculé et stocké dans l'histogramme de peau. Le même procédé est appliqué aux pixels représentant les couleurs de "non-peau" pour construire l'histogramme correspondant. Finalement, les valeurs ainsi déterminées sont normalisées afin d'obtenir, pour chaque couleur c, les probabilités $P_{peau}(c)$ et $P_{\neg peau}(c)$ représentant respectivement les chances d'observer la couleur c sachant qu'il s'agit d'une couleur de peau ou de "non-peau" soit, $P(c|peau)$ et $P(c|\neg peau)$. Plus formellement, $P(c|peau)$ et $P(c|\neg peau)$ se calculent comme suit :

$$P(c|peau) = \frac{HistoPeau(c)}{NormalisationPeau}, \tag{2.1}$$

$$P(c|\neg peau) = \frac{HistoNonPeau(c)}{NormalisationNonPeau} \tag{2.2}$$

Les coefficients de normalisation $NormalisationPeau$ et $NormalisationNonPeau$ peuvent, par exemple, correspondre respectivement au nombre total de pixels des histogrammes de peau et de "non-peau" [14].

Une fois les histogrammes construits, la classification d'un pixel p de couleur c peut s'effectuer en calculant la probabilité d'observer une couleur de peau en sachant que la couleur effectivement observée est c ($P(peau|c)$) grâce au théorème de Bayes :

$$P(peau|c) = \frac{P(c|peau)P(peau)}{P(c|peau) * P(peau) + P(c|\neg peau) * P(\neg peau)} \tag{2.3}$$

où $P(peau)$ et $P(\neg peau)$ représentent respectivement les probabilités à priori d'appartenance aux classes peau et non peau. Cette probabilité peut ensuite être comparée à un seuil Θ et

si

$$P(peau|c) > \Theta, \tag{2.4}$$

alors p est classifié comme pixel de peau [14].

Une autre façon de déterminer la classe d'un pixel p est de comparer les valeurs de $P(c|peau)$ et de $P(c|\neg peau)$. Si le ratio entre ces deux probabilités dépasse un seuil Θ, le pixel p appartient à la classe de peau [30].

$$\frac{P(c|peau)}{P(c|\neg peau)} > \Theta \tag{2.5}$$

Avec les réseaux **SOM**, la distribution des couleurs de peau au sein de l'espace est représentée grâce à un réseau de noeuds. Durant une phase d'entraînement, chacun de ces noeuds se voit attribuer un vecteur v_i de dimension égale au nombre de composantes de couleur considérées. Cette phase d'entraînement peut être réalisée de deux façons menant chacune à une méthode de classification particulière [6].

La première méthode consiste à entraîner le réseau en ne lui fournissant que des pixels représentant de la peau. La classification d'un pixel p de couleur c s'effectue alors en vérifiant que la distance euclidienne entre c et le vecteur v_k du noeud le plus proche de c ne dépasse pas un seuil d'acceptation défini.

La deuxième méthode d'entraînement nécessite de fournir au réseau à la fois des pixels de peau et des pixels de "non-peau". En plus du vecteur v_i, chaque noeud se voit attribuer un label représentant la classe avec laquelle il a eu le plus de correspondance durant la phase d'entraînement. Pour classifier un pixel p de couleur c, il suffit alors de regarder le label du noeud n_i le plus proche de c, c'est-à-dire le noeud pour lequel la distance entre c et v_i est minimale.

La dernière catégorie de méthodes, **les méthodes paramétriques**, représente la distribution des pixels de peau à l'aide de modèles mathématiques délimitant le cluster des couleurs de peau au sein de l'espace de couleur. Plusieurs modèles mathématiques ont été proposés pour représenter la distribution des couleurs de peau à savoir les modèles gaussiens uniques et multiples (SGM et GMM) ainsi que les limites elliptiques. Dans les modèles gaussiens uniques, la probabilité $P(c|skin)$ se détermine grâce à la fonction de probabilité

$$P(c|skin) = \frac{1}{2\pi|\sum_s|^{\frac{1}{2}}} * e^{-\frac{1}{2}(c-\mu_s)^T \sum_s^{-1}(c-\mu_s)} \tag{2.6}$$

où μ_s est le triplet (de couleurs) moyen et $\sum_s$ la matrice de covariance, tous les deux déterminés depuis les données d'entraînement. Il est ensuite possible de combiner plusieurs SGM afin de représenter une distribution plus complexe. La probabilité $P(c|skin)$ finale est alors obtenue en effectuant une somme pondérée des probabilités $P(c|skin)$ de chaque modèle simple

$$P(c|skin) = \sum_{i=1}^{k} \pi_i * P_i(c|skin) \tag{2.7}$$

où la somme des $\pi_i = 1$. La classification peut alors s'effectuer en comparant $P(c|skin)$ à un seuil défini.

Le modèle des limites elliptiques a été proposé afin de pallier les limites des modèles gaussiens simples tout en évitant un coût de traitement trop élevé comme c'est le cas avec les GMM. Le modèle elliptique utilise une fonction Φ pour déterminer l'appartenance d'une couleur à la classe peau. Pour une couleur c, $\Phi(c)$ sera calculée en utilisant notamment la valeur de c et la valeur moyenne des couleurs de peau présentes dans l'ensemble d'entraînement. Afin d'obtenir de bons résultats avec cette méthode, il est nécessaire de supprimer de l'ensemble d'entraînement les couleurs de peau pour lesquelles le nombre d'occurrences est trop peu élevé. Cette étape de suppression peut supprimer jusqu'à 5% des données d'entraînement. La classification à l'aide de ce modèle est finalement réalisée en comparant $\Phi(c)$ à un seuil Θ. Une explication détaillée de la méthode peut être trouvée dans [19].

Détection par forme

Les techniques de détection par forme sont capables de détecter les mains grâce à leurs contours.

D'abord, ces contours doivent être calculés. Généralement, ce calcul est réalisé en déterminant et en connectant les différents bords présents dans l'image. Un bord est défini comme un point de l'image où l'intensité des pixels change brusquement. La détection de ces bords est très souvent réalisée par la convolution de l'image à l'aide d'un opérateur ayant la propriété de retourner la valeur 0 pour les régions où l'intensité est uniforme. De nombreux opérateurs différents existent dont le Sobel ou l'opérateur de Prewitt [20]. Les détails de chacun de ces opérateurs sortant du cadre de la reconnaissance gestuelle, ceux-ci ne seront pas abordés. La détection des bords ne fait pas de distinction entre les bords provenant des parties du corps et ceux provenant d'autres objets présents dans l'image. Cela résulte donc généralement en un grand nombre de bords superflus détectés et donc, de bruit. Pour supprimer un maximum de ces bords superflus, cette méthode est souvent combinée à d'autres techniques comme la détection par couleur, les techniques de soustraction d'arrière-plan, etc. [28][34].

Une fois les contours des objets extraits, les mains peuvent être détectées en analysant les formes représentées par ces contours. Plusieurs méthodes d'analyse de ces formes ont été proposées [34]. La méthode la plus simple consiste à identifier la main directement depuis le résultat de la détection de contours. Cela fait toutefois l'hypothèse que seul le contour de la main est détecté. Cette méthode ne peut donc être considérée que dans le cas d'un environnement très contrôlé où aucun élément extérieur ne viendrait perturber la détection.

Certaines méthodes plus complexes visent à faire correspondre les différentes formes observées dans l'image à des images modèles pour détecter les mains. Dans [4], les auteurs proposent une méthode de comparaison de formes basée sur l'association d'un descripteur, "le contexte de forme", à chaque point de la forme observée et du modèle. Ce descripteur permet alors de caractériser les différents points p_i de la forme. Plus précisément, pour un

point p_i donné, le descripteur va fournir une représentation de la distribution des autres points de la forme relativement à p_i. La correspondance entre une forme observée et un modèle se base alors sur le fait qu'un point p de la forme observée possédera un contexte de forme semblable au point correspondant dans le modèle.

D'autres méthodes encore tentent de détecter les mains grâce aux particularités morphologiques de celles-ci. La détection des mains va alors être réalisée sur base de caractéristiques comme les bouts des doigts ou les doigts en général. Pour repérer ces caractéristiques, une approche est d'utiliser les courbures formées entre deux doigts. Cela implique cependant que ces courbures soient effectivement identifiables. Une autre approche est d'utiliser des techniques de "template matching" en utilisant, par exemple, des images représentant des doigts.

Détection par mouvement

Les techniques de détection basées sur le mouvement détectent les parties du corps d'intérêt grâce aux déplacements des différents objets durant la séquence vidéo. En comparant la position de ces objets d'une image à l'autre, il est possible de déterminer les objets mobiles. L'hypothèse est ensuite faite que seules les parties du corps d'intérêt bougent. Pour que cette hypothèse soit vérifiée, l'environnement doit être très fortement contrôlé. Pour cette raison, cette technique n'est utilisée que par peu d'approches de détection [28].

Pour pallier cette contrainte, certains travaux ont combiné les informations de mouvement avec d'autres telles que les informations de couleur et de forme. Dans [33] les auteurs se basent par exemple sur le principe selon lequel les changements d'apparence entre deux images sont plus importants pour les mains que pour les autres objets. Ces changements plus importants étant provoqués par le fait que les mains effectuent généralement des mouvements non rigides. Grâce à ce principe, les mains peuvent être détectées en comparant deux images consécutives et en identifiant les régions n'ayant que peu de ressemblance entre les deux images. D'abord, l'image précédente est divisée en un certain nombre de blocs. Ensuite, la correspondance de ces blocs est recherchée dans l'image analysée par translation. Finalement, les résidus de mouvement sont estimés pour chacun des blocs. Ces résidus correspondent à la différence moyenne entre l'intensité des pixels d'un bloc et l'intensité de sa correspondance. En raison des mouvements non-rigides des mains, les blocs contenant celles-ci possèdent généralement plus de résidus que les autres. En appliquant cette méthode, les auteurs obtiennent un certain nombre de zones de l'image pouvant contenir les mains. Ces zones sont alors filtrées afin de supprimer les faux positifs et de trouver la position des mains.

2.2.2 Suivi

La deuxième étape du processus de reconnaissance gestuelle est le suivi (ou tracking). Cette étape permet de réaliser la correspondance entre les régions détectées dans deux images successives. Étant donné deux images f_i et f_{i-1}, l'étape de suivi va tenter de faire correspondre les régions de l'image f_i avec celles de l'image f_{i-1}. Disposer d'une méthode

de suivi robuste est essentiel pour être en mesure d'extraire les caractéristiques du signe variant avec le temps, comme la trajectoire par exemple.

Plusieurs techniques de suivi existent. Nous nous concentrerons cependant sur les plus largement répandues. Cela inclut donc les filtres Kalman, l'algorithme de condensation et les méthodes basées sur l'algorithme MeanShift.

Filtres Kalman

La méthode des filtres Kalman est un des algorithmes de fusion de données le plus courant aujourd'hui [10]. La méthode peut être utilisée pour les systèmes pour lesquels certaines informations sont incertaines mais qu'il est malgré tout possible de réaliser une estimation de l'évolution du système. Les filtres Kalman sont donc idéaux pour les systèmes dynamiques changeant continuellement. L'un des principaux avantages de la méthode est que seul l'état précédent du système doit être conservé. En effet, l'estimation de l'état du système à l'instant t peut être entièrement défini sur base de l'état à l'instant $t-1$ et des observations faites à l'instant t. Il n'est donc pas nécessaire de conserver en mémoire l'historique complet des états du système.

D'un point de vue plus fonctionnel, l'estimation de l'état t fonctionne en deux temps :
- la prédiction de l'état e_t sur base des données de l'état e_{t-1} et des éventuelles forces extérieures connues.
- la mise à jour de la prédiction grâce aux données mesurées (ou observées) pour l'état e_t.

Il est important de noter que, en raison du fait que toutes les informations ne sont pas connues, les paramètres constituant l'état du système sont représentés à l'aide de fonctions de densité de probabilité (pdf) et non de valeurs discrètes. Plus précisément, ces fonctions sont des fonctions gaussiennes. C'est cette particularité qui permettra l'aspect récursif de la méthode, comme nous le verrons par la suite.

Connaissant l'état e_{t-1} (sous forme de pdf gaussienne), l'état à l'instant t peut être estimé en considérant les paramètres de e_{t-1}. Par exemple, dans le cas de l'estimation de la position d'un objet mobile, les paramètres d'un état e serait la position p et la vitesse v de l'objet. À l'instant t, la valeur de p serait estimée grâce à la valeur de p et de v à l'instant $t-1$. L'estimation des valeurs entre e_{t-1} et e_t provoque une réduction de la précision (précision de la position dans notre exemple). C'est pour cette raison que les paramètres de l'état du système à l'instant t sont également définis grâce aux observations de l'état réel du système à cet instant.

L'estimation finale des paramètres de l'état e_t est alors réalisée en combinant les valeurs estimés grâce à l'état e_{t-1} et les valeurs mesurées, c'est-à-dire les valeurs observées à l'instant t. Pour cela, les fonctions de densité de probabilités de l'estimation et des mesures sont multipliées l'une avec l'autre. Avant de réaliser cette multiplication, il est toutefois nécessaire de transformer la pdf de l'estimation afin de s'assurer que celle-ci possède bien le

même domaine que la pdf des mesures. Le résultat de cette multiplication est également une pdf gaussienne. Cela est du à l'une des propriétés de la multiplication des fonctions gaussiennes. Ce résultat peut donc servir directement pour réaliser l'estimation de l'état e_{t+1}, la méthode peut donc être utilisée de manière complètement récursive.

Condensation

La méthode des filtres Kalman est très efficace lorsqu'il s'agit de suivre des objets évoluant dans des environnements peu encombrés, contenant peu d'objets autres que celui suivi. Cependant, lorsque l'environnement est fortement encombré, certains objets peuvent présenter des caractéristiques similaires à l'objet suivi. Cela a pour conséquence que la fonction de distribution des états possibles de l'objet à un instant t ne peut plus être représentée sous forme d'une fonction gaussienne. Pour tenter de résoudre ce problème des filtres Kalman, Isard et Blake [13] ont proposé l'algorithme de condensation.

À chaque instant t, l'algorithme produit une estimation de la distribution de probabilité de l'état x_t sachant que les observations $z_1, ..., z_t$ ont été faites ($p(x_t|z_1, ..., z_k)$). Pour éviter de traiter tous les pixels de l'image, l'algorithme produit cette estimation sur base d'un échantillon de points. En outre, l'échantillon utilisé est choisi aléatoirement. Comme la méthode Kalman, l'algorithme de condensation fonctionne de manière itérative. À chaque instant t, la densité de probabilité est estimée grâce aux informations de l'état précédent et aux observations faites jusque là (les z_t incluent). Cette estimation est ensuite corrigée grâce aux z_t. Le cas du moment $t = 0$ consiste en une étape d'initialisation se basant sur une distribution préalable pouvant, par exemple, être une gaussienne. Finalement, la densité de probabilité produite peut être utilisée pour calculer diverses informations comme, par exemple, la position la plus probable de l'objet.

MeanShift et CamShift

L'algorithme du MeanShift [11] est un algorithme non paramétrique et itératif permettant notamment de trouver les modes d'une distribution de probabilité en déplaçant une fenêtre (de taille fixe) le long de cette distribution. La première étape, l'initialisation, consiste à fixer la taille de la fenêtre et à spécifier la position initiale de celle-ci. Ensuite, les trois étapes suivantes sont répétées jusqu'à ce que le déplacement de la fenêtre soit plus petit qu'un certain seuil ou que le nombre d'itérations maximum autorisé soit atteint.

- La moyenne pondérée m de la densité au sein de la fenêtre est calculée. Le poids accordé à chaque point (x, y) de la densité est défini grâce à un noyau K.
- Le vecteur "mean-shift" est ensuite calculé en calculant la distance entre la position actuelle de la fenêtre et les coordonnées de m.
- Finalement, la position de la fenêtre subit un déplacement correspondant au vecteur "mean-shift". La position de la fenêtre est donc maintenant la position de m.

Lorsque la fenêtre n'a pas été déplacée de plus d'un certain seuil, celle-ci est considérée comme ayant convergé. Le maxima local de la densité a donc été trouvé. Dans le cadre du traitement d'images, la densité considérée correspond à une image dans laquelle chaque pixel s'est vu attribuer comme valeur une probabilité (par exemple la probabilité d'être un

pixel de peau). Les différents traitements sont alors réalisés sur la valeur des pixels et la fenêtre est déplacée le long de l'image.

Lors de l'application de l'algorithme pour le suivi, la position initiale de la fenêtre est initialisée à la dernière position connue de l'objet suivi. La suite de l'algorithme déplacera alors (idéalement) la fenêtre vers la nouvelle position de l'objet. La figure 2.4 illustre l'application de l'algorithme dans le cas de la détection d'un visage.

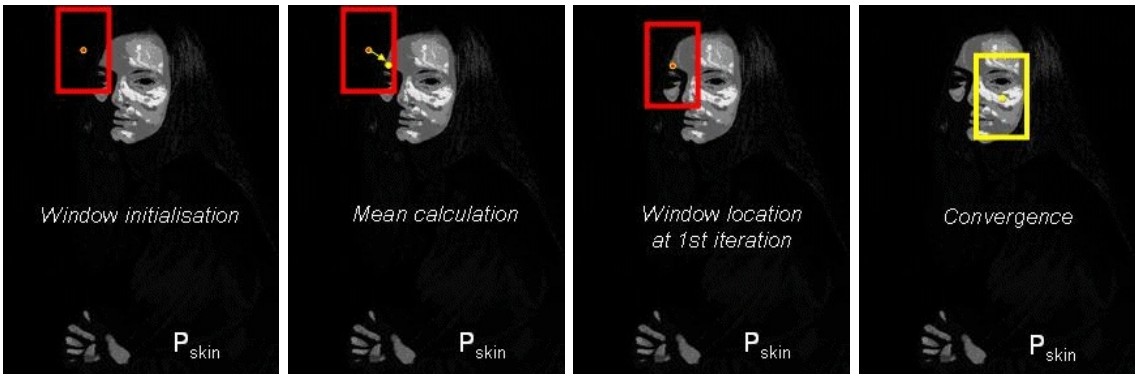


FIGURE 2.4 – Algorithme MeanShift (initialisation, première itération et résultat) appliqué à la détection de visage. Autheur : Rachid Belaroussi.

L'un des inconvénients majeurs du MeanShift est que la taille de la fenêtre est fixe tout au long de la procédure. Ainsi, si la taille de l'objet suivi change (par exemple car celui-ci se rapproche ou s'éloigne de la caméra), la taille de la fenêtre risque de ne plus être adéquate. Pour pallier ce problème, l'algorithme CamShift [5] peut être utilisé. Le CamShift applique le MeanShift jusqu'à arriver à convergence. Ensuite, la taille de la fenêtre est mise à jour. Dans le cas de l'application au traitement d'images, cette mise à jour est réalisée sur base des axes d'une ellipse comme illustré par la figure 2.5. D'abord, la taille de la fenêtre est augmentée de $\pm$ 10 pixels verticalement et horizontalement. Ensuite, l'ellipse correspondant le mieux à cette nouvelle fenêtre est calculée. Finalement, la taille de la fenêtre est adaptée pour correspondre à la taille de l'ellipse. Le MeanShift est alors ré-appliqué en considérant la nouvelle taille de la fenêtre. Ce processus se poursuit ainsi jusqu'à arriver à convergence comme dans le cas du MeanShift.

FIGURE 2.5 – Adaptation de la taille de la fenêtre dans le cas du CamShift appliqué au traitement d'images. Autheur : Rachid Belaroussi.

2.2.3 Reconnaissance

L'objectif de l'étape de reconnaissance est d'interpréter les caractéristiques extraites durant les étapes de détection et de suivi afin de reconnaitre le geste réalisé. De nombreuses méthodes existent pour réaliser cette interprétation [28]. Parmi celles-ci, nous aborderons la méthode des machines à vecteurs de support (SVM), les méthodes basées sur les automates (à savoir les modèles de Markov cachés et les machines à états finies) ainsi que la déformation temporelle dynamique (DTW).

Machines à vecteurs de support (SVM)

L'algorithme des machines à vecteurs de support est un algorithme de classification supervisé visant à déterminer la classe d'une donnée reçue en entrée parmi deux classes possibles. Puisqu'il s'agit d'un algorithme supervisé, celui-ci nécessite des données d'entraînement labelisées. Grâce à ces données, un hyperplan [5] maximisant la séparation entre les deux classes est construit. Pour cela, les données sont représentées sous forme de points dans l'espace. La séparation optimale entre les clusters formés par les deux classes est ensuite déterminée en cherchant les deux points de ces classes les plus proche l'un de l'autre. Le vecteur entre ces deux points est alors calculé. La séparation optimale est finalement trouvée en considérant la droite coupant ce vecteur à sa moitié et étant perpendiculaire à celui-ci.

La figure 2.6 illustre le concept de séparation optimale. Sur l'image de gauche, plusieurs séparations possibles entre les deux classes d'objets sont représentées. L'image de droite présente la séparation maximisant la distance entre les classes.

5. Un hyperplan est un sous-espace de l'espace ambiant et possédant une dimension de moins que ce dernier.

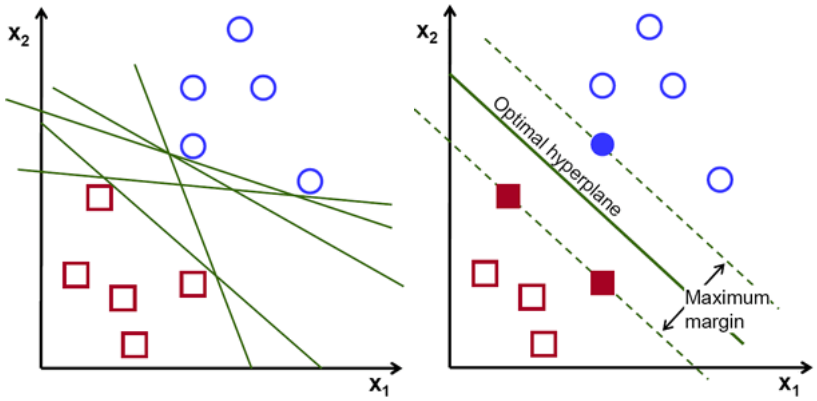


FIGURE 2.6 – Concept de séparation optimale pour l'algorithme SVM. A gauche, les séparations possibles. A droite la séparation optimale. Images reproduites de : `http://docs.opencv.org/2.4/doc/tutorials/`.

Modèles de Markov cachés (HMM)

La technique des modèles de Markov cachés représente le système sous la forme d'un processus de Markov. À chaque instant t, le système se trouve dans un état particulier. Chaque état E peut produire des observations O_i avec une probabilité $P(O_i|E)$ et faire transiter le système vers un certain nombre d'états. À chaque transition d'état est également associée une probabilité et la transition d'un état à un autre ne dépend pas des états précédents. La particularité des HMM par rapport aux processus de Markov classiques provient du fait que les états par lesquels passe le système ne sont pas directement visibles. Seuls les résultats produits par ces états (les observations) sont visibles. L'algorithme a alors pour objectif de déterminer la séquence d'états la plus probable en utilisant les différentes probabilités $P(O_i|E)$ et les probabilités de transition. Ainsi, étant donné une séquence d'observations $O_0, O_1, ..., O_n$, l'algorithme commence par définir les états du système pouvant produire l'observation O_0. Ensuite, pour chacun de ces états e_i, la liste des états suivants est déterminée, c'est-à-dire la liste des états e'_j pouvant donner lieu à l'observation O_1 et pour lesquels une transition $e_i -> e'_j$ existe. Ce processus est répété jusqu'à avoir traité toutes les observations. L'algorithme a alors produit une liste de séquences d'états possibles. L'étape finale est de déterminer, sur base des différentes probabilités, la séquence la plus probable. Dans le cas de l'exemple illustré par la figure 2.7 (et repris de `https://en.wikipedia.org`) deux séquences d'états possibles sont par exemple les séquences $(5, 3, 2, 5, 3, 2)$ et $(4, 3, 2, 5, 3, 2)$.

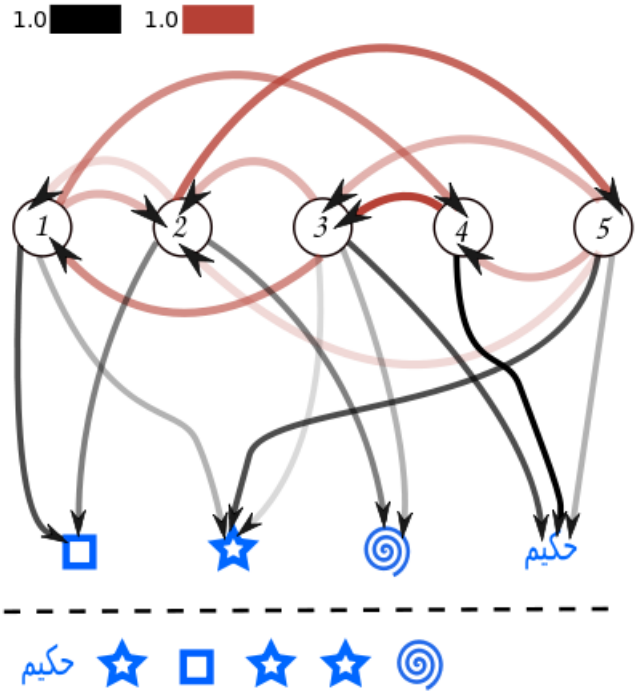


FIGURE 2.7 – Exemple de modèle de Markov et d'une série d'observations. Auteur : Hakeem Gadi.

Lorsqu'appliqué au cas de la reconnaissance gestuelle, un modèle de Markov est construit pour chacun des gestes S devant être reconnu. Pour chacun de ces gestes S le modèle associé représente ainsi les différentes séquences de configurations (forme, position, etc.) de la main pouvant représenter le geste S. Les gestes peuvent alors être reconnus en traitant les données reçues en entrée vis-à-vis des différents modèles et en considérant celui ayant la plus grande probabilité.

Machines à états finies

Lorsque la reconnaissance est réalisée avec des machines à états finies, une machine est construite grâce à un grand nombre d'exemples de réalisations des gestes possibles. Lorsqu'un geste doit être reconnu, la séquence d'états observée est confrontée à la machine construite précédemment. À chaque traitement d'un état observé, l'algorithme décide s'il doit rester dans l'état actuel de la machine ou transiter vers un autre. Si au terme de la séquence l'algorithme a atteint un état final, le geste a été reconnu.

Déformation temporelle dynamique

L'algorithme de déformation temporelle dynamique (DTW) permet de comparer deux séquences de données. La particularité et l'intérêt majeur de l'algorithme provient du fait

que ces deux séquences n'ont pas besoin d'être d'une longueur identique. Pour cette raison, le DTW est très souvent utilisé dans les domaines de la reconnaissance vocale et gestuelle. Dans ces domaines, il n'est en effet pas rare que les données observées soient réalisées plus rapidement (ou plus lentement) que les données en mémoire. Les séquences représentant ces données ne sont donc pas de taille identique et ne peuvent pas être comparées simplement élément par élément.

Pour calculer une mesure de similarité entre les deux séquences, le DTW va chercher à les aligner afin de trouver la correspondance optimale entre celles-ci. Soit deux séquences s et s' respectivement de taille m et n. La correspondance optimale entre s et s' sera trouvée en construisant une matrice de taille m par n. Chaque élément (x, y) de cette matrice se verra ensuite assigner la valeur du coût d'alignement cumulé entre l'élément s_x et l'élément s'_y. Ce coût cumulé correspond à la distance entre s_x et s'_y à laquelle est additionné le coût minimum pour arriver à l'élément (x, y). Comme l'alignement des deux séquences ne peut se faire qu'en avançant d'un pas à la fois, c'est-à-dire que x et y ne peuvent être incrémentés que de 0 ou 1 entre chaque étape, cela nous donne la formule suivante :

$$CoutCumule(x, y) = distance(s_x, s_y) + Min((x-1, y), (x, y-1), (x-1, y-1)) \quad (2.8)$$

La distance entre les éléments de s et de s' peut être calculée de différentes façons. Par exemple, dans le cas de coordonnées une distance possible est la distance euclidienne. Finalement, la correspondance optimale entre les deux séquences correspond au chemin partant de l'élément $(0, 0)$ jusqu'à l'élément (m, n) telle que la distance cumulée de chaque élément est la plus petite possible. La mesure de similarité entre les deux séquences correspond alors à la valeur contenue dans la cellule (m, n) de la matrice. Plus cette valeur est petite, plus les séquences sont semblables.

Chapitre 3

Background technologique

3.1 Technologies de reconnaissance gestuelle

Deux grands types de technologies de reconnaissance gestuelle existent. Le premier type concerne les technologies orientées contact se basant sur une interaction physique de l'utilisateur pour recueillir les données. Ces technologies sont généralement très précises mais elles sont également très intrusives. Le deuxième type reprend les technologies orientées vision. Ces technologies récupèrent les données depuis des flux vidéos et ne nécessitent ainsi pas (ou peu) d'interaction de la part de l'utilisateur. L'objectif du laboratoire LSFB étant de disposer de vidéos représentant des conversations les plus naturelles possibles, les technologies orientées contact ont été écartées dès le début du projet. Pour cette raison, nous n'aborderons, dans cette section, que les technologies orientées vision.

Les périphériques orientés vision utilisent les informations provenant de flux vidéos pour reconnaitre les gestes réalisés par l'utilisateur. Ces flux vidéos peuvent provenir de caméras couleurs ou de caméras plus complexes comme des caméras infrarouges (IR) capables de fournir des informations 3D. En outre, certains périphériques, comme Kinect, embarquent des algorithmes leur permettant de fournir des informations de plus haut niveau comme, par exemple, la présence de personnes. Enfin, il est parfois demandé à l'utilisateur de porter des marqueurs de couleur pour rendre le traitement des données vidéos plus aisé. Les périphériques orientés vision les plus courants sont la caméra traditionnelle et Kinect de Microsoft. Les vidéos actuelles du corpus LSFB ont été tournées avec de simples caméras. Cependant, il n'est pas exclu que de nouvelles vidéos soient tournées en utilisant Kinect. Pour cette raison, nous avons décidé d'aborder les deux technologies dans cette section.

3.1.1 Caméra traditionnelle

La caméra couleur constitue probablement le périphérique de reconnaissance gestuelle le plus abordable et le plus disponible pour le grand public. En effet, de nombreux appareils (laptops, smartphones, etc.) en sont aujourd'hui équipés et il est également possible d'en acquérir une pour une cinquantaine d'euros.

En raison de sa faible complexité, l'utilisation de la caméra traditionnelle pour la reconnaissance gestuelle peut poser certains problèmes. D'une part, les seules données fournies par ces caméras concernent la couleur de chaque pixel des images filmées. Il est donc nécessaire d'implémenter (ou d'inventer si nécessaire) les algorithmes permettant, sur base de ces informations, d'inférer des informations de plus haut niveau comme la position des parties du corps. Dans certains cas, où l'environnement est idéal, ces informations de haut niveau pourront être inférées très facilement. En revanche, si le système est utilisé dans un environnement engendrant beaucoup de bruit visuel, cette étape pourra constituer une tâche bien plus complexe. D'autre part, la caméra ne fournissant pas d'information de profondeur, la résolution des problèmes d'occlusion, c'est-à-dire lorsqu'une partie du corps passe devant une autre, constituent généralement un défi de taille.

Il est cependant possible de pallier ces problèmes en demandant à l'utilisateur de porter des marqueurs de couleur permettant ainsi d'identifier sans ambiguïté les différentes parties du corps. Dans cette optique, Robert Y. Wang et Jovan Popović ont proposé un concept de gants de couleurs [31] permettant d'identifier facilement chaque partie de la main. La figure 3.1 illustre ce concept. L'objectif principal de ces gants est de faciliter l'estimation de

FIGURE 3.1 – Gants de couleurs proposés par Robert Y. Wang et Jovan Popović

la "pose", c'est-à-dire de rendre l'estimation de la position et de l'orientation de la main plus aisée. La figure 3.2 permet de se rendre compte de ce problème. Les deux images de gauche représentent une main non gantée avec la paume d'abord orientée vers le bas et ensuite vers le haut. D'un point de vue purement visuel, la distinction entre les deux orientations est assez faible et même probablement trop faible pour pouvoir être distinguée par un ordinateur. Les images de droite présentent les mêmes configurations de la main mais avec cette fois l'utilisation du gant coloré. Contrairement à la main nue, la distinction entre les deux orientations est beaucoup plus claire et peut être identifiée par un ordinateur en se basant sur les couleurs visibles.

3.1.2 Microsoft Kinect

Kinect est un périphérique de détection de mouvement fabriqué par Microsoft. A l'origine conçue pour permettre aux joueurs d'interagir de manière plus naturelle avec la console Xbox, Kinect a rapidement été adoptée par les développeurs et chercheurs afin de concevoir

FIGURE 3.2 – Illustration du problème de "pose". A gauche la visualisation d'une main nue paume baisée et paume levée. A droite, la visualisation de cette même main mais cette fois-ci, gantée. Image reproduite de [31].

des applications dans divers domaines tels que la médecine [27][23], l'aide aux personnes âgées [22] et bien sûr la reconnaissance de la langue des signes [32][15].

Kinect est équipé d'une caméra couleur classique, d'émetteurs infrarouges et d'un capteur de profondeur. Grâce à ces composants, Kinect est capable de fournir les informations de couleur, comme le ferait une caméra traditionnelle, mais également les informations de profondeur ainsi que les informations infrarouges (permettant un fonctionnement dans le noir complet). En outre, Kinect est en mesure de calculer des informations de plus haut niveau comme la position et l'orientation des différentes parties du corps. Ces informations sont alors mises à la disposition des développeurs via un kit de développement logiciel (SDK) fourni par Microsoft.

Flux couleurs

La caméra couleur de Kinect est une caméra haute résolution (1920 x 1080 pixels) permettant d'obtenir toutes les informations que fournirait une caméra traditionnelle. En outre, le SDK de Kinect permet d'obtenir les données filmées sous différents formats :

- Rgba (Red-Green-Blue-Alpha) : Il s'agit de l'espace de couleur RGB (Red-Green-Blue) traditionnel auquel une information de transparence (Alpha) a été rajoutée.
- Bgra (Blue-Green-Red-Alpha) : Ce format est identique au Rgba à ceci près que l'ordre des différents composants est différent.
- Bayer : Une image au format Bayer est obtenue en appliquant un filtre sur la caméra de sorte que chaque pixel ne puisse capter qu'une seule des trois couleurs du modèle RGB. La répartition du nombre de pixels pour chaque composante n'est cependant pas uniforme. En effet, la moitié des pixels reçoivent la composante verte tandis que la moitié restante est partagée entre le rouge et le bleu. L'application de ce filtre sur la caméra est nécessaire à l'obtention d'images couleurs. La transformation du format Bayer au format RGB se fait à l'aide d'un processus ("Demosaicing") permettant de dériver les informations de couleur complètes grâce aux pixels voisins [7]
- Yuv : Le YUV est un espace de couleur encodant celles-ci à l'aide de 3 composants. le composant Y reprend les informations de luminosité tandis que le composant U et V les informations chromatiques.

— Yuy2 : Le format YUY2 est un format YUV 8-bits récommandé pour le rendu vidéo.

Flux de profondeur

Le flux de profondeur permet d'obtenir une image (d'une résolution de 512 x 424 pixels) dans laquelle la valeur de chaque pixel représente la distance (en millimètres) entre l'objet le plus proche et Kinect relativement à la position de ce pixel. Grâce à ce flux, Kinect est en mesure de fournir les informations 3D de la scène filmée et donc, de faciliter la résolution des problèmes d'occlusion.

Pour obtenir les informations de profondeur, Kinect se base sur le principe de "time-of-flight" (ToF). Ainsi, pour calculer la distance entre Kinect et un objet de la scène, un signal lumineux infrarouge est émis. La distance entre l'objet et Kinect est proportionnelle au temps de retour de ce signal. Plus celui-ci est important et plus la distance l'est. Connaissant la vitesse de la lumière, il est alors possible d'estimer cette distance. Bien que, techniquement, la distance maximale pouvant être mesurée est de 8 mètres, la fiabilité commence à décroître à partir de 4,5 mètres.

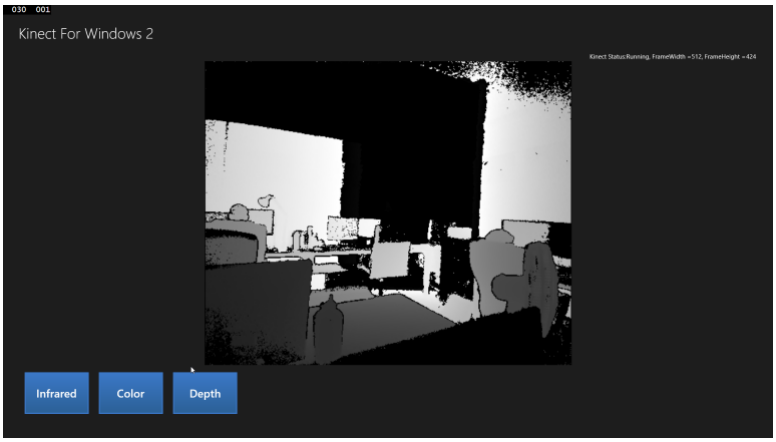


FIGURE 3.3 – Représentation du flux de profondeur. Image reproduite de `http://kinect.github.io/tutorial/lab04/index.html`.

Flux infrarouge

Le flux infrarouge permet d'obtenir une image de la scène en nuances de gris. L'avantage de ce flux est que celui-ci est capable de fonctionner malgré l'absence totale de lumière. Le flux infrarouge est très utile pour les algorithmes où les textures sont importantes (comme la reconnaissance faciale), pour suivre des marqueurs réflectifs ou pour filtrer les pixels dont les informations de profondeur sont de faible qualité. Ce flux provenant des mêmes capteurs que le flux de profondeur, les images provenant de ces deux flux sont parfaitement alignées. Cela signifie qu'il est possible de faire correspondre un pixel du flux infrarouge à un pixel du flux de profondeur sans avoir besoin de convertir les coordonnées de ceux-ci.

Détection des corps

Kinect est capable de détecter jusqu'à six corps (à une distance allant de 0,5 à 4,5 mètres) et de reconnaitre 25 articulations pour chacun d'eux comme l'illustre la figure 3.4. Pour chaque articulation, Kinect est en mesure de fournir des informations sur la position, l'orientation et l'état du suivi. Ce dernier élément permet de savoir dans quelles mesures les informations fournies par Kinect sont fiables. Trois états d'articulation différents sont possibles :

— Traquée : Cela signifie que Kinect est en mesure de suivre l'articulation et que les données sont fiables.
— Inférée : Dans ce cas, Kinect n'est pas en mesure de suivre l'articulation et les données sont inférées grâce à la position des autres articulations.
— Non traquée : Cet état implique que Kinect n'est pas en mesure de voir l'articulation et ne peut inférer de données grâce aux autres articulations.

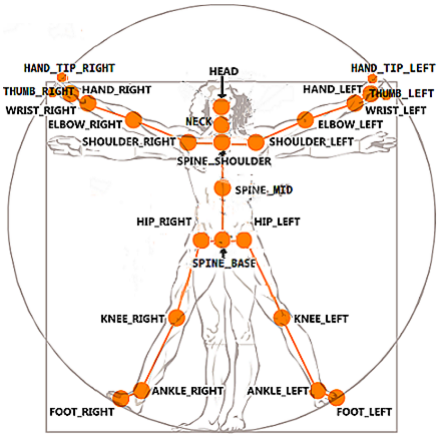


FIGURE 3.4 – Représentation des 25 articulations détectées par Kinect. Image reproduite de `https://msdn.microsoft.com/en-us/library/microsoft.kinect.jointtype.aspx`.

Outre le suivi des corps, Kinect est capable, depuis la version 2, de suivre l'état des mains des utilisateurs selon certaines configurations prédéfinies. Cinq états différents sont ainsi possibles : Ouvert, fermé, lasso, non suivi et inconnu. Une main dans l'état "lasso" est une main fermée avec l'index et le majeur tendus. Il s'agit donc d'un geste de pointage réalisé avec deux doigts. La nécessité d'utiliser deux doigts pour ce signe de pointage provient du fait qu'un seul doigt n'est généralement pas assez large pour être reconnu. Cette contrainte peut cependant être levée si la main est suffisamment proche du capteur.

Enfin, Kinect est également capable de fournir une image dans laquelle les corps détectés sont distingués des éléments du décor. Pour cela, chaque pixel a comme valeur soit l'identifiant du corps auquel il appartient (de 0 à 5) soit une valeur plus grande que le nombre de corps détectés. Cette image est particulièrement utile lorsque l'on souhaite obtenir des informations n'étant pas fournies directement par la Kinect. En effet, celle-ci permet de

ne pas devoir se soucier des problèmes de suppression de décor lors de la mise en place d'algorithmes de reconnaissances plus poussés que ceux embarqués par Kinect.

3.2 Espaces de couleur

Un espace de couleur est une manière de représenter les différentes couleurs existantes. L'objectif de ces espaces est de fournir un standard de représentation de la couleur adapté à certains domaines d'application. En pratique, ces espaces sont tri-dimensionnels, c'est-à-dire qu'ils représentent les couleurs possibles à l'aide de trois composantes.

Il a été prouvé qu'en théorie chaque espace de couleur est capable de fournir des performances équivalentes dans le domaine de la détection de peau, à condition qu'un classificateur optimal existe pour chacun d'eux [2]. Cependant, en pratique, certains espaces sont plus adaptés que d'autres en raison des composantes de représentation choisies. Parmi les espaces existants, les plus utilisés dans les techniques de détection de peau sont [24] :

— L'espace RGB et l'espace RGB normalisé ;
— Les espaces perceptuels ;
— Les espaces orthogonaux ;
— Les espaces perceptuellement uniformes.

L'espace RGB et l'espace RGB normalisé

L'espace **RGB** est l'espace de couleur le plus couramment utilisé pour représenter les couleurs dans le monde numérique. Dans cet espace, chaque couleur est représentée comme une combinaison de rouge (R), de vert (G) et de bleu (B). La figure 3.5 illustre l'espace RGB et permet d'avoir une meilleure idée de la représentation de chaque couleur en terme de rouge, vert et bleu.

Malgré qu'il soit fortement répandu pour représenter la couleur numériquement, cet espace n'est pas le meilleur choix pour les applications de détection de peau. Cela s'explique en raison de la forte corrélation entre les différentes composantes et de la non séparation des informations chromatiques et de luminance [30].

L'espace **RGB normalisé (rgb)** a pour objectif de réduire les inconvénients de l'espace RGB vis-à-vis de la détection de peau. Pour cela, chaque composante est normalisée en divisant sa valeur par la somme des valeurs des composantes R, G et B. La somme des trois composantes ainsi obtenue étant fixe (valant 1), il est possible d'omettre la troisième composante sans perdre d'information.

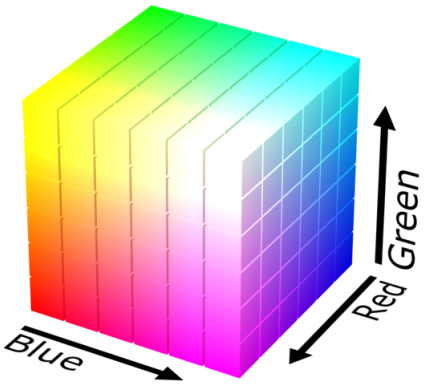


FIGURE 3.5 – Représentation graphiques de l'espace RGB. Image reproduite de `https://commons.wikimedia.org/`, auteur : SharkD.

Les espaces perceptuels

Les espaces perceptuels ont été conçus afin de disposer d'une représentation de la couleur plus intuitive pour l'homme permettant un encodage manuel plus facile pour les utilisateurs d'application graphiques. Ainsi, ces espaces représentent la couleur à l'aide d'une composante de teinte (hue), d'une composante de saturation et d'une composante de luminosité. La teinte permet de définir la couleur dominante. La composante de saturation définit quant à elle la vivacité de la couleur. Une saturation nulle résultera en une couleur grise. Finalement, la composante de luminosité permet de faire varier la couleur du noir vers le blanc.

Les espaces perceptuels les plus courants sont le HSV (Hue-Saturation-Value) et le HSL (Hue-Saturation-Lightness). Ces deux espaces se distinguent l'un de l'autre par la façon dont la composante de luminosité est calculée depuis les composantes RGB ainsi que par la façon dont la saturation est calculée depuis la luminosité. La figure 3.6 représente graphiquement les espaces HSV et HSL, permettant de visualiser l'impact de ces différences.

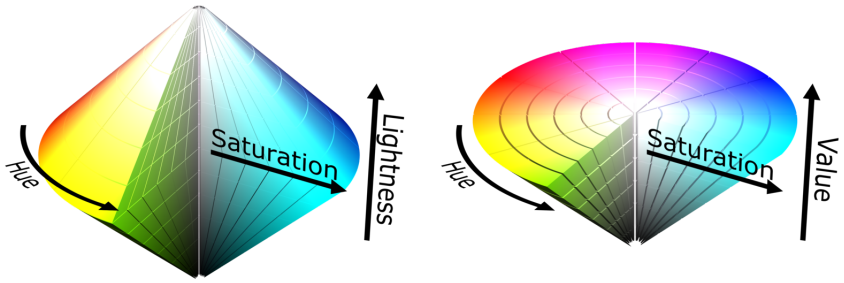


FIGURE 3.6 – Représentations graphiques des espaces de couleurs HSL (à gauche) et HSV (à droite). Images reproduites de `https://commons.wikimedia.org/` auteur : SharkD

Les espaces orthogonaux

Les espaces orthogonaux modélisent la couleur à l'aide d'une composante de luminance (appelée Y) et de deux composantes chromatiques. Ces espaces étaient au départ utilisés par les téléviseurs afin de pouvoir transmettre des informations de couleurs tout en restant compatible avec les téléviseurs noir et blanc (n'utilisant que la composante Y).

L'espace orthogonal le plus couramment utilisé dans le traitement d'images est l'espace YCbCr. Les composantes chromatiques de cet espace sont obtenues en soustrayant la composante Y des composantes R et B (obtenues du modèle RGB).

Les espaces perceptuellement uniformes

Les espaces perceptuellement uniformes ont été conçus de sorte de représenter la couleur telle que la verrait un observateur humain. Ainsi, dans ces espaces, la distance entre deux couleurs est proportionnelle à la différence de perception de ces deux couleurs par un observateur humain. Les espaces CIE-Lab et CIE-Luv sont deux espaces satisfaisant cette propriété.

Troisième partie

Contribution

Chapitre 4

Analyse des besoins

Le laboratoire de langue des signes de Belgique francophone (LSFB-lab) travaille actuellement sur le corpus LSFB, premier large corpus informatisé illustrant l'usage actuel de la LSFB et disponible en accès libre. Le corpus contient actuellement 150h de vidéo où 100 signeurs de profils variés âgés de 18 à 95 ans utilisent la LSFB. Les vidéos sont réparties en sessions. Une session reprend toutes les discussions entre les deux mêmes signeurs. Chaque session est ensuite divisée en une vingtaine de tâches. Une tâche représente une discussion autour d'un sujet précis. Les vidéos de ces discussions doivent être annotées et ensuite traduites. Ce travail, commencé il y a plusieurs années, demande énormément de temps.

Actuellement, l'annotation est réalisée manuellement à l'aide du logiciel ELAN (comme illustré par la figure 4.1). Ainsi, pour réaliser l'annotation d'une discussion, les annotateurs doivent, pour chaque signe présent dans la vidéo, délimiter les temps de début et de fin du signe et attribuer un "tag" (représentant le signe) à l'intervalle ainsi défini.

Malgré la possibilité de réaliser l'annotation de deux vidéos en parallèle (une vidéo par signeur impliqué dans la discussion), le temps nécessaire à l'annotation est extrêmement long. En effet, environ huit heures de travail sont nécessaires pour annoter deux minutes de vidéo. À priori, ce temps peut sembler excessif. Cependant, une discussion comporte en moyenne 660 signes et chaque signe dure environ une demi seconde. Les annotateurs sont donc amenés à travailler à l'échelle de la milliseconde et à re-visionner plusieurs fois certains passages des vidéos afin de déterminer les temps de début et de fin de chaque signe avec suffisamment de précision. À l'heure de l'écriture de ce document, douze heures ont été annotées depuis le lancement du projet en 2015. En considérant deux ans pour annoter douze heures, il est possible d'extrapoler le temps d'annotation "manuelle" restant à environ vingt ans.

Outre la charge de travail importante que représente chaque conversation, l'un des problèmes actuels de l'annotation réside dans la qualification requise par les annotateurs. En effet, seule une personne bilingue LSFB-français est capable de reconnaitre les gestes présents dans une vidéo. Cette contrainte réduit donc le nombre d'annotateurs potentiels et peut, de ce fait, encore augmenter le temps nécessaire à l'annotation du corpus complet.

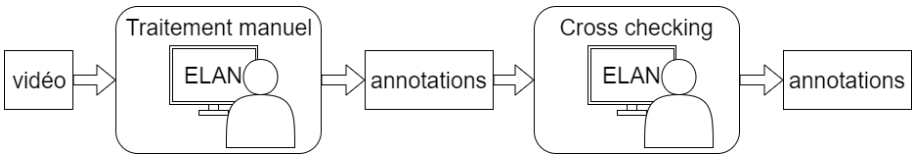


FIGURE 4.1 – L'annotation manuelle d'une vidéo.

Pour chaque vidéo annotée, une phase de vérification croisée (cross-checking) est réalisée. Durant cette phase, l'annotation de la vidéo est vérifiée (et corrigée si nécessaire) par un annotateur autre que celui ayant réalisé l'annotation initiale. Cette phase augmente évidemment le temps nécessaire à l'annotation du corpus complet. Cependant, par rapport à la phase d'annotation, le temps nécessaire à la vérification est relativement minime. Ce temps, même limité, passé à la vérification permet de s'assurer que le corpus proposé est de qualité.

Au vu de la charge de travail respective des phases d'annotation et de cross-checking, il semble évident que la phase d'annotation doit être améliorée. Pour remplir cet objectif, deux axes de solutions possibles ont été identifiés :

— Développer des add-on pour le programme d'annotation afin de faciliter et d'accélérer son utilisation ;
— Produire de manière automatique ou semi-automatique les annotations pour les vidéos, soit grâce à des technologies comme Kinect, soit en traitant les images directement.

Après observation des annotateurs durant leur travail, il a été déterminé que l'ajout d'add-on dans ELAN ne permettrait pas d'améliorer efficacement la phase d'annotation. En effet, les annotateurs travaillent déjà aussi vite qu'il est humainement possible et n'hésitent pas dans leur utilisation du logiciel. En outre, l'ensemble des tags utilisés pour l'annotation a déjà été intégré dans ELAN. Il est dès lors difficile d'imaginer un add-on permettant d'accélérer le processus d'annotation sans automatiser au moins une partie de celui-ci.

La solution envisagée se concentrera donc sur le second axe de solution : produire de manière automatique ou semi-automatique les annotations pour les vidéos, soit grâce à des technologies comme Kinect, soit en traitant les images directement. La figure 4.2 illustre le processus d'annotation avec la solution envisagée.

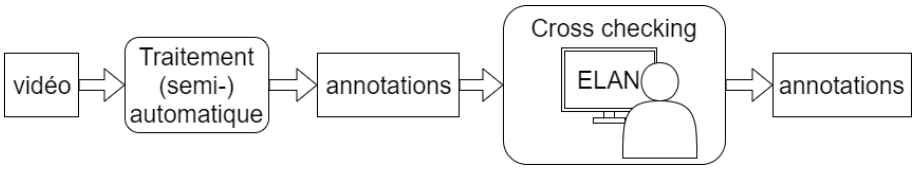


FIGURE 4.2 – L'annotation (semi-)automatique d'une vidéo.

En raison de ses capteurs et de ses algorithmes Kinect permet la mise en place aisée d'une solution de reconnaissance gestuelle efficace. En effet, comme expliqué au point 3.1.2, Kinect embarque certains algorithmes de détection et de suivi des différentes parties du corps. L'utilisation de ceux-ci permet alors de concentrer le travail d'implémentation sur les aspects de reconnaissance des gestes du domaine considéré. En outre, si des algorithmes plus poussés que ceux embarqués doivent être développés, par exemple pour mettre en place une détection plus poussées des mains, Kinect fournit d'avantage d'informations qu'une caméra couleur. Par exemple, les informations de profondeurs peuvent être récupérées grâce au capteur infrarouge. Ces informations peuvent alors servir à distinguer très facilement les parties du corps étant visuellement confondues. Toutefois, malgré ses avantages, Kinect a été rapidement éliminé car son utilisation ne permettait pas d'aider à l'annotation des vidéos constituant actuellement le corpus. En effet, ces vidéos ont été tournées avec de simples caméras couleurs. L'utilisation de Kinect nécessitait donc de reproduire l'ensemble des discussions des vidéos face à l'appareil. De plus, si ces reproductions pouvaient être annotées en tirant profit des avantages de Kinect, un travail supplémentaire aurait du être fourni afin de déterminer les temps de début et de fin de celles dans les vidéos du laboratoire. Pour déterminer, les annotateurs auraient probablement du spécifier le début et la fin de chaque signe présent dans les vidéos originales. Au final, en considérant ces deux étapes, la charge de travail des annotateurs ainsi que le temps d'annotation auraient probablement augmenté. Pour cette raison, nous avons préféré explorer une solution basée sur l'application des techniques de traitement automatique d'images.

Pour pouvoir proposer une solution adaptée, les ressources nécessaires à l'annotation doivent être observées et analysées. Cette phase d'analyse permet de bien définir les difficultés pouvant impacter le développement de la solution. Énormément d'observations peuvent être faites depuis les vidéos filmées par le LSFB-lab, mais il est également important de discuter avec les personnes travaillant au laboratoire.

4.1 Observation des vidéos

Lors de l'observation des vidéos filmées par le LSFB-lab, plusieurs particularités pouvant impacter le développement d'une solution d'annotation automatique ont été identifiées. Ces particularités sont essentiellement liées au cadrage, aux conditions d'éclairage et à l'habillement des personnes filmées. L'objectif initial n'étant pas de réaliser une annotation automatique, les vidéos n'ont pas été réalisées en considérant les défis auxquels font face les techniques de reconnaissance gestuelle.

4.1.1 Occlusion

Durant la réalisation des signes, les mains peuvent se croiser ou passer devant la tête. Visuellement, cela implique que l'une des mains (ou la tête) est masquée en partie (parfois totalement) par la seconde. C'est ce qui est appelé occlusion.

Les occlusions sont un problème courant dans la reconnaissance automatique. En effet, lorsque cela arrive, la distinction entre les deux parties du corps impliquées peut être difficile

à réaliser.

FIGURE 4.3 – Exemple d'occlusion entre la tête et la main droite (images originales provenant de [21]).

4.1.2 Sortie de cadre

Lorsqu'elles ne signent pas, les personnes filmées placent leurs mains dans une position pouvant être qualifiée de repos. Cette position se traduit généralement par un croisement des bras ou un positionnement des mains à hauteur des cuisses ou des genoux. Dans ce dernier cas, il arrive fréquemment que les mains ne soient plus visibles à l'image.

Les moments d'inactivité du signeur ne sont pas les seules causes de la sortie de cadre des mains. Lors de certains grands signes les mains peuvent également ne plus être visibles pendant un certains laps de temps. Cela peut, par exemple, être provoqué par l'amplitude du mouvement nécessaire à la réalisation du signe. De plus, lors de ces mouvements larges, lorsque les mains réapparaissent dans le cadre, leur zone de réapparition ne coïncide pas toujours à leur zone de sortie.

Le figure 4.4 illustre un cas de sortie de cadre lors de la réalisation d'un signe ainsi que le moment de réapparition des mains.

FIGURE 4.4 – Exemple de sortie des mains avec le moment de sortie (à gauche) et le moment d'entrée des mains dans le cadre (à droite) (images originales provenant de [21]).

4.1.3 Angle de vue

Les caméras ayant filmé les signeurs durant leurs discussions n'étaient pas positionnées en face de ceux-ci mais légèrement de biais. De plus, l'angle de vue peut légèrement varier selon la manière dont le signeur s'assoit et s'oriente.

D'un point de vue de la reconnaissance automatique, cela signifie que le nombre de formes de mains pouvant être perçues augmente fortement. De plus, certains signes visuellement proches peuvent être confondus. Par exemple, les signes "tu" et "il" sont très similaires. Ils consistent tous les deux en un signe de pointage de l'index. Dans le cas du signe "tu", l'index est pointé en direction de l'interlocuteur. Dans le cas du "il", l'index pointe vers le coté. Comme l'illustre la figure 4.5 la distinction entre les deux est aisée lorsque les signes sont perçus de face avec un angle de vue idéal. Lors d'une reconnaissance manuelle, l'angle de vue est pris en compte naturellement et ne prête pas à confusion. En revanche, lors d'une reconnaissance automatisée et lorsque l'angle de vue est légèrement en biais, les deux signes peuvent être facilement confondus. La distinction entre un pointage vers l'interlocuteur et un pointage "de coté" est difficile à distinguer.

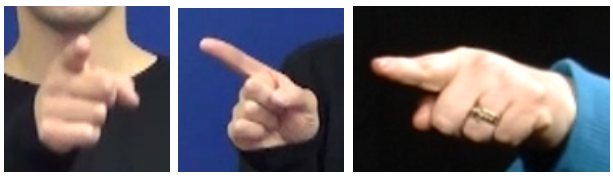


FIGURE 4.5 – Exemple d'ambiguïté entre signes du à l'angle de vue. De gauche à droite : signe "tu" vu de face ; signe "il" vu de face ; signe "tu" vu de biais (images originales provenant de [21]).

4.1.4 Présence du second signeur dans le champ

En raison du placement des interlocuteurs et du cadrage des vidéos, le second signeur apparaît parfois dans le champ de vision. Pour un être humain, la distinction entre les

mains du signeur filmé et celles du second signeur est aisée. Cependant, pour un algorithme de reconnaissance, cette distinction peut être très compliquée à réaliser. Cela est d'autant plus compliqué lorsque les signeurs se touchent ou que le second passe devant le signeur filmé. Lorsque cela arrive, les mains des deux signeurs se confondent visuellement. La figure 4.6 illustre ce problème d'apparition du second signeur.

FIGURE 4.6 – Exemple de présence du second signeur dans le champ (image originale provenant de [21]).

4.1.5 Éclairage non uniforme

Durant le tournage des séquences, les signeurs ont été éclairés à l'aide de spots. Malheureusement, l'éclairage produit par ces derniers n'était pas uniforme. En raison de cet éclairage non uniforme, certaines zones de peau se retrouvent plus éclairées que d'autres. Ces différences d'éclairage se traduisent par des zones très sombres et d'autres zones très lumineuses. Visuellement, ces zones lumineuses apparaissent comme des tâches blanches sur la peau (voir figure 4.7). Les zones d'ombres quant à elles sont vues comme des tâches presque noires.

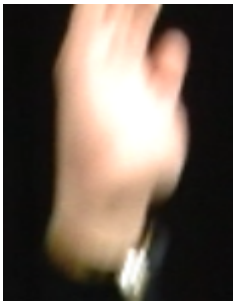


FIGURE 4.7 – Impact de l'éclairage sur la couleur de la peau (image originale provenant de [21]).

4.1.6 Absence de contrainte sur l'habillage

Dans le domaine de la reconnaissance gestuelle, il est courant que certaines contraintes soient imposées sur l'habillage. Par exemple, l'obligation de porter des vêtements sombres

ou des gants colorés. Cependant, le but premier du LSFB-lab n'étant pas d'automatiser l'annotation, aucune contrainte n'a été imposée aux signeurs quant à leur habillage.

En raison de cette absence de contrainte, certains signeurs portent des vêtements non appropriés pour le traitement automatique des vidéos. Parmi ceux-ci, deux grandes catégories peuvent être distinguées :

— les vêtements laissant paraître de nombreuses zones de peau comme les manches courtes et les décolletés ;
— les vêtements ayant une teinte proche de la teinte de la peau.

La figure 4.8 illustre ces deux catégories.

FIGURE 4.8 – Exemples de problèmes dus à l'absence de contrainte sur l'habillage (images originales provenant de [21]).

4.2 Observations générales

Des observations plus générales ont pu être faites en discutant avec des personnes du LSFB-lab et en se renseignant sur la langue des signes. Par exemple, un mot peut être signé de plusieurs manières différentes. Cela est souvent du au fait que, en raison du traité de Milan, des patois se sont formés dans différentes régions et la langue des signes a évoluée différemment par endroit. Cela peut aussi venir du fait que la langue des signes est très imagée. Un même mot peut souvent avoir plusieurs "images" pour le décrire.

Ces différentes manières de signer un même mot sont en général reflétées dans le corpus grâce à des tags distincts. Par exemple, le corpus possède les tags "MAMAN", "MAMAN.FEMME", "MAMAN.HANCHE" et "MAMAN.MONSIEUR" qui sont tous traduits par "maman" mais signés de manière différente. Cependant, même de la sorte, il arrive que pour un même tag, les manières de signer soient légèrement différentes.

Pour la reconnaissance gestuelle, cette possibilité de signer de manière différente un même signe a un impact important. En effet, il ne sera pas possible de définir pour chaque signe un et un seul modèle auquel il suffirait de comparer le signe observé. De plus, cela veut

également dire que ce n'est pas parce que le signe observé ne ressemble à aucun modèle connu d'un signe donné qu'il ne correspond pas à celui-ci : il pourrait s'agir d'une nouvelle manière de signer. Par la suite, nous appellerons ces différentes manière de signer un même signe les *signatures* du signe.

4.3 Exigences

Pour s'assurer que l'annotation (semi-)automatique des vidéos est possible, la solution produite devra être en mesure de répondre aux exigences suivantes :

- Accélérer le processus d'annotation ;
- Reconnaitre les signes réalisés dans une vidéo ;
- Produire un fichier d'annotations compatible avec ELAN ;
- Dans le cas d'une solution semi-automatique, pouvoir être utilisée par une personne non bilingue.

Le but premier du programme sera donc d'accélérer le processus d'annotation afin de soulager le travail des annotateurs. Ayant décidé d'explorer la piste de l'automatisation (ou semi-automatisation), le logiciel devra bien entendu être en mesure de reconnaitre et de différencier les signes présents dans chaque vidéo. Cette reconnaissance sera évidemment réalisée en tenant compte des difficultés énumérées ci-dessus. Une reconnaissance totale n'est pas nécessaire pour atteindre cette exigence. En effet, l'important est qu'une large majorité des signes soit correctement reconnue afin que le travail d'annotation manuel soit réduit de manière conséquente. Un taux de reconnaissance moyen d'au moins 70% semble être un objectif raisonnable. Cette exigence de reconnaissance constituera le point centrale pour évaluer la solution mise en place.

Outre la reconnaissance, la solution devra être en mesure de produire un fichier d'annotations compatible avec ELAN. Le fichier résultant du processus de reconnaissance devra donc respecter la structure des fichiers .eaf. Ceci est important pour permettre aux annotateurs de vérifier et/ou compléter le résultat produit.

Enfin, dans le cas où la solution ne serait pas totalement automatique, celle-ci ne devrait pas nécessiter une connaissance particulière de la langue des signes. De cette façon, les interactions nécessaires avec le logiciel pourront, éventuellement, être déléguées à des personnes non bilingues. Cela permettant alors aux annotateurs bilingues de se concentrer sur la validation des résultats.

Chapitre 5

Solution proposée

La solution proposée met en place le processus illustré par la figure 5.1. Ce processus fonctionne en deux phases : la phase d'apprentissage et la phase de reconnaissance.

Durant la phase d'apprentissage, les méthodes de reconnaissance gestuelle sont appliquées à des vidéos annotées afin d'extraire les caractéristiques des signes présents dans celles-ci. Grâce à ces caractéristiques, une base de connaissances est construite en associant à chaque tag du fichier d'annotations un ou plusieurs ensembles de caractéristiques. Chaque ensemble représente alors signature particulière du signe.

Durant la phase de reconnaissance, un ensemble de caractéristiques est extrait pour chaque occurrence de signe présent dans une vidéo devant être annotée. Ensuite, pour reconnaitre les signes effectivement réalisés, ces ensembles de caractéristiques sont comparés avec ceux présents dans la base de connaissances. En appliquant des méthodes de machine learning, un tag est alors associé à chaque signature observée.

Il est a noter que dans le cadre de notre stage, la phase de reconnaissance est réalisée sur base de vidéos pour lesquelles les signes ont déjà été segmentés (c'est-à-dire que le début et le fin de chaque signe ont été déterminés). Nous avons fait ce choix car nous n'avions pas la prétention de réaliser un système de reconnaissance complet en seulement quatre mois. Nous avons donc choisi d'élaguer le projet afin de pouvoir proposer un système fonctionnel qu'il suffirait d'améliorer par la suite. En effet, la segmentation automatique n'est pas une tâche aisée et le principal objectif de notre mémoire était de déterminer la faisabilité de la reconnaissance automatique des signes présents dans les vidéos. C'est pourquoi nous avons décidé de nous concentrer sur la reconnaissance en laissant la segmentation comme amélioration possible si les tests s'avéraient concluants.

5.1 Extraction des caractéristiques

La première étape importante pour reconnaitre des gestes est d'en extraire certaines caractéristiques. Pour savoir quelles caractéristiques observer, nous avons utilisé les caractéristiques qui permettent de différencier les signes de la langue des signes (voir section 1.2). Pour rappel, ces caractéristiques sont au nombre de cinq :

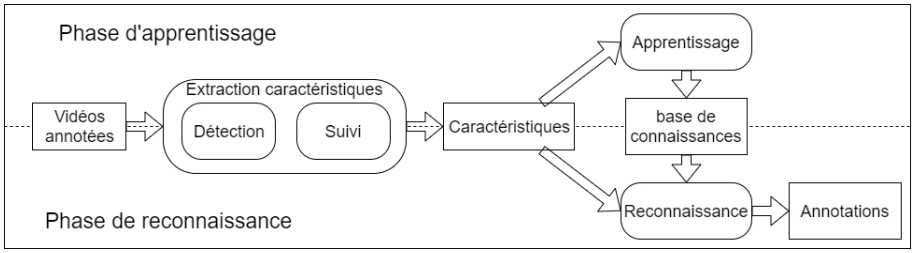


FIGURE 5.1 – Schéma du fonctionnement général de la solution proposée.

— la configuration de la main ;
— l'emplacement de la main ;
— l'orientation de la main ;
— le mouvement de la main ;
— l'expression du visage.

En raison des limitations de temps auxquelles nous devions faire face, un choix à du être opéré parmi les caractéristiques à prendre en compte.

Parmi ces caractéristiques, la configuration de la main et son emplacement sont les caractéristiques ayant été jugées les plus importantes. Il est toutefois nécessaire de préciser que dans notre cas, nous n'utiliserons pas la configuration de la main comme entendue au point 1.2.2 mais plutôt sa forme générale.

Une troisième et dernière caractéristique considérée par l'algorithme présenté dans ce mémoire est le mouvement effectué par la main. Cette caractéristique a été jugée assez importante car elle permet de faire la différence entre certains signes pour lesquels la configuration et la position des mains sont identiques.

L'orientation de la main et l'expression du visage ont volontairement été laissées de côté. Concernant l'orientation de la main, son analyse semblait trop compliquée à mettre en place dans le temps imparti. De plus, celle-ci semblait moins utile que les trois caractéristiques précédentes.

En ce qui concerne l'expression du visage, de nombreux logiciels existants permettent de l'analyser. L'ajout de cette caractéristique serait donc aisé.

Plus d'explications sur les pistes pouvant être suivies ultérieurement se trouvent à la section 7.5.

Afin d'extraire les caractéristiques souhaitées, il est nécessaire de savoir localiser les mains et la tête à tout moment de la vidéo. Pour cela, il faut, dans un premier temps, reconnaitre les pixels de couleur peau. Ensuite, il faut identifier les groupes de pixels de couleur peau. Pour finir, les zones correspondantes aux mains et à la tête doivent être déterminées.

5.1.1 Reconnaitre un pixel de couleur peau

La détection des zones de peau est réalisée sur base de la méthode des histogrammes présentée au point 2.2.1. Pour rappel, la méthode classifie les pixels en pixels de "peau" ou de "non-peau" sur base d'histogrammes représentant la répartition des couleurs au sein de ces classes.

La méthode a toutefois été légèrement adaptée afin d'obtenir le meilleur compromis entre pourcentage de détection correct et qualité des formes détectées. En effet, bien que la méthode d'origine permette de détecter un grand nombre de pixels correctement, les pixels de peau non détectés provoquaient régulièrement une division de la forme dans la main. Pour permettre une reconnaissance efficace des signes nous avons jugé nécessaire de sacrifier un peu de qualité dans la détection afin d'obtenir plus fréquemment des formes "complètes".

Concernant l'espace de couleur, la détection utilise l'espace YCbCr. Cet espace a été sélectionné suite à la comparaison des performances des espaces RGB, HSV et YCbCr lors de la détection de peau sur un petit ensemble d'images. Les résultats de ces détections montraient des taux de vrais positifs [1] relativement équivalents pour les trois espaces (aux alentours de 95%). Toutefois, là où les espaces RGB et HSV possédaient un taux de faux positifs [2] de l'ordre de 3% et 4% respectivement, le YCbCr possédait lui un taux inférieur à 2%. En outre, la qualité des formes était similaire pour chacun des espaces. Le YCbCr a donc été jugé plus efficace.

Comme la méthode présentée au point 2.2.1, deux étapes sont nécessaire à la réalisation de la détection : la construction des histogrammes et la classification des pixels de l'image.

Construction des histogrammes

La première étape de la détection, réalisée une seule fois, est de construire les histogrammes représentant les classes de pixels de couleur de peau et de "non peau". Pour cela, des images d'apprentissage, pour lesquelles chaque pixel a été manuellement classifié, sont chargées. Ensuite, pour chaque pixel de ces images, l'histogramme de la classe correspondante de ce pixel est mis à jour. Par exemple, si le pixel à traiter est de la classe peau, l'histogramme de la classe peau sera mis à jour.

Le procédé décrit ci-dessus présente la construction des histogrammes utilisée par la méthode traditionnelle. Toutefois, comme spécifié précédemment, nous avons légèrement modifié cette méthode. Ainsi, au lieu de ne construire qu'un histogramme par classe, l'algorithme de détection va concevoir deux histogrammes par classe. Le premier est construit en tenant compte des trois composantes de l'espace YCbCr. Le second en revanche ne considère que les composantes Cb et Cr. Bien que cette méthode puisse paraître étrange, il s'agit de celle qui, d'après les tests réalisés durant le stage, permet de mieux gérer les

1. Un vrai positif est un pixel de peau correctement détecté.
2. Un faux positif est un pixel de "non peau" détecté comme un pixel de peau.

problèmes de détection du à l'éclairage pour obtenir le meilleur compromis "qualité de détection - qualité de forme" tel qu'expliqué précédemment. Cela peut s'expliquer par le fait que les deux méthodes entraînent chacune une distribution des couleurs de peau différente et donc, une classification des pixels différente. Il s'est avéré qu'en combinant les résultats des deux classifications, il était possible d'obtenir des formes de mains plus pertinentes, au prix toutefois d'un peu plus de faux positifs.

Classification de pixel

Deux types d'histogrammes par classe sont donc utilisés : un histogramme YCbCr et un histogramme CbCr. Pour classifier un pixel, la première étape est d'appliquer, pour chaque type d'histogramme, la procédure suivante :

1. Chaque couleur c se voit attribuer une probabilité $P(c|peau)$ et $P(c|\neg peau)$. Ces probabilités sont calculées en utilisant les formules 2.1 et 2.2 et en considérant comme coefficients de normalisation le nombre de pixels total des histogrammes.
2. Le ratio $\frac{P(c|peau)}{P(c|\neg peau)}$ de chaque couleur c est calculé. Les couleurs dont ce ratio est supérieur à un certain seuil, défini empiriquement, sont alors considérées comme des couleurs de peau.
3. Chaque pixel de l'image est classifié comme un pixel de peau si sa couleur est une couleur de peau.

Généralement, le résultat de ce genre de procédure est une image pour laquelle chaque pixel a été classifié. On attribue la couleur "blanc" aux pixels répondant aux critères recherchés (c'est-à-dire les pixels de peau dans ce cas-ci) et la couleur "noir" aux autres. L'image finale est donc une image noir et blanc (ou binaire) dans laquelle tout pixel blanc est un pixel de peau dans l'image originale. La figure 5.2 illustre ce concept.

FIGURE 5.2 – Illustration du concept d'image binaire. A gauche, l'image originale (image originale provenant de [21]). A droite, l'image binaire correspondante.

Une fois la procédure précédente appliquée pour les deux types d'histogrammes, deux images binaires ont été produites : i_{YCbCr} et i_{CbCr}. Pour obtenir la classification finale (et donc l'image binaire finale), une opération "OR" est effectuée sur ces deux images. Ainsi, un pixel sera finalement classifié comme pixel de peau s'il est considéré comme tel dans i_{YCbCr} ou dans i_{CbCr}.

Encore une fois, la méthode n'est pas parfaite (notamment car les défauts de détection des deux types d'histogrammes sont conservés). Cependant, il s'agit de la méthode ayant fourni les meilleurs résultats lors de nos tests. Ces tests ont été effectués en réalisant la détection de peau avec différentes méthodes sur une dizaine d'images. Les méthodes ainsi testées comprenaient la détection à l'aide de régions explicitement définies, d'un seul type d'histogrammes et, bien sûr, de deux types d'histogrammes. En terme de vrais positifs, les régions explicitement définies fournissaient un meilleur résultat que les deux autres avec un taux de 99%. Cependant, la méthode a été écartée car elle engendrait un trop grand nombre de faux positifs (presque 5%) entraînant trop de déformation de la forme des mains. Les méthodes à un et deux types d'histogrammes possédaient des résultats relativement identiques en terme de taux de vrais positifs ($\pm$95%) et de faux positifs ($\pm$1,5%). Le choix entre ces deux méthodes a donc été réalisé sur base de la qualité des formes des mains obtenues. De ce point de vue là, l'utilisation de deux types d'histogrammes semblait fournir des résultats de meilleure qualité.

Le schéma 5.3 illustre la méthode de classification.

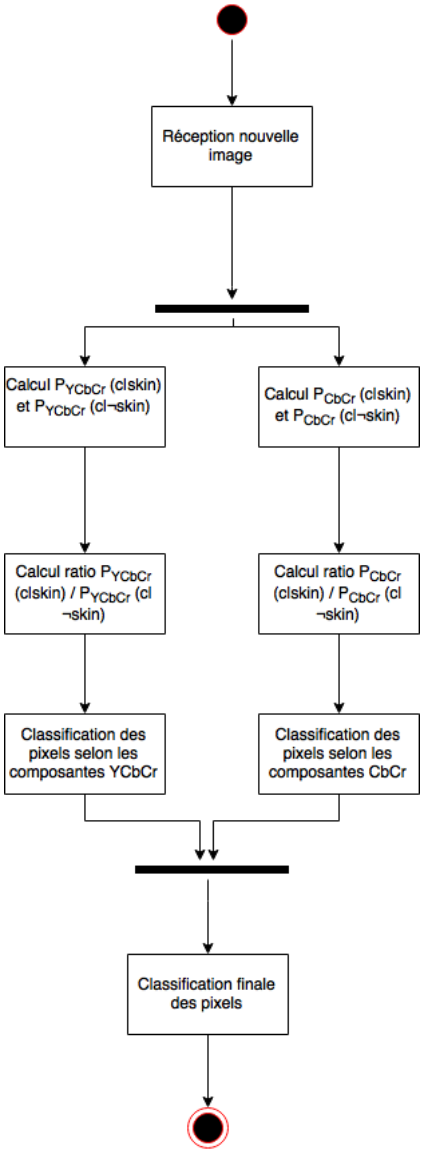


FIGURE 5.3 – Processus de classification des pixels.

5.1.2 Identifier les groupes de pixels de couleur peau

Une fois l'image binaire obtenue, il est nécessaire d'identifier les différents blobs [3] présents dans l'image pour permettre la réalisation de la phase de suivi.

Pour identifier les blobs, chaque pixel de l'image est parcouru. Lorsqu'un pixel blanc (c'est-à-dire classifié comme pixel de peau) est rencontré, une vérification des pixels adjacents est réalisée. Si aucun pixel adjacent n'est blanc, cela signifie que ce pixel constitue un nouveau blob à lui seul. Si certains pixels adjacents sont également blancs, cela signifie

3. Un blob représente un groupe de pixels connectés dans une image binaire. L'image binaire représentant, dans notre cas, les pixels de couleur peau, il s'agit d'un groupe de pixels connectés de couleur peau.

qu'ils appartiennent tous à un même blob.

Notons que, en principe, seuls trois blobs devraient être découverts (un pour la tête et un pour chaque main). Cependant, il arrive souvent d'en trouver plus. Cela est du notamment à des faux positifs, c'est-à-dire au fait que des pixels qui ne sont pas de la peau sont vus comme tels.

5.1.3 Suivi

Pour identifier les parties du corps, une méthode naïve est de garder les trois plus gros blobs de l'image et de supposer que la tête se trouve en haut, la main droite se trouve à droite et la main gauche à gauche. Cette méthode montre très vite des handicaps. Notamment, parce que, parfois, la main gauche se retrouve à droite ou parce que les mains se retrouvent au dessus de la tête. De plus, bien souvent les parties du corps "fusionnent", c'est-à-dire qu'une d'entre elles passe devant une autre et qu'elles ne forment alors qu'un seul blob. Pour ces raisons, une méthode plus robuste a du être trouvée.

Pour réaliser le suivi des parties du corps, nous nous sommes inspirés de la méthode des filtres Kalman (cf. point 2.2.2), c'est-à-dire que le suivi est réalisé sur base des données observées combinées aux informations connues de l'image précédente. Le principe de base de l'algorithme utilisé ici est simple. En raison du nombre d'images par seconde (fps) élevé dans les vidéos filmées par le laboratoire LSFB, à savoir 50 fps, il est raisonnable de supposer que les blobs qui correspondent à une partie du corps présents dans l'image t_i possèdent des pixels en commun avec leur équivalent dans l'image t_{i-1}. Ainsi, les blobs représentant les mains et la tête dans l'image t_i sont déterminés sur base des blobs présents dans l'image t_{i-1}.

L'algorithme se basant toujours sur l'image précédente, il faut réfléchir au cas de base et aux cas particuliers. Le cas de base est celui de la première image (où $i = 0$). Dans ce cas il est impossible de se baser sur l'image précédente. Il est donc nécessaire de connaître l'emplacement des mains et de la tête dans l'image t_0 pour pouvoir effectuer le suivi. La technique la plus simple est de demander à l'utilisateur d'identifier lui-même les mains et la tête dans l'image t_0. En ce qui concerne les cas particuliers, nous n'en avons rencontré qu'un seul dans les vidéos fournies par le LSFB-lab ; celui des coupures. En effet, lorsqu'il y a une coupure dans la vidéo, on ne peut pas se baser sur l'image précédente car, en général, les mains ne se trouvent plus du tout au même endroit. Dans ce cas, tout comme pour la première image, il faut demander à l'utilisateur l'emplacement de la tête et des mains.

Dans le cas où $i > 0$, l'algorithme procède comme suit. Pour chaque blob représentant une partie du corps (tête ou main) dans l'image t_{i-1}, un équivalent parmi les blobs de l'image t_i est recherché. En fonction de l'état du blob dans l'image t_{i-1}, cette recherche sera légèrement différente. Les blobs peuvent avoir trois états différents : hors cadre, en bordure de l'image ou dans l'image (c'est-à-dire ni hors cadre, ni en bordure).

Blobs hors cadre

Dans le cas d'un blob qui était hors-cadre dans l'image t_{i-1}, l'algorithme vérifie si le blob est revenu dans l'image, c'est-à-dire s'il existe un blob situé en bordure de l'image t_i dans la zone où le blob de l'image t_{i-1} avait disparu.

- Si aucun blob n'est trouvé, cela signifie que le blob est toujours hors cadre. La position de ce blob dans l'image t_i est alors considérée comme étant la même que la position du blob dans l'image t_{i-1}, c'est-à-dire l'endroit où le blob était sorti de l'image qui a été mémorisée comme étant la zone de recherche future.
- Si un ou plusieurs blobs sont trouvés, seul le blob le plus proche de l'endroit où le dernier blob était sorti de l'image est retenu.

Blobs en bordure

Si le blob était en bordure dans l'image t_{i-1}, seuls les blobs de l'image t_i ayant au moins un pixel en commun avec le blob de l'image t_{i-1} sont considérés. Parmi ces blobs, ceux étant trop petits (pour représenter une main ou la tête) sont retirés des correspondants potentiels sauf s'ils sont en bordure. Cette dernière distinction est importante car dans le cas d'une partie du corps sortant de l'image, il est possible que seule une petite partie de celle-ci soit visible.

- Si aucun blob n'a été trouvé, cela signifie que la partie du corps concernée est sortie de l'image. Il n'existe alors aucun blob correspondant à celle-ci dans l'image. Une zone correspondant aux environs de l'endroit où le blob a disparu est alors définie comme zone de recherche future.
- Si un seul blob a été trouvé, celui-ci est considéré comme l'équivalent du blob de l'image t_{i-1} dans l'image t_i.
- Si plusieurs blobs ont été trouvés, nous allons regarder si le blob précédent était un blob fusionné. S'il n'était pas fusionné, on prendra le blob avec le plus de pixels en commun. S'il était fusionné, on le "défusionnera".

Blobs dans l'image

De la même manière que pour les blobs situés en bordure de l'image, seuls les blobs de l'image t_i qui ont au moins un pixel en commun avec le blob de l'image t_{i-1} sont considérés. Les blobs trop petits sont également retirés des correspondants potentiels sauf dans le cas où ils touchent le bord pour les mêmes raisons que précédemment.

- Si un seul blob a été trouvé, celui-ci est considéré comme l'équivalent du blob de l'image t_{i-1} dans l'image t_i.
- Si plusieurs blobs peuvent correspondre, nous allons regarder si le blob précédent était un blob fusionné. Si c'est le cas, on le défusionnera. S'il n'était pas fusionné, on prendra le blob avec le plus de pixels en commun.
- Si aucun blob n'est trouvé, c'est peut être parce que le blob de l'image t_{i-1} a bougé trop vite. Dans ce cas, il faudra élargir légèrement les recherches et chercher dans l'image t_i s'il n'y a pas de blob aux alentours.
  - Si un ou plusieurs blobs sont trouvés, celui le plus proche du blob de l'image t_{i-1} sera considéré.

— Si malgré cela aucun blob potentiel n'a été trouvé, le blob est considéré comme étant caché par un autre objet (comme par exemple une manche). Dans ce cas, un blob avec une zone de recherche par défaut sera considéré pour la suite.

Fusionner des blobs

Lorsque la méthode de suivi donne les blobs de l'image t_i correspondant aux blobs de l'image t_{i-1}, il faut vérifier que certains d'entre eux n'ont pas fusionné. En effet, il se peut qu'une partie du corps soit passée devant une autre provoquant la "fusion" des blobs les représentant. Pour vérifier si des blobs ont fusionné, il faut regarder si plusieurs blobs de l'image t_{i-1} n'ont pas trouvé le même équivalent dans l'image t_i. Si c'est le cas, cela signifie que le blob de l'image t_i représente la fusion des parties de corps des blobs correspondants de l'image t_{i-1}.

Prenons un exemple (figure 5.4) : dans l'image t_{10}, le blob $b0_{10}$ représentait la tête, le blob $b2_{10}$ représentait la main droite et le blob $b3_{10}$ représentait la main gauche. Dans l'image t_{11}, les équivalents suivants ont été trouvé : $b0_{10}$ correspond à $b1_{11}$, $b2_{10}$ correspond à $b2_{11}$ et $b3_{10}$ correspond à $b1_{11}$. Il est, alors, possible de conclure que $b1_{11}$ représente la tête ($b0_{10}$) et la main gauche ($b3_{10}$) et que $b2_{11}$ représente la main droite ($b2_{10}$).

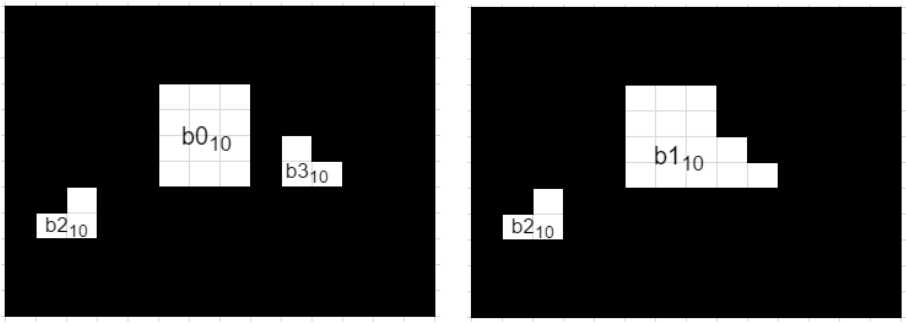


FIGURE 5.4 – Exemple de fusion de blobs. À gauche, l'image t_{10} et à droite, l'image t_{11}.

Dé-fusionner des blobs

Lorsque des blobs doivent être dé-fusionnés, beaucoup de suppositions sont faites. Plus précisément, l'hypothèse est faite que la tête reste plus ou moins au même endroit et que la main droite et la main gauche se trouvent respectivement à sa droite et à sa gauche. Cependant, ces suppositions sont quelque peu naïves et certains cas sont incertains. Lorsqu'un de ces cas incertains se présente, une aide est demandée à l'utilisateur.

Voici les différents cas et la façon dont ils sont résolus.

— Si le blob de l'image t_{i-1} représentait les deux mains : dans l'image t_i, le blob de gauche est la main gauche et celui de droite, la main droite. Cependant, si aucun blob n'est significativement plus d'un côté que l'autre, le cas est incertain.

— Si le blob de l'image t_{i-1} représentait la tête et la main gauche : dans l'image t_i, le blob se trouvant là où se trouvait la tête est la tête et l'autre est la main gauche. Si aucun blob ne se trouve là où se trouvait la tête, le cas est incertain.

— Si le blob de l'image t_{i-1} représentait la tête et la main droite : dans l'image t_i, le blob se trouvant là où se trouvait la tête est la tête et l'autre est la main droite. Si aucun blob ne se trouve là où se trouvait la tête, le cas est incertain.
— Si le blob de l'image t_{i-1} représentait la tête et les deux mains : dans l'image t_i, il faut regarder si le blob se divise en deux ou en trois.
  — Si le blob se divise en deux : on pourrait supposer que le plus gros des deux blobs est le blob qui est encore fusionné. Cependant, ce n'est pas souvent le cas. Ce cas ci est donc incertain.
  — Si le blob se divise en trois : le blob se trouvant là où se trouvait la tête est la tête. Les deux autres blobs sont déterminés comme le cas de dé-fusion des deux mains. Si aucun blob ne se trouve là où se trouvait la tête, le cas est incertain.

5.2 Comparaison des caractéristiques

Avant de pouvoir réaliser les phases d'apprentissage et de reconnaissance, une méthode de comparaison pour chacune des caractéristiques extraites doit être définie afin d'être en mesure de comparer deux signes. Pour rappel, les caractéristiques utilisées sont les suivantes :

— la forme de la main ;
— l'emplacement de la main (par rapport au visage) ;
— le mouvement de la main.

5.2.1 Comparer la forme de la main

La méthode de comparaison de forme utilisée part du principe qu'une forme géométrique peut être entièrement définie grâce à ses angles et à leur position. Ainsi, l'idée de base est de comparer les angles respectifs des deux formes. La méthode fonctionne en trois étapes retournant chacune un certain pourcentage de similitude, respectivement s_1,s_2 et s_3.

D'abord, une comparaison du nombre d'angles est réalisée. Le pourcentage de similitude s_1 est alors déterminé sur base de la différence du nombre d'angles des deux formes. Pour tenir compte du bruit éventuel dans les observations, une certaine marge d'erreur est tolérée.

Ensuite, un couplage est réalisé entre les angles des deux formes. L'objectif est d'associer à chaque angle de la première forme au maximum un angle de la seconde. Cette étape est très importante car, sans cela, une comparaison des angles n'est pas possible. En effet, rien ne garantit que l'angle i de la première forme corresponde à l'angle i de la seconde. L'une des deux formes peut, par exemple, disposer d'un angle de plus que l'autre à cause de bruit dans l'observation. L'algorithme chargé de ce couplage fonctionne de la la façon expliquée ci-après. Soit deux formes F et F' et leurs ensembles d'angles respectifs A et A' avec, $|A| \leq |A'|$. Une translation des angles de A' est tout d'abord réalisée afin de s'assurer que les deux formes soient alignées, c'est-à-dire que leurs centres soient situés aux mêmes coordonnées. Ensuite, pour tout angle $\alpha \in A$, une vérification est effectuée afin de déterminer s'il existe déjà un couple (α, β) avec $\beta \in A'$. Si un tel couple existe, le

traitement de l'ange α s'arrête et l'algorithme passe à l'angle suivant. Sinon, l'algorithme tente de former le couple (α, β) tel que :

- $d(\alpha, \beta) < x$ où $d(\alpha, \beta)$ représente la distance entre les angles α et β et x représente une distance maximale autorisée. Cette condition permet de limiter le couplage de l'angle α avec des angles de A' situés dans un certains rayon.
- il n'existe pas de couple (α', β) tel que $d(\alpha, \beta) > d(\alpha', \beta)\ \forall \alpha' \in A$.
- $d(\alpha, \beta) < d(\alpha, \beta')\ \forall \beta' \in A'$

Le processus de création des couples est répété jusqu'à arriver à un point fixe, c'est-à-dire au moment où la liste de couples de l'itération i est la même que celle de l'itération $i-1$. Après cela, le pourcentage de similitude s_2 est calculé sur base du nombre maximum de couples possibles et du nombre de couples effectivement formés.

Finalement, le pourcentage de similitude s_3 est calculé grâce aux couples réalisés à l'étape précédente. Pour chaque couple (α, β) avec $\alpha \in A$ et $\beta \in A'$, un pourcentage de similitude entre α et β, noté p_i, est calculé sur base de la différence de leur amplitude. Le pourcentage de similitude s_3 correspond alors à la moyenne des pourcentages p_i.

Une fois ces trois étapes réalisées, les pourcentages s_1, s_2 et s_3 sont combinés au moyen d'une multiplication afin d'obtenir le pourcentage de similitude total entre les deux formes.

5.2.2 Comparer la position de la main

La position de la main relative à la tête est déterminée en fonction de l'emplacement de cette dernière et du centre du blob représentant la main. Comme l'illustre la figure 5.5, l'image est divisée en six zones de tailles variables par une droite horizontale correspondant à l'extrémité basse de la tête et par deux droites verticales correspondants aux extrémités droite et gauche de la tête. La position de la main correspond ainsi à la zone dans laquelle celle-ci se trouve.

Pour réaliser la comparaison de deux positions relatives, il est nécessaire de prendre en compte que l'angle de vue des vidéos utilisées n'est pas toujours le même. En fonction de l'orientation de la personne filmée, la position relative de la main peut-être perçue différemment bien que, physiquement, la main soit au même endroit. Cependant, ce changement de perception n'impacte que la composante horizontal de la position. Pour cette raison, deux positions à la même hauteur doivent être comparées en considérant un certain degré de liberté.

La comparaison de deux positions relatives est réalisée en comparant les zones respectives de celles-ci. Si les deux positions ne sont pas à la même hauteur (c'est-à-dire une dans la partie supérieure et l'autre dans la partie inférieure de l'image), les deux positions sont considérées comme différentes. Si, par contre, les deux positions sont à la même hauteur, un "score" de correspondance entre les deux positions est calculé en fonction du nombre de zones d'écart entre celles-ci. Plus le nombre de zones d'écart est grand, plus le score sera petit.

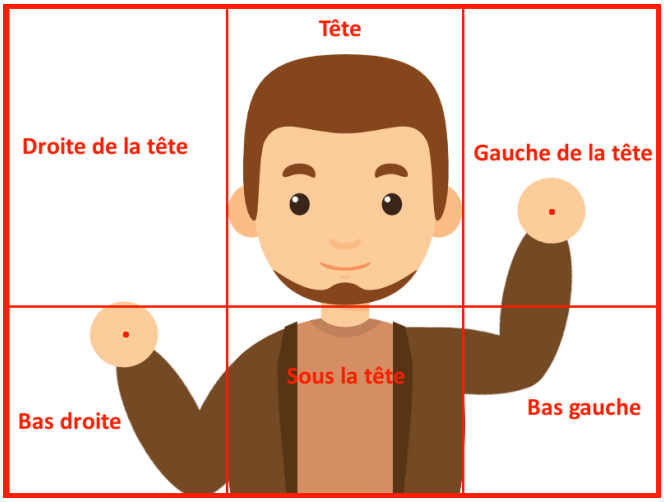


FIGURE 5.5 – Positions relatives possibles des mains

5.2.3 Comparer le mouvement de la main

Il faut pouvoir comparer deux mouvements, ou plutôt deux trajectoires définies par le mouvement d'une main, indépendamment de la vitesse de ces mouvements et de leur position de départ dans l'espace. Nous avons donc choisi d'utiliser la méthode de déformation temporelle dynamique ou Dynamic Time Warping en anglais (DTW). Pour rappel, cette méthode présentée au point 2.2.3 mesure la ressemblance entre deux séquences temporelles qui peuvent varier en vitesse. En général, le DTW calcule la correspondance optimale entre deux séquences qui, si elles sont déformées par cette correspondance seront alignées en temps. Outre ce matching, le DTW fournit une mesure de similarité. C'est cette mesure qui sera utilisée pour comparer des mouvements. Plus cette mesure est basse, plus les mouvements se ressemblent.

5.3 Apprentissage

La phase d'apprentissage est réalisée sur base de vidéos pour lesquelles les annotations ont déjà été produites par les annotateurs. Pour chacune de ces vidéos, les ensembles des caractéristiques des signes réalisés sont extraits. Ensuite, une base de connaissances des signes est construite à l'aide de ces ensembles et des tags contenus dans les fichiers d'annotations. Cette base de connaissances est constituée de la liste des tags rencontrés dans les fichiers d'annotations. À chacun de ces tags est associé une ou plusieurs signatures représentant chacune une façon particulière de réaliser le signe correspondant. Enfin, un nombre d'occurrences est associé à chaque signature afin de mémoriser le nombre d'apparitions de celle-ci pour le tag auquel elle est associée. Le but est de pouvoir réaliser par la suite une reconnaissance plus efficace ou d'effectuer des statistiques à des fins de recherche.

Pour construire la base de connaissances, le processus suivant, illustré par la figure 5.6, est appliqué à chaque signe réalisé dans les vidéos. D'abord, une vérification a lieu afin de déterminer si le tag associé au signe est déjà présent dans la base de connaissances. Ensuite, un traitement différent sera effectué en fonction de la présence ou non du tag.

- Si le tag n'est pas encore présent, une nouvelle entrée est créée. La signature observée est ensuite associée à cette nouvelle entrée. Enfin, le nombre d'occurrences de cette signature est initialisé à un.
- Si le tag est déjà présent dans la base de connaissances, une comparaison est réalisée entre les signatures déjà associées à ce tag et la signature observée.
  - Si la signature observée est identique à une de celles déjà associées au tag, c'est-à-dire que toutes les caractéristiques de celles-ci sont identiques (cf. 5.2), le nombre d'occurrences de cette signature est incrémenté de un.
  - Dans le cas où aucune signature déjà associée au tag n'est identique à celle observée, une association entre cete dernière et le tag est créée. Le nombre d'occurrences de cette nouvelle signature est ensuite initialisé à un.

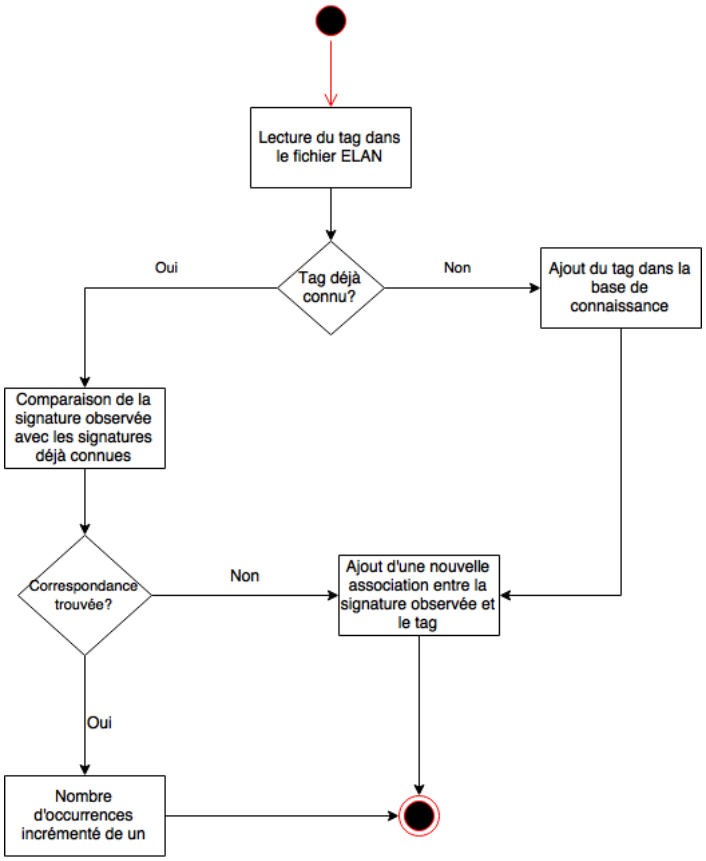


FIGURE 5.6 – Processus d'ajout d'une observation dans la base de connaissances.

5.4 Reconnaissance

Le système de reconnaissance fonctionne sur base de la méthode des arbres de décisions. Ainsi, la reconnaissance d'un signe débute avec l'ensemble des signatures de la base de connaissances. Cet ensemble est ensuite réduit successivement en comparant les caractéristiques de l'observation avec celles des signatures de la base de connaissances. L'ensemble ainsi réduit constitue alors l'ensemble des signatures potentielles pouvant correspondre au signe observé.

Grâce à cet ensemble de signatures potentielles, le signe ayant le plus de chance de correspondre à l'observation est déterminé. Pour cela, deux méthodes différentes ont été mise en place afin de déterminer celle fournissant les meilleurs résultats.

La première méthode attribue à chaque signature une probabilité de correspondance. Cette probabilité est pondérée en fonction de son nombre d'occurrences dans la base de connaissances et du fait que le signe soit réalisé à deux mains plutôt qu'une. Ce dernier facteur est important à considérer car, lorsqu'il s'agit de signes à deux mains, il existe un plus grand risque d'imprécision dans la détection des mouvements et de la forme que s'il s'agissait d'un signe à une main (puisque dans le premier cas, les imprécisions peuvent venir des deux mains). Pour ce qui est du nombre d'occurrences, il est normal que cela impacte la reconnaissance. Un signe déjà rencontré cent fois a plus de chance d'être de nouveau observé qu'un signe rencontré cinq fois.

La seconde méthode permet quant à elle d'attribuer une probabilité de correspondance $P(S)$ au signe dans sa globalité. Par exemple, si le signe S dispose de deux signatures parmi l'ensemble des signatures potentielles, $P(S)$ sera calculée sur base de ces deux signatures. La première méthode aurait quant à elle attribué une probabilité à chacune des deux signatures. Le calcul des probabilités de correspondance des signes possédant au moins une signature dans l'ensemble des signatures potentielles est réalisé comme suit. D'abord, la probabilité à priori de ce signe, c'est-à-dire la probabilité déduite uniquement de la base de connaissances sans considérer les observations, est calculée grâce à la formule

$$P'(S) = A/B \tag{5.1}$$

où A est la somme du nombre d'occurrences de toutes les signatures du signe présentent dans la base de connaissances et B est la somme du nombre d'occurrences de toutes les signatures de la base de connaissances (tout signe confondu)

Ensuite, la probabilité $P'(S)$ est multipliée par la somme du nombre d'occurrences des signatures de S présentes dans l'ensemble des signatures potentielles.

Chapitre 6

Implémentation de la solution

La structure générale de notre implémentation est illustrée à la figure 6.1. Comme illustré, trois modules ont été développés.

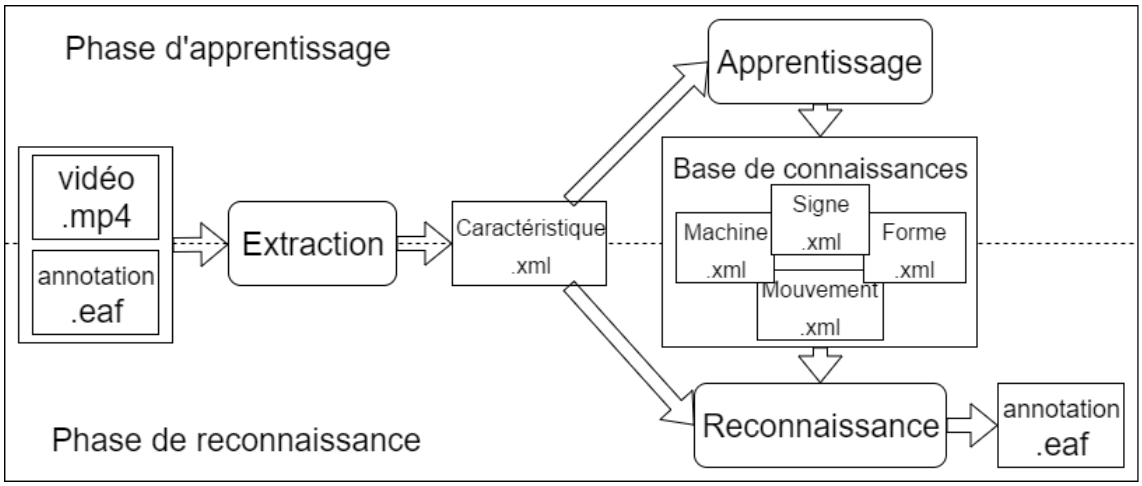


FIGURE 6.1 – Structure générale de l'implémentation.

Le premier, le module d'extraction, prend en entrée les vidéos du laboratoire ainsi que les fichiers d'annotations correspondants et produit un fichier de caractéristiques pour les signes présents dans les vidéos. Le fichier d'annotations est actuellement nécessaire pour la phase d'apprentissage et la phase de reconnaissance. Cela peut s'expliquer par le fait que la segmentation automatique des signes n'a pas été développée dans la solution actuelle. Un fichier contenant les temps de début et de fin de chaque signe doit donc être fourni lors de la phase de reconnaissance. Les fichiers ELAN contenant déjà ces informations nous avons décidé de réutiliser ceux-ci pour fournir les informations temporelles. En outre, cela nous permet de réutiliser les composants de lecture des fichiers ELAN.

Le fichier de caractéristiques produit est ensuite utilisé soit par le module d'apprentissage soit par le module de reconnaissance. Le module d'apprentissage va remplir une base de

connaissances sur base de ce fichier. Le module de reconnaissance va quant à lui utiliser le fichier de caractéristiques en conjonction avec la base de connaissances afin de produire un fichier contenant les annotations des vidéos.

Dans la suite de ce chapitre, nous détaillerons les choix technologiques réalisés ainsi que les différents composants de l'implémentation. Cela inclut les données reçues et produites ainsi que les trois modules réalisant les différents traitements.

6.1 Choix technologiques

6.1.1 OpenCV

La mise en place des différentes méthodes de traitement d'images (extraction de caractéristiques, suivi, etc.) a été réalisée grâce à la librairie OpenCV. OpenCV est une librairie open source supportant les langages C, C++, Python, MATLAB et Java. Elle a été conçue pour faciliter le développement de logiciels utilisant la vision par ordinateur et le machine learning. Nous avons fait ce choix de librairie car lors de nos recherches sur les travaux déjà réalisés dans le domaine de la reconnaissance gestuelle, OpenCV était souvent citée. En outre, comme indiqué sur le site officiel, la librairie est utilisée par un large nombre de développeurs et par de grandes entreprises telles que Google et Microsoft. Nvidia présente même OpenCV comme "la première librairie pour la vision par ordinateur, le traitement d'images et le machine learning"[1].

La librairie contient une multitude d'algorithmes de vision par ordinateur et de machine learning (plus de 2.500 selon le site officiel). Nous n'aborderons toutefois que les principales méthodes que nous avons utilisées ainsi que la classe `Mat` permettant de manipuler les images. Pour un aperçu plus complet des différentes méthodes disponibles, nous renvoyons le lecteur intéressé à la documentation officielle[2].

La classe la plus importante de la librairie dans le cadre de notre travail est la classe `Mat`. Cette classe permet de représenter des tableaux numériques denses à n dimensions et à k canaux, c'est-à-dire des tableaux à n dimensions dont chaque élément est associé à un k-uplet de valeurs en mémoire. La classe peut être utilisée pour stocker des matrices, des nuages de points, des images, des histogrammes, etc. Toutes les méthodes de traitement d'images de la librairie manipulent donc des objets de type `Mat`. Pour cette raison, lorsque nous parlons d'image dans la suite de ce document, nous entendrons en réalité un objet `Mat`. Nous ferons parfois également référence à la classe `Mat` en utilisant le terme "matrice".

Nous n'avons utilisé qu'un petit ensemble des méthodes disponibles dans OpenCV. Parmi celles-ci, les plus notables sont :

— `calcHist` et `calcBackProject` permettant respectivement de calculer l'histogramme d'un ensemble de `Mat` et de calculer la rétro-projection d'un histogramme sur base d'une image initiale. Calculer la rétro-projection d'un histogramme revient à calculer

1. `https://developer.nvidia.com/opencv`
2. `http://opencv.org/`

une nouvelle matrice dont chaque élément (x, y) a comme valeur celle contenue dans la région de l'histogramme correspondant à la couleur du pixel (x, y) de l'image de départ. Par exemple, si le pixel $(0, 0)$ de l'image initiale possède la couleur c et que la valeur correspondant à c dans l'histogramme vaut 5, alors l'élément $(0, 0)$ de la nouvelle matrice possédera la valeur 5.

— Les méthodes permettant de trouver les contours et les enveloppes convexes des objets d'une image telles que `findContours` et `convexHull`.
— Les méthodes appliquant des opérations morphologiques et des filtres aux images. Cela inclut notamment les méthodes `open`, `close` et `medianBlur` appliquant respectivement une opération d'ouverture, une opération de fermeture et un filtre médian.
— Les méthodes appliquant les opérations logiques bit à bit. Par exemple, la méthode `bitwise_not` appliquant l'opération "not".
— Les méthodes de dessin permettant de dessiner diverses formes géométriques sur une image. Par exemple, la méthode `circle` permet de dessiner un cercle en spécifiant, entre autre, son centre, son rayon et une couleur de trait.

6.1.2 Langage de programmation

OpenCV supporte les langages C, C++, Python, MATLAB et Java. Bien qu'elle ait été écrite en C++ et que, par conséquent, la version C++ de la librairie est mieux documentée et plus largement utilisée, nous avons décidé d'utiliser le langage Java. La raison de ce choix est essentiellement liée aux contraintes de temps auxquelles ne devions faire face. La durée de notre stage étant assez courte, nous avons jugé qu'il nous était impossible d'apprendre à la fois les techniques de traitement d'images, la librairie OpenCV et le langage C++. Ce choix de langage a parfois eu des effets négatifs. En effet, la documentation et la majorité des exemples d'OpenCV font référence à la version C++ de la libraire. Or, les structures de la version C++ et de la version Java sont parfois légèrement différentes et certaines classes se retrouvent dans des espaces de noms différents d'une version à l'autre. Une fois les différences de structure entre les deux versions bien assimilées, ces problèmes se sont toutefois dissipés.

6.1.3 Persistance des données

La persistance des données produites par les différents modules a été mise en place en utilisant des fichiers XML. Nous avons fait ce choix pour plusieurs raisons.

Concernant le résultat produit par le module de reconnaissance, la principale raison était la compatibilité avec ELAN. L'objectif final étant de produire un fichier d'annotations ELAN et ces fichiers utilisant une syntaxe XML, le choix d'un fichier XML pour stocker les résultats du module de reconnaissance lors de nos tests semblait évident.

Les résultats de l'extraction et la base de connaissances ont, quant à eux, été stockés au format XML principalement pour des raisons pratiques. Travaillant avec un ensemble de données très réduit et de manière assez locale, nous avons jugé le XML plus pratique

qu'une base de données SQL. De plus, nous manipulions déjà des fichiers XML avec les fichiers ELAN et le résultat de la reconnaissance.

Il est important de noter que le code de gestion des fichiers XML a été encapsulé dans des classes prévues uniquement à cet effet. Dans le cas où le format XML devrait être changé, très peu de code, en dehors de ces classes, ne devrait être changé.

6.1.4 Autres librairies

Outre OpenCV, nous avons utilisé certaines librairies afin de gérer l'accès aux données (stockées au format XML) et de mettre en place l'interface graphique.

La lecture et l'écriture des fichiers XML ont été réalisées avec la librairie JOOX. JOOX est un wrapper pour le package `org.w3c.dom`. La librairie permet donc de manipuler les documents XML comme le ferait le DOM mais possède une syntaxe moins verbeuse, permettant ainsi du code moins complexe. La syntaxe de la librairie est fortement inspirée de la librairie JavaScript `JQuery`. JOOX contient donc une méthode `$` permettant, notamment, de sélectionner un élément ou une liste d'éléments DOM. La figure 6.2 illustre l'utilisation de JOOX.

```
// Find the order at index 4 and add an element "paid"
$(document).find("orders").children().eq(4).append("<paid>true</paid>");

// Find those orders that are paid and flag them as "settled"
$(document).find("orders").children().find("paid").after("<settled>true</settled>");
```

FIGURE 6.2 – Exemples d'utilisation de JOOX. Image reproduite de `https://github.com/jOOQ/jOOX`

Pour les interfaces graphiques utilisées par le programme, nous avons choisi d'utiliser la librairie `swing`. Il n'y a pas de raison particulière à l'utilisation de celle-ci plutôt que `JavaFx` ou d'autres librairies graphiques. Cependant, l'interface graphique n'étant pas l'élément le plus important, nous avons décidé d'utiliser `swing` car nous étions déjà familiers avec cette librairie.

6.2 Représentation des données

6.2.1 Données en entrée

Les données reçues en entrée sont constituées des vidéos à traiter ainsi que des fichiers d'annotations ELAN.

Les vidéos du LSFB-lab ont été enregistrées au format mp4. En outre, les vidéos ont un taux de rafraîchissement de 50 frames par seconde. Cela implique que, pour chaque seconde de vidéo, 50 images doivent être traitées. La durée d'une vidéo varie énormément en fonction du sujet de la discussion filmée. Les vidéos les plus courtes possèdent une durée d'environ

deux minutes. Les vidéos les plus longues peuvent en revanche durer quinze minutes ou plus. Cette variation de durée implique que certaines vidéos contiennent plus de signes à analyser et seront plus longues à traiter.

Les deux signeurs de chaque discussion apparaissent sur deux vidéos séparées et synchronisées. Ces vidéos contiennent uniquement les moments où les signeurs parlent l'un à l'autre. Les moments où le modérateur explique la tâche ou intervient pour donner une précision aux signeurs sont coupés. Ces coupures sont indiquées par des plans de couleur unie. Plusieurs couleurs sont utilisées pour ces coupures. L'ordre des ces couleurs est toujours le suivant : rouge, jaune, vert, orange, turquoise, bleu et blanc.

Les fichiers d'annotations ELAN sont des fichiers au format ".eaf". Ces fichiers disposent d'une syntaxe XML. Un fichier contient les informations concernant les deux signeurs d'une discussion et fait correspondre à chaque signe, une annotation. Outre quelques informations utiles pour le programme ELAN, le fichier contient une liste de "time slot" [3] prédéfinis et, pour chaque signeur, les annotations de la main droite et de la main gauche. Chaque annotation possède un identifiant, un premier time slot (début du signe), un second time slot (fin du signe) et une "valeur d'annotation", c'est-à-dire le tag du signe réalisé. De plus, si la traduction de la vidéo a déjà été réalisée, le fichier contient également la traduction des propos pour chaque signeur.

6.2.2 Fichier de caractéristiques

Le fichier résultant de l'étape d'extraction contient les caractéristiques de tous les signes observés dans la conversation. Afin de refléter la structure des fichiers ELAN, chaque fichier d'extraction contient les informations des vidéos des deux signeurs d'une même conversation. En outre, pour permettre un re-traitement ultérieur plus rapide, le fichier contient plus d'informations que les seules caractéristiques des signes. Pour chaque signe réalisé, le fichier de caractéristiques contient un élément "track". Un track représente une portion de vidéo correspondant à un signe et contient notamment les informations utiles à l'apprentissage et/ou à la reconnaissance. Ces informations proviennent soit de l'analyse de la vidéo soit du fichier d'annotations correspondant. La liste complète de ces informations est la suivante :

- le tag du signe ;
- le numéro du signeur dans le corpus ;
- la vidéo contenant le signe ;
- le début et la fin du signe dans la vidéo (en millisecondes) ;
- la ou les mains nécessaires à la réalisation du signe ("MD" pour main droite, "MG" pour main gauche ou "MM" pour les deux mains) ;
- s'il y a eu, lors de la réalisation du signe, une fusion des mains ;
- s'il y a eu, lors de la réalisation du signe, une fusion entre les mains et la tête ;
- les informations de forme et de position de chaque main ;
- les informations de mouvement de chaque main.

3. Chaque time slot est une paire identifiant-valeur en millisecondes.

Les informations de forme et de position sont représentées par des machines à états (voir figure 6.3). Une machine à états est caractérisée par un identifiant et une liste d'états. Chaque état contient, pour chaque main impliquée dans la réalisation du signe, la forme et la position relative à la tête de cette main. Une forme est caractérisée par une liste d'angles (chacun caractérisé par ses coordonnées et son amplitude) et par deux booléens précisant respectivement s'il y a fusion entre les mains ou entre la main et la tête. La position relative à la tête est représentée grâce à une énumération. Ce choix a été pris car seules sept positions sont possibles. Pour rappel, la position relative est assignée en fonction de l'endroit où se trouve la tête et de la zone dans laquelle se trouve le centre du blob, comme expliqué à la section 5.2.2.

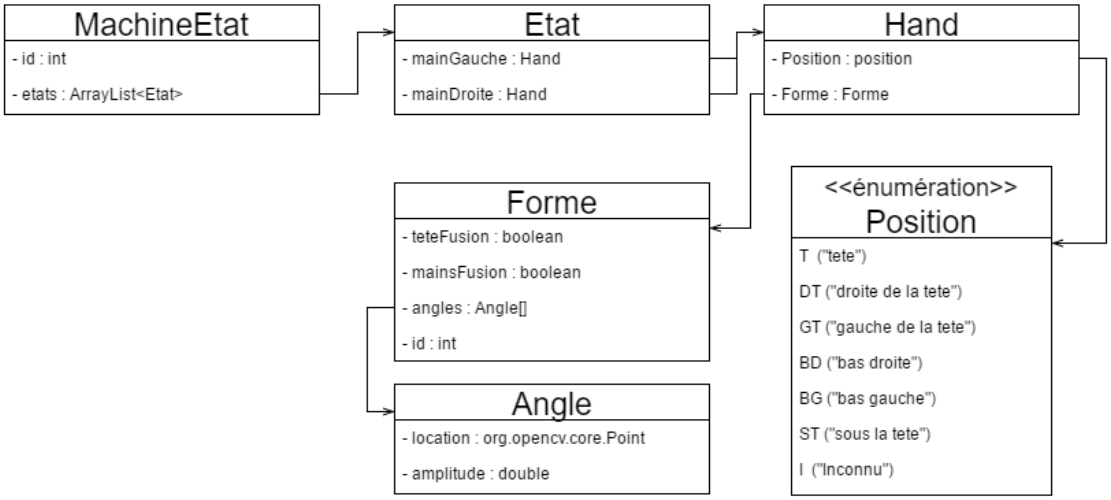


FIGURE 6.3 – Classes utilisées pour représenter les machines à états.

Les informations de mouvements sont quant à elles représentées grâce à des listes de points. Chaque mouvement est caractérisé par un identifiant et par une liste de points représentant la succession de positions dans l'image occupées par le centre de la main durant le mouvement.

6.2.3 Base de connaissances

La base de connaissances est constituée de quatre fichiers XML :
— Signe.xml reprenant les signes et leurs informations ;
— Mouvement.xml contenant les différents mouvements ;
— Machine.xml contenant les différentes machines à états ;
— Forme.xml contenant les différentes formes observées.

Chaque signe du fichier Signe.xml possède un attribut `nom` correspondant au tag associé à ce signe et une liste d'éléments `signature` représentant toutes les façons de réaliser le signe. Une signature possède quatre attributs : l'identifiant de sa machine à états, l'identifiant du

mouvement de la main gauche, l'identifiant du mouvement de la main droite et le nombre de fois où cette signature a été utilisée. Lorsque le signe est un signe réalisé à une main, l'identifiant du mouvement de la main qui n'est pas impliquée dans la réalisation du signe se voit assigner la valeur -1. La structure de ce fichier est illustrée par le listing 1

Les mouvements, formes et machines à états sont représentés tels qu'expliqués au point 6.2.2.

```
<signe nom="EXEMPLE">
    <signatures>
        <signature machine = "0" mouvDroit = "0" mouvGauche = "-1"
↪  occurrence = "25"/>
        <signature machine = "10" mouvDroit = "145" mouvGauche = "2"
↪  occurrence = "1"/>
        <signature machine = "20" mouvDroit = "10" mouvGauche = "-1"
↪  occurrence = "12"/>
    </signatures>
</signe>
```

Listing 1: Exemple de fichier Signe.Xml

6.2.4 Données en sortie

Le fichier produit par l'étape de reconnaissance est un fichier XML comportant, pour chaque signe à reconnaitre, un élément `Annotation`. Une annotation est caractérisée par un couple d'attributs représentant les temps de début et de fin du signe et, de trois éléments `Prediction` représentant les trois tags ayant le score de prédiction le plus élevé par ordre décroissant.

En outre, afin de pouvoir réaliser des tests sur la précision de notre solution, les fichiers d'annotations produits en vue de tests contiennent, pour chaque annotation, un attribut reprenant le tag correct. De cette façon, la vérification de la reconnaissance peut être effectuée plus aisément.

6.3 Extraction des caractéristiques

Avant de réaliser la phase d'extraction, un pool de threads est créé. Ce pool sera utilisé afin de pouvoir traiter les différentes vidéos en parallèle. En outre, un objet de type `Phaser` est utilisé afin de s'assurer de la terminaison du traitement de chaque vidéo avant de réaliser l'enregistrement du fichier de caractéristiques.

L'extraction en tant que telle commence par la sélection par l'utilisateur de la (des) vidéo(s) à analyser et du fichier ELAN correspondant. Cette sélection peut se faire via

une fenêtre, appelée `FirstWindow`, permettant de spécifier l'emplacement des vidéos et du fichier ELAN. Pour chaque vidéo, un objet de type `VideosAnalyser` est créé. Cet objet est chargé de réaliser l'entièreté de la phase d'extraction de la vidéo. Cela commence par la création d'une liste d'objets `track` tirée du fichier ELAN. Ensuite, une méthode est soumise au pool de threads. Cette méthode va s'enregistrer auprès du `Phaser`, réaliser un appel à `VideosAnalyser.analyze()` (contenant la logique d'extraction des caractéristiques des signes) et finalement signaler sa terminaison auprès du `Phaser`. Après terminaison de toutes les analyses, le fichier de caractéristiques correspondant à celles-ci est créé et enregistré.

La méthode `VideosAnalyser.analyze()` contient donc la logique d'extraction des caractéristiques. Dans un premier temps, les objets permettant de dégager ces caractéristiques des images de la vidéo vont être construits. Ces objets incluent notamment un `BlobIdentifier` et deux `MouvementIdentifier` (nous reviendrons sur ces deux classes une fois la logique d'extraction terminée). Ensuite, tant que tous les tracks de la liste n'ont pas été parcourus, la vidéo sera lue image par image.

- Tant que il n'y a pas de signe dans la vidéo, c'est-à-dire tant que le moment correspondant à l'image actuelle (nombre d'images*20 millisecondes) est plus petit que le moment de début du track actuel, aucun traitement particulier n'est réalisé. La vidéo est simplement lue grâce à `lireFrame()`.
- Tant que le moment correspondant à l'image actuelle est compris entre le moment de début et le moment de fin du track actuel, les caractéristiques des mains sont calculées.
- Lorsque le moment correspondant à l'image actuelle est plus grand que le moment de fin du track, les caractéristiques extraites sont assignées aux attributs de la track. Le traitement de la vidéo se poursuit ensuite avec le track suivant.

La classe `BlobIdentifier`, comme son nom l'indique, s'occupe de l'identification complète des blobs. Elle contient notamment toutes les méthodes permettant d'isoler les groupes de pixels de couleur peau (6.3.2) et la méthode de suivi (6.3.3).

La classe `MouvementIdentifier` contient, quant à elle, la logique permettant de calculer les centres des mains dans le but de calculer les mouvements comme expliqué au point 6.3.5.

6.3.1 reconnaitre un pixel de couleur peau

Pour des raisons d'adaptabilité, la signature de la méthode réalisant la détection est définie dans une interface, `SkinDetector`. Grâce à cela, l'implémentation de la détection peut facilement être modifiée sans que le reste du code ne subisse trop de changements. Cette interface est implémentée par les classes `DoubleHistogramSkinDetector` et `HistogramSkinDetector`. Pour mettre en place la méthode de détection combinant deux types d'histogrammes comme expliqué précédemment, la classe `DoubleHistogramSkinDetector` possède deux champs de type `HistogramSkinDetector`. Cette dernière classe contient la logique permettant la réalisation de la détection avec un seul type d'histogramme, que nous détaillerons dans la suite de cette section. L'implémentation de la méthode de détection de peau présente dans la

classe `DoubleHistogramSkinDetector` consiste alors en l'application d'une opération "OR" sur les résultats obtenus lors de la détection réalisée par les deux instances de la classe `HistogramSkinDetector`. Notons que l'opération "OR" est réalisée à l'aide d'une méthode définie dans OpenCV.

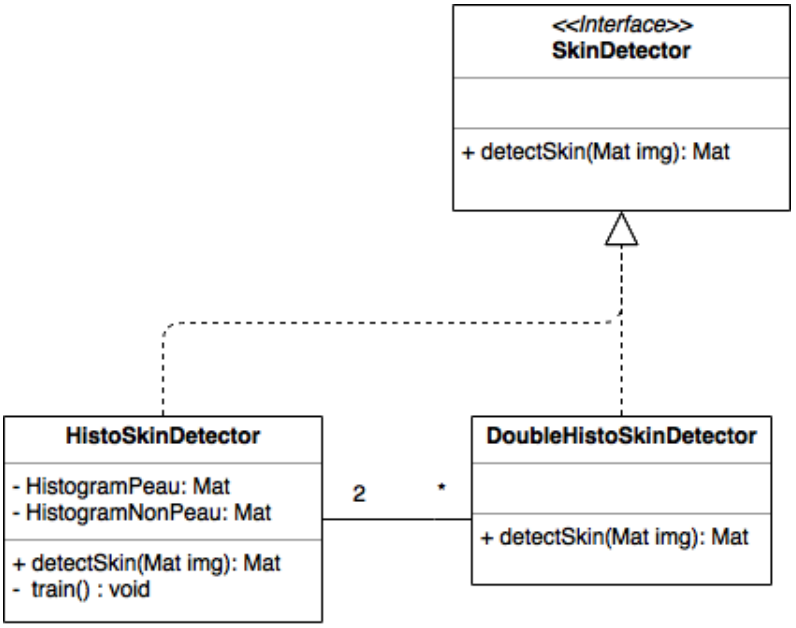


FIGURE 6.4 – Diagramme de classes du composant de détection de peau.

Détection avec un seul type d'histogramme

La classe `HistogramSkinDetector` implémente donc la logique permettant la détection sur base d'un seul type d'histogramme. Pour rappel, cette détection fonctionne en deux étapes : la construction des histogrammes et la classification des pixels sur base de ces histogrammes.

La création des histogrammes représentant les classes peaux et non peau n'est réalisée qu'à l'instanciation de la classe `HistogramSkinDetector`. À cette fin, une série d'images et les masques[4] définissant les pixels de peau dans celles-ci sont chargés en mémoire. Pour chaque image, l'histogramme représentant la classe des pixels de peau au sein de l'image est calculé grâce au masque de cette dernière et des méthodes d'OpenCV. En outre, le nombre de pixels de peau de l'image est calculé en comptant le nombre de pixels ayant une valeur différente de zéro dans le masque. La valeur ainsi calculée est alors additionnée au nombre total de pixels de peau (`nbPeau`) toutes images confondues. Ce nombre total de pixels servira par la suite à calculer les probabilités d'appartenance à la classe peau. Une fois l'histogramme de peau construit, le masque définissant les zones de "non-peau" est calculé

4. Un masque est une image définissant les pixels à traiter dans une autre image. Typiquement, les pixels ne devant pas être traités auront une valeur 0 dans le masque tandis que ceux à traiter auront une valeur différente de 0.

en inversant le masque initial à l'aide d'une opération "NOT". L'histogramme représentant la classe des pixels "non-peau" au sein de l'image est alors calculé en appliquant le même processus que pour la classe peau mais en utilisant le nouveau masque. De même, le nombre de pixels "non-peau" dans l'image est calculé en comptant le nombre de pixels ayant une valeur différente de zéro dans le nouveau masque. La valeur est ensuite additionnée au nombre total de pixels "non-peau" (`nbNonPeau`). Les deux histogrammes ainsi calculés sont finalement additionnés élément par élément aux histogrammes représentant les classes toutes images confondues.

Lors d'un appel à la méthode `detectSkin()`, la classification des pixels commence par le calcul des rétro-projections (backprojection) des histogrammes par rapport à l'image devant être analysée (voir méthode calcBackProject au point 6.1.1).

Une fois les rétro-projections calculées, les valeurs obtenues des histogrammes sont transformées en probabilités. Cette transformation est effectuée en divisant la valeur de chaque pixel des rétro-projections par le nombre total de pixels de leur classe respective. Par exemple, les valeurs de la rétro-projection de l'histogramme de la classe peau seront divisées par le nombre total de pixels représentés par ce dernier c'est-à-dire `nbPeau`. Chaque pixel (x, y) possède ainsi une probabilité $P(peau)$ d'appartenir à la classe "peau" et une probabilité $P(\neg Peau)$ d'appartenir à la classe "non-peau". Pour chaque pixel (x, y) de l'image, le ratio entre $P(peau)$ et $P(\neg Peau)$ est calculé.

- Si ce ratio est supérieur à 4,5 (valeur définie empiriquement), le pixel (x, y) de l'image binaire résultat se voit attribuer la couleur "blanc", c'est-à-dire que la valeur (x, y) de la matrice résultat vaut 255.
- Sinon, le pixel (x, y) de l'image binaire se voit attribuer la couleur "noir", ce qui se traduit par une valeur (x, y) de 0 dans la matrice résultat.

Une fois le processus appliqué à chaque pixel, l'image binaire représentant la classification des pixels a été construite. Avant de retourner celle-ci, certaines opérations morphologiques et certains filtres sont toutefois appliqués pour réduire le bruit et le nombre de faux-positifs. Les opérations et filtres que nous avons décidé d'utiliser sont une opération d'ouverture et un filtre médian. Pour mettre en place ces techniques, nous avons utilisé les méthodes fournies par OpenCV.

6.3.2 Identifier les groupes de pixels de couleur peau

L'identification des groupes de pixels de peau consiste à extraire les blobs de la matrice (`imageBin`) retournée par la méthode de détection expliquée au point précédent.

Nous utilisons la classe "Blob" pour représenter le concept de blob. Un blob contient :

- Les informations permettant son suivi :
  - deux points [5] définissant un rectangle (aussi appelé fenêtre) dans lequel est inscrit

5. Nous avons travaillé avec des points de type (ligne,colonne) comme pour les matrices. Cependant, les méthodes de "dessin" prévues par OpenCV fonctionnent avec des points de type (colonne,ligne). Les deux coordonnées du point devront donc être inversées lorsque ces méthodes seront utilisées.

le blob ;
- le nombre de pixels de peau ;
- la (les) partie(s) du corps représentée(s) ;
- un booléen indiquant s'il est en bordure ou non ;
- un numéro identifiant.

— les informations servant à la reconnaissance :
- la forme du blob ;
- la position du blob relativement à la tête.

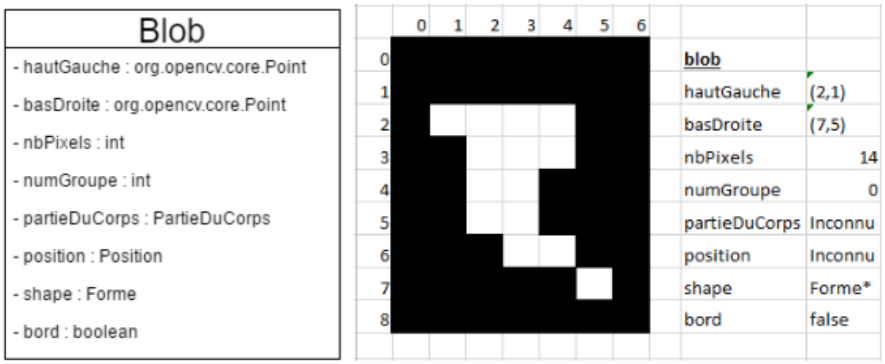


FIGURE 6.5 – Diagramme de classe de la classe "Blob" (à gauche) et exemple d'instance (à droite).

Afin de démarrer l'identification des blobs, une seconde matrice (`check`) de la même taille que la matrice image est utilisée. Cette matrice permet de retenir à quel blob chaque pixel de peau appartient. Avant l'identification, les éléments de `check` sont initialisés à -1.

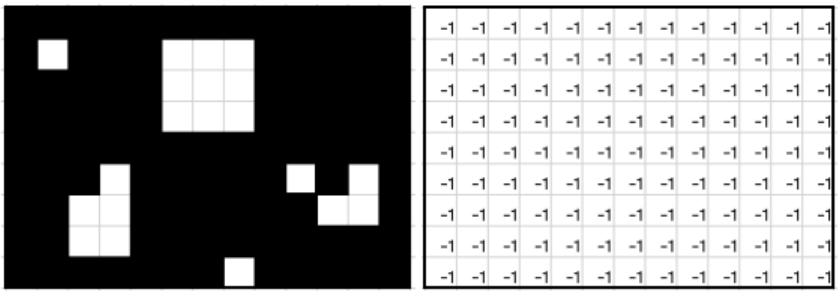


FIGURE 6.6 – Un exemple de imageBin (à gauche) et de la matrice check initialisée (à droite)

Outre les matrices `imageBin` et `check`, une liste de blobs est utilisée afin de stocker les blobs trouvés. Cette liste sera par la suite appelée `listGroupe`.

Pour l'identification à proprement parler, `imageBin` est parcourue ligne par ligne et colonne par colonne. Pour chaque pixel $(x, y) \in$ `imageBin`, si la valeur du pixel est de 0

(noir), nous passons simplement au pixel suivant. Si la valeur du pixel est 255 (blanc), une vérification des cases adjacentes à (x, y) dans `check` est réalisée [6].

— Si elles ont toutes comme valeur -1 : le pixel appartient à un nouveau blob. Ce nouveau blob se voit attribuer comme numéro identifiant la valeur correspondant à la taille de `listGroupe` et est ajouté à la liste. Ceci permet aux blobs contenus dans `listGroupe` d'avoir comme numéro leur position dans la liste. Finalement, l'élément (x, y) de `check` se voit attribuer le numéro du blob.

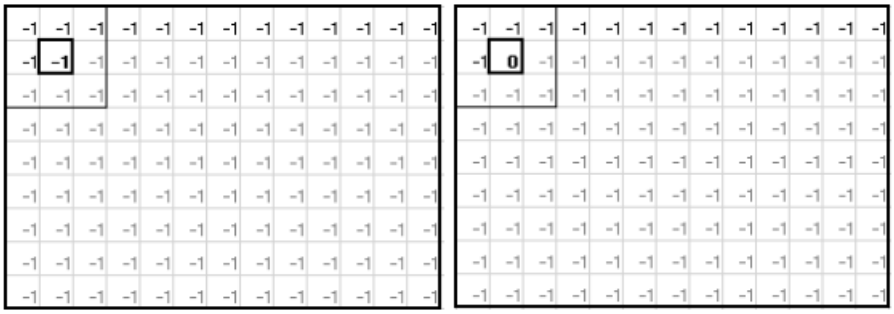


FIGURE 6.7 – Exemple d'identification lorsque les cases adjacentes valent toutes -1.

— Sinon, si les cases différentes de -1 ont toutes la même valeur : le pixel appartient au même blob que ces cases. Le pixel (x, y) est alors ajouté au blob et l'élément (x, y) de `check` se voit attribuer le numéro identifiant du blob.

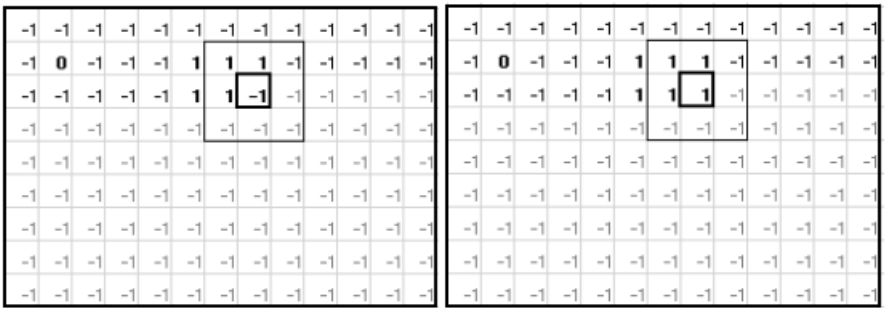


FIGURE 6.8 – Exemple d'identification lorsque les cases adjacentes différentes de -1 ont la même valeur.

— Sinon, si les cases différentes de -1 n'ont pas toutes la même valeur : les blobs correspondants à ces cases ainsi que le pixel actuel ne forment en réalité qu'un seul et unique blob. Ces blobs sont alors fusionnés avec la méthode `fusionner()`. Le nouveau blob ainsi formé aura comme numéro identifiant celui du premier blob. Le pixel en cours est ensuite ajouté au blob. Finalement, l'élément (x, y) de `check` se voit attribuer le numéro identifiant du blob.

6. Notez qu'il ne sert à rien de vérifier les cases adjacentes qui n'ont pas encore été traitées car elles seront de toutes façon à -1. Par conséquent, nous pouvons nous contenter d'analyser uniquement la valeur des trois cases au dessus et de celle à gauche.

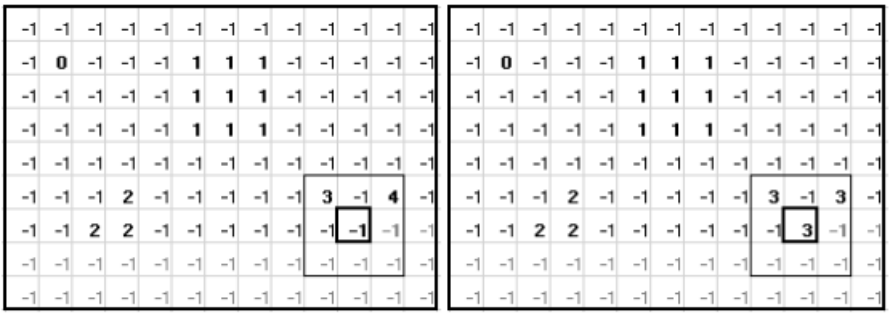
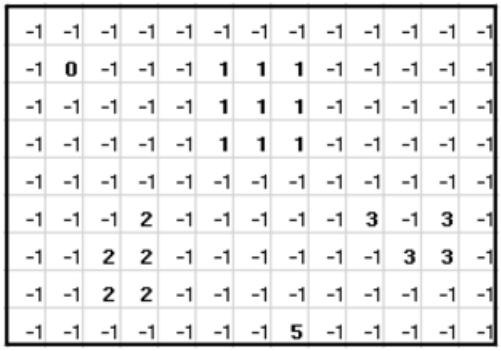


FIGURE 6.9 – Exemple d'identification lorsque les cases adjacentes différentes de -1 ont des valeurs différentes.

La matrice `check` correspondant à l'image présentée à la figure 6.6, sera la suivante :



La méthode `fusionner()` prend les deux blobs à fusionner en paramètre et fonctionne comme suit : le second blob est fusionné dans le premier grâce à la méthode `fusionner()` de la classe `Blob`. Le second blob est ensuite réinitialisé. De plus, les éléments de `check` ayant comme valeur l'identifiant du second blob sont modifiés pour prendre la valeur de l'identifiant du premier blob. La méthode de fusion retourne finalement le numéro identifiant du premier blob.

Après avoir parcouru tous les pixels de l'image i, `listGroupe` contient tous les blobs découverts, y compris ceux réinitialisés suite à une fusion. Ces derniers, qui ne contiennent donc aucun pixel, ne sont plus d'aucune utilité à la fin de l'identification. Nous les avions conservés jusqu'à présent afin que les identifiants des blobs contenus dans `listGroupe` reflètent leur position dans la liste. Ainsi, il était facile de retrouver un objet blob possédant un certain identifiant. Pour la suite des traitements, deux choix sont possibles : soit `listGroupe` est conservée sans enlever les blobs inutiles, soit une table de hashage possédant comme clé l'identifiant des blobs est créée afin de stocker ceux-ci. Nous avons fait le choix d'une table de hashage appelée `hashGroupe`.

6.3.3 Suivi

Dans cette section, nous détaillerons comment est implémenté l'algorithme introduit au point 5.1.3. L'algorithme se basant sur l'image précédente, nous allons d'abord expliquer la gestion du cas de la première image ainsi que des cas où une coupure est survenue. Ensuite, nous détaillerons l'implémentation du suivi pour les images suivantes. Pour finir, nous parlerons de la gestion des fusions et dé-fusions ainsi que des exceptions liées aux incertitudes en bordure de cadre.

La première image (i=0)

Comme expliqué précédemment, le suivi ne peut être réalisé sur base de l'image précédente dans le cas où l'image à traiter est la première. L'utilisateur est donc invité à indiquer manuellement l'emplacement des différentes parties du corps. Pour ce faire, une classe, appelée StartWindow, est utilisée. Cette classe utilise la librairie `swing` de Java afin d'afficher l'image et les différents éléments de l'interface graphique (boutons, etc.).

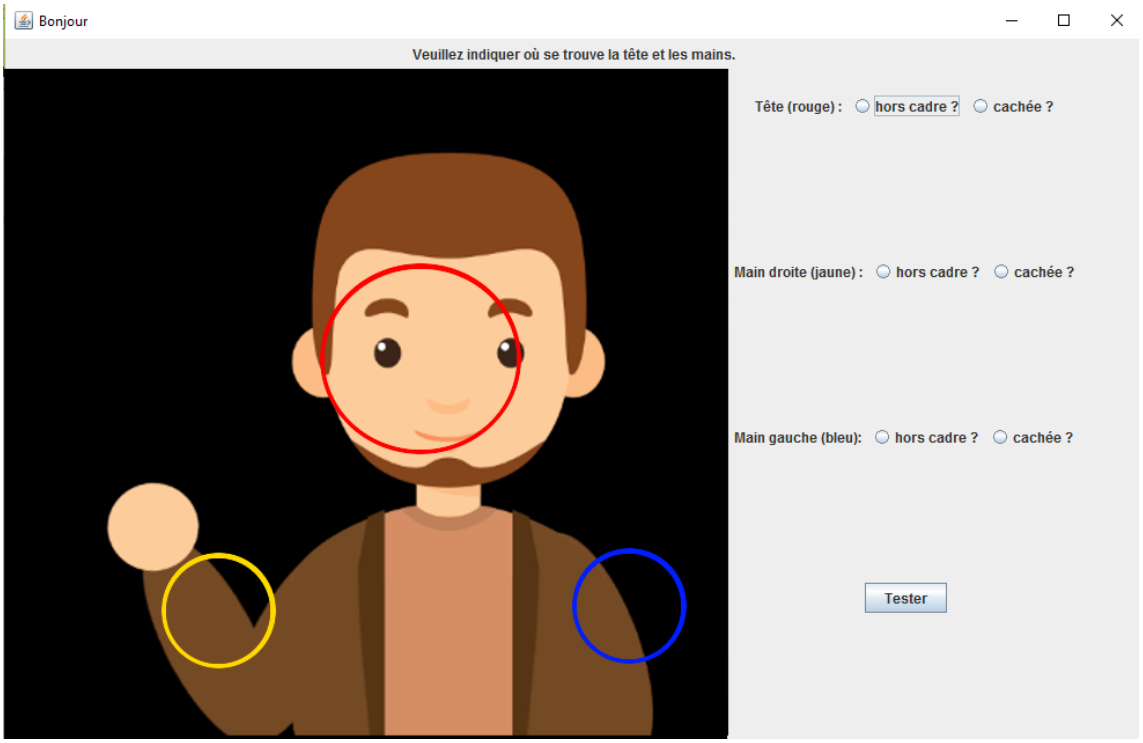


FIGURE 6.10 – Exemple de StartWindow.

Les emplacements des différentes parties du corps sont représentés visuellement par trois cercles dessinés sur l'image grâce à la méthode Circle d'OpenCV. Pour indiquer clairement à l'utilisateur quel cercle est associé à quelle partie du corps, des couleurs différentes sont utilisées pour chacun : rouge pour la tête, jaune pour la main droite et bleu pour la main gauche. De plus, ils sont initialement positionnés de sorte que

— Le cercle de la tête soit positionné en haut et au milieu de l'image.
— Le cercle de la main droite soit positionné en bas de l'image à gauche
— Le cercle de la main gauche soit positionné en bas de l'image à droite.

Ensuite, l'utilisateur doit positionner chacun de ces cercles à l'endroit où se trouve la partie du corps à laquelle il est associé. De plus, si une partie du corps n'est pas visible dans l'image, l'utilisateur doit indiquer si celle-ci est hors cadre ou cachée à l'aide de boutons de type "radio". Dans un cas comme dans l'autre, le cercle correspondant à la partie du corps doit être déplacé à la position la plus proche de celle où devrait se situer la partie du corps si elle était visible à l'image. Par exemple, dans le cas présenté par la figure 6.10, l'utilisateur devra cliquer le bouton "hors cadre ?" à côté de "Main gauche". Il devra également positionner le cercle bleu sur le bord de l'image le plus proche de l'endroit où réapparaîtra la main.

Une fois les cercles déplacés et les boutons radios cochés, une vérification est réalisée afin de s'assurer que les blobs correspondant aux mains et à la tête sont correctement détectés. Cette vérification démarre lorsque l'utilisateur appuie sur le bouton "Tester". Suite à cette vérification

— si tous les blobs ont été détectés, ceux-ci sont ajoutés à une liste de blobs, appelée `blobs`. Un bouton "Confirmer" permettant de fermer la fenêtre et de poursuivre le traitement est affiché. Le programme principal récupérera alors les blobs présents dans `blobs` pour commencer le suivi.
— si tous les blobs n'ont pas pu être détectés, un message d'erreur apparaît. L'utilisateur doit alors modifier les positions des cercles posant problème.

Les coupures

Les coupures sont rares dans les vidéos du LSFB-lab. Cependant, il est nécessaire de les prendre en compte comme expliqué à la section 5.1.3. Les coupures présentes dans les vidéos du laboratoire sont marquées par une image composée d'une couleur unie. Cette couleur peut varier d'une coupure à l'autre. La détection des coupures ne peut donc pas se baser sur la détection d'une couleur en particulier.

Pour repérer ces coupures, l'image binaire résultant de la détection de peau est analysée. Si tous les pixels de cette image sont blancs ou si tous les pixels sont noirs, il s'agit d'une coupure. Dans ce cas, aucun suivi n'est réalisé. Lorsqu'une image non unie est rencontrée, le suivi peut reprendre. Comme dans le cas de la première image, l'utilisateur est invité à signaler manuellement la position des parties du corps dans l'image à l'aide d'une `StartWindow`.

Les image t_i où i>0

Pour identifier les parties du corps sur les images t_i où i>0, nous utilisons la méthode `identification()` du `BlobIdentifier`. La méthode `identification()` prend en paramètre une liste de blobs `bprecedents` qui sont les blobs représentant les parties du corps sur l'image t_{i-1} et une image `img` qui est l'image t_i. La méthode `identification()` retourne `resultatIdentification` qui est la liste des blobs représentant les parties du

corps sur l'image t_i. La méthode `identification()` lance deux types d'exception appelés `DefusionException` et `BordureException` dont la gestion sera détaillée plus loin.

Le fonctionnement de la méthode `identification()` est le suivant : pour commencer, il faut réinitialiser certaines variables. La méthode `findBlob()` présente dans le `BlobIdentifier` sera appelée. Pour rappel, cette méthode identifie les blobs présents dans l'image comme expliqué au point 6.3.2 ; `hashGroupe` contient donc les blobs présents dans l'image t_i.

Ensuite, pour chaque blob `blobPrec` présent dans `bprecedent`, un équivalent parmi les blobs dans `hashGroupe` est cherché. Pour cela, en fonction de l'état de `blobPrec`, la recherche sera légèrement différente. Les blobs précédents peuvent avoir 3 états différents :

- Hors cadre : `blob.isBord()` = `true` et `blob.getNbPixels()` = 0.
- En bordure : `blob.isBord()` = `true` et `blob.getNbPixels()` $=/=$ 0.
- Dans l'image : `blob.isBord()` = `false`.

Traitement des blobs hors cadre : La fonction `getGroupesFromFenetre(blobPrec)` est utilisée. Elle renvoie une liste d'entiers, `groupes`, correspondant aux identifiants des blobs de l'image actuelle présents dans la fenêtre de `blobPrec`. Les blobs n'étant pas en bordure de l'image sont ensuite supprimés de `groupes`.

- Si `groupes` est vide : c'est que la partie du corps est toujours hors cadre, `blobPrec` est donc ajouté à `resultatIdentification`.
- Si `groupes.size()`$=$ 1 : la partie du corps de `blobPrec` est attribuée au blob découvert qui est ajouté à `resultatIdentification`.
- Si `groupes.size()` $>$ 1 : le blob le plus proche du centre de `blobPrec` se voit attribuer la partie du corps de `blobPrec` puis, il est ajouté à `resultatIdentification`.

Traitement des blobs en bordure : La fonction `getGroupesFromFBlob()` est utilisée avec, comme arguments, `blobPrec` ainsi que la matrice de check de l'image précédente `checkPrecedent`. La méthode renvoie une liste d'entiers, `groupes`, correspondant aux identifiants des blobs ayant au moins un pixel en commun avec `blobPrec`. Les blobs qui font moins de 400 pixels sont ensuite supprimés de `groupes`, sauf s'ils sont en bordure, car un blob de moins de 400 pixels est beaucoup trop petit pour représenter une partie du corps.

Si aucun blob n'est trouvé (`groupes` est vide) et que le `blobPrec` faisait moins de 50 pixels, la recherche est légèrement élargie car, avec un blob si petit au départ, il est possible qu'il n'aie aucun pixel en commun avec son équivalent. La fenêtre est agrandie de 5 pixels de tous les côtés et les blobs présents dans cette fenêtre sont mis dans `groupes`.

- Si `groupes` est vide : c'est que la partie du corps est sortie du cadre. La fenêtre de `blobPrec` est définie par défaut avec `adaptFenetreHC`. La fenêtre ne désigne alors plus un rectangle dans lequel est inscrit le blob mais plutôt l'endroit où le blob est susceptible de rentrer dans le cadre. Cette nouvelle fenêtre est calculée comme suit : si le blob est sorti en haut ou en bas, la largeur de la fenêtre est définie comme étant 1/10 de la largeur de l'image, et s'il est sorti à gauche ou à droite, la hauteur de la fenêtre est définie comme étant 1/8 de la hauteur de l'image.

`blobPrec` se voit assigner le groupe -1 et le nombre de pixel 0. Ensuite, il est ajouté à `resultatIdentification`.

— Si `groupes.size()`= 1 : la partie du corps de `blobPrec` est attribuée au blob découvert avant d'ajouter ce dernier à `resultatIdentification`.

— Si `groupes.size()` > 1 : nous regardons si `blobPrec` était un blob fusionné (qui représentait 2 parties du corps ou plus). S'il n'était pas fusionné, la partie du corps de `blobPrec` est assignée au blob avec le plus de pixels en commun avec `blobPrec`. Ce blob est alors ajouté à `resultatIdentification`. S'il était fusionné, il est "dé-fusionné" comme expliqué plus loin.

Traitement des blobs pas en bordure : de la même manière que pour les blobs en bordure du cadre, la méthode `getGroupesFromFBlob()` est utilisée pour obtenir une liste d'entiers, `groupes`, correspondant aux identifiants des blobs ayant au moins un pixel en commun avec `blobPrec`. Les blobs qui font moins de 400 pixels sont également supprimés de `groupes` sauf s'ils sont en bordure.

Si aucun blob (`groupes` est vide) n'est trouvé, la recherche est légèrement élargie car il n'y a aucune raison de ne pas trouver de correspondance. La fenêtre est alors agrandie de 5 pixels de tous les côtés et les blobs présents dans cette fenêtre sont mis dans `groupes`. Les recherches sont élargies maximum trois fois.

— Si `groupes` est vide : c'est que le blob recherché est caché (par une manche par exemple). La fenêtre de `blobPrec` est alors adaptée avec `adaptFenetreMC` qui définit une fenêtre de 50 pixels sur 50 pixels centrée sur `blobPrec`. On assigne à `blobPrec` le numéro de groupe -1 et 0 au nombre de pixels. Ensuite, `blobPrec` est ajouté à `resultatIdentification`.

— Si `groupes.size()` = 1 : la partie du corps de `blobPrec` est attribuée au blob découvert avant d'ajouter ce dernier à `resultatIdentification`.

— Si `groupes.size()` > 1 : nous regardons si `blob` était un blob fusionné. S'il n'était pas fusionné, le blob avec le plus de pixels en commun est choisi. La partie du corps de `blobPrec` lui est assignée avant d'ajouter ce dernier à `resultatIdentification`. S'il était fusionné, il sera "dé-fusionné" comme expliqué au point suivant.

À ce stade, `resultatIdentification` contient tous les blobs représentant une partie du corps sur l'image t_i. Le suivi à proprement parlé est donc terminé. Cependant, la méthode `identification()` exécute encore quelques opérations :

— Elle appelle la méthode `calculShapeAndPosition()` qui prend en argument `resultatIdentification` et qui, comme son nom l'indique, va calculer la forme et la position des blobs présents dans la liste.

— Elle vérifie si elle doit envoyer une `DefusionException` ou une `BordureException`.

— Elle retourne `resultatIdentification`.

Fusion de blobs

Les risques de fusion de blobs sont gérés au moment de l'ajout du blob dans `resultatIdentification`. Au lieu d'ajouter simplement le blob dans `resultatIdentification`, la fonction `addBlob` est utilisée. Cette méthode s'occupe de

vérifier que le blob qui doit être ajouté à la liste ne s'y trouve pas déjà pour une autre partie du corps. C'est-à-dire que si le blob portant l'identifiant i et représentant la partie du corps pc doit être ajouté à `resultatIdentification`, la méthode va vérifier s'il n'existe pas déjà un blob portant le même numéro identifiant i dans `resultatIdentification`. S'il en existe un, elle va simplement lui rajouter comme partie du corps pc. S'il n'en existe pas, elle va rajouter le blob i.

Dé-fusion d'un blob

Comme expliqué dans le chapitre précédent à la fin du point 5.1.3, lorsque des blobs doivent être dé-fusionnés, nous partons du principe que la tête ne déplace presque pas tout au long de la vidéo et que la main se trouvant à droite est la main droite et que celles se trouvant à gauche est la main gauche.

Lorsque un blob qui contenait la tête et une ou deux mains dé-fusionne, le blob qui contient le point `centreTete` est considéré comme la tête et le reste comme la ou les mains. Le point `centreTete` est calculé à chaque image où la tête n'est pas fusionnée comme étant le centre du blob représentant la tête.

Lorsqu'un blob qui contenait les deux mains dé-fusionne, les fenêtres des deux blobs résultants de la dé-fusion sont comparées. Si l'une se trouve plus à droite que l'autre, on attribuera au blob de la fenêtre de droite la main droite et à l'autre blob, la main gauche.

Cependant, ces suppositions sont quelque peu naïves et certains cas sont incertains. Par exemple si un blob "tête et main gauche" dé-fusionne mais qu'aucun des blobs ne contient `centreTete`. Lorsqu'un de ces cas incertains se présente, une exception est lancée. Elle sera rattrapée par la méthode `lireFrame()` qui, pour lever l'incertitude, n'a d'autre choix que de demander à l'utilisateur de donner la solution à l'aide d'une `helpWindow`.

Le fonctionnement d'une `helpWindow` est fort semblable au fonctionnement d'une `startWindow`. Il y a toujours trois cercles, un rouge, un bleu et un jaune. Cependant, les couleurs ne représentent pas une partie du corps en particulier. En effet, dans le cas d'une `helpWindow`, l'image posant problème est affichée ainsi que les blobs qui y sont identifiés comme sur la figure 6.11. Si ce qui est affiché est correct, l'utilisateur peut directement appuyer sur le bouton "Confirmer". Si, par contre, ce n'est pas correct, l'utilisateur peut déplacer les cercles et attribuer une (des) partie(s) du corps différente(s) aux cercles. De même que pour une `startWindow`, des boutons de type "radio" permettent d'indiquer si le blob est caché ou hors cadre. Une fois les réglages effectués, l'utilisateur doit cliquer sur le bouton "Tester" pour vérifier que les blobs sont bien identifiés, après quoi, il pourra confirmer. Si tous les blobs ne sont pas identifiés, un message d'erreur apparaîtra.

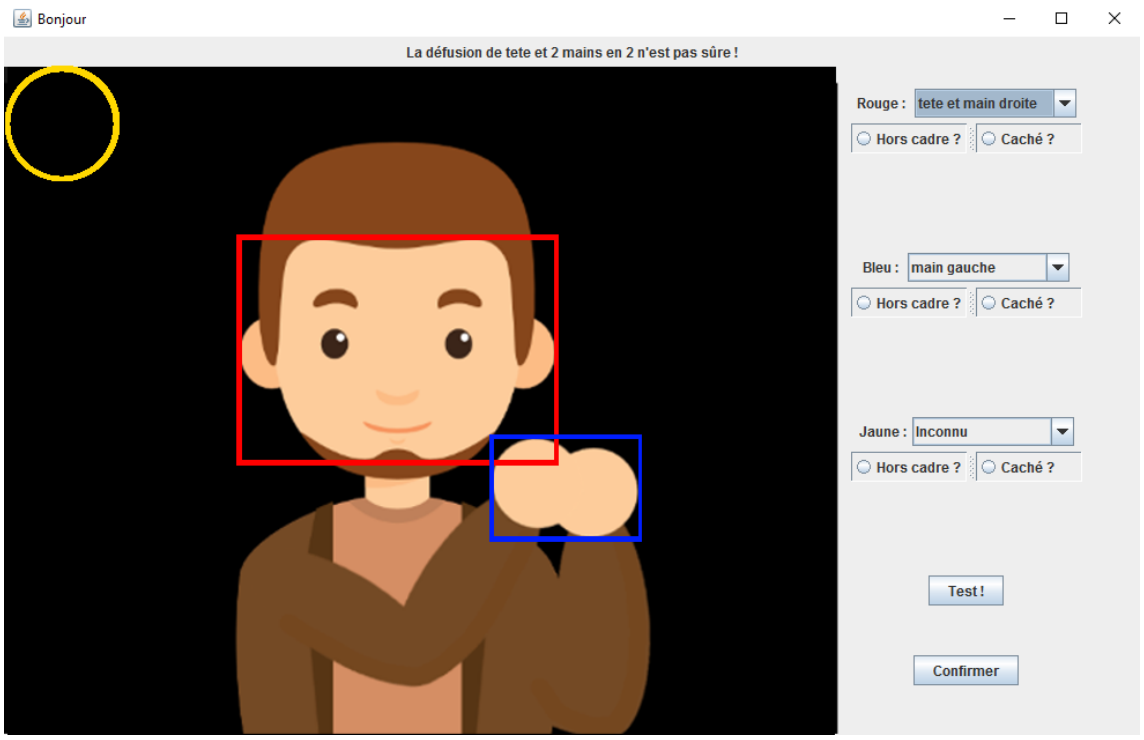


FIGURE 6.11 – Exemple de HelpWindow

Exceptions dues aux bordures

Au vu des risques de sortie de cadre énoncés à la section 4.1.2, certains problèmes peuvent apparaître, ce qui pourrait fausser le suivi. En effet, si, par exemple, le signeur fait un mouvement ample qui a pour effet de sortir ses mains du cadre, ces dernières pourraient rentrer dans le cadre loin de l'endroit de sortie. Le programme est conçu pour attendre que les mains rentrent dans les environs de la zone où elles sont sorties. Dans ce cas, il pourrait ne pas se rendre compte que les mains ont réapparu. Un autre cas qui pourrait poser problème est celui des mains au repos. Croisées sur les genoux, elles se trouvent souvent hors cadre. Ce qui pourrait poser problème pour le suivi est le fait que le blob représentant les deux mains sorte du cadre mais que les mains ne rentrent que l'une après l'autre. Le programme identifiera la main qui rentre comme étant les deux mains.

Pour éviter que ces deux problèmes ne se produisent, la méthode de suivi envoie une `BordureException` dans les cas suivants :

- si un blob est considéré hors cadre et qu'un blob ne correspondant à aucune partie de corps est en bordure.
- si un blob fusionné rentre dans le cadre de l'image.

Les exceptions ainsi envoyées seront traitées par la méthode `lireFrame()` qui gérera le conflit en affichant une `HelpWindow`.

6.3.4 Extraction des machines à états

La construction de la machine à états me d'un signe s est effectuée en calculant, pour chaque image, la forme et la position relative à la tête de chaque main. Un état e_i est alors construit grâce à ces informations. Ce nouvel état e_i est ajouté à la machine à états me si, et seulement si, la correspondance entre e_i et le dernier état de me, e_k, est inférieure à un certain seuil (le calcul de cette correspondance est détaillé au point 6.4.1). C'est-à-dire que e_i et e_k sont différents. Afin de tenir compte des occlusions possibles avec la tête, la valeur du seuil de correspondance d'un état e, est calculée comme suit :

- si aucune main ne fusionne avec la tête, la valeur du seuil est fixée à 0,8.
- si toutes les mains de l'état fusionnent avec la tête, les formes de celles-ci ne pourront être comparées efficacement, le seuil est alors fixé à 0,5.
- si une main fusionne et que l'autre non, le seuil correspond à la moyenne des deux valeurs, c'est-à-dire 0,65.

6.3.5 Extraction des mouvements

Les mouvements observés sont extraits en construisant une liste de points représentant les différentes positions prises par ce que nous appelons le centre de la main durant la séquence vidéo. La position de ce centre est estimée sur base de deux autres points de sorte que les éventuelles erreurs puissent être corrigées. En ne se basant que sur un point, les positions erronées ne pouvaient en effet pas être corrigées du au manque d'information.

Pour chaque image représentant un signe, les deux points servant à estimer la position du centre sont calculés comme suit :

- le point (x_1, y_1) correspond à la moyenne des positions des points situés dans l'enveloppe convexe de la main et dont la distance par rapport aux bords de celle-ci est maximale.
- le point (x_2, y_2) correspond à la moyenne des positions des points situés dans le contours de la main et dont la distance par rapport à celui-ci est maximale.

Les déplacements selon l'axe X de ces deux points, respectivement dx_1, et dx_2, sont ensuite calculés grâce à leurs positions dans l'image précédente. Si dx_1 est deux fois supérieur à dx_2 (ou inversement), dx_1 est considéré comme non pertinent. La position en X du point (x_1, y_1) est alors corrigée sur base de sa position en X dans l'image précédente et du déplacement dx_2. Le même procédé de correction est ensuite appliqué aux déplacements selon l'axe Y. Le centre de la main est finalement calculé en réalisant la moyenne des coordonnées de (x_1, y_1) et de (x_2, y_2).

Le calcul du centre ayant besoin de mémoriser certaines informations lui étant propres, nous avons décidé d'encapsuler celui-ci dans une classe (`MouvementIdentifier`). Cette classe contient donc, outre les méthodes de calcul du point, deux attributs `Point` représentant les dernières positions des points (x_1, y_1) et (x_2, y_2). Ces attributs sont réinitialisés entre le traitement de chaque signe. De cette façon, les cas pour lesquels les positions précédentes ne sont pas pertinentes (comme les coupures) peuvent être gérés aisément. De plus, la propagation d'erreurs entre les signes est réduite.

La figure 6.12 illustre le résultat du calcul du centre de la main. Le point rouge représente le point (x_1, y_1) , le point bleu représente le point (x_2, y_2). Le point rose représente quant à lui le centre calculé de la main. S'il ne représente pas exactement le centre de la main, sa position reste plus ou moins fixe ce qui reflète bien le faible mouvement réalisé par la main. Sur la seconde image, les coordonnées du point (x_2, y_2) ont subi un changement important malgré la quasi immobilité de la main. Les coordonnées du centre ne sont cependant pas impactées par cette erreur.

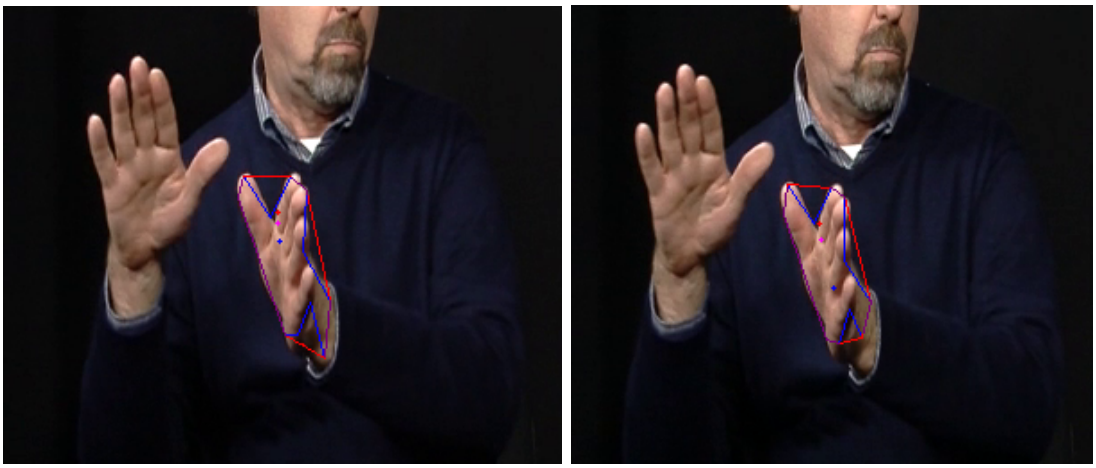


FIGURE 6.12 – Illustration du résultat du calcul du centre de la main (images originales provenant de [21]).

6.4 Comparaison des caractéristiques

6.4.1 Comparer les machines à états

La comparaison entre deux machines à états s'effectue en comparant les états de celles-ci. Avant d'expliquer le fonctionnement de la comparaison entre machines, il est donc nécessaire de bien comprendre comment sont comparés les états.

La correspondance entre deux états e et e' est calculée comme la moyenne des correspondances des mains pour lesquelles à la fois e et e' possèdent des informations. Bien que les états construits depuis les images observées possèdent toujours des informations pour les deux mains, cela n'est pas vrai pour les états des machines de la base de connaissances. En effet, comme nous le détaillerons par la suite, les états de ces machines ne disposent que des informations des mains réellement impliquées dans le signe. La comparaison d'états doit donc tenir compte de ce fait afin que la reconnaissance des signes puisse se faire de manière aussi correcte que possible. La correspondance entre deux mains m et m' est calculée en comparant leurs forme et position respectives.

Cette correspondance des mains est calculée différemment en fonction des éventuelles fusions avec la tête.

— Dans le cas où ni la forme de m ni celle de m' n'implique de fusion avec la tête, la valeur de correspondance entre m et m' est la moyenne pondérée des correspondances entre leurs formes et leurs positions. La correspondance entre les formes se voit attribuer un poids supérieure à la correspondance des positions relatives car, selon nous, la forme véhicule plus d'informations que la position relative.
— Si la forme de m et celle de m' impliquent toutes les deux une fusion avec la tête la correspondance ne peut être définie efficacement. En effet, les formes ne peuvent être jugées pertinentes (car incluant la forme de la tête) et, les positions relatives vaudront "Tête". La valeur de correspondance entre m et m' est alors fixée par défaut à 0,5.
— Enfin, si seule la forme de m (ou de m') implique une fusion avec la tête, la correspondance entre m et m' est considérée nulle et vaut donc 0.

La méthode effectuant la comparaison entre machines procède comme suit : Soit deux machines à états me et me' possédant respectivement les listes d'états $[e_0, ..., e_i]$ et $[e'_0, ..., e'_k]$. La comparaison commence avec les états e_0 et e'_0 et procède de la façon suivante jusqu'à avoir traité les états e_i et $e'k$ ou jusqu'à ce que le nombre d'états distincts successifs dépasse un certain seuil (défini à 5).

— Tant que les états e_a et e'_b sont identiques et que me et me' possèdent un état suivant : la méthode avance d'un pas dans me et me' et le score de correspondance s_i entre e_a et e'_b est additionné à une valeur de correspondance globale $matchTotal$.
— Lorsque e_a et e'_b ne correspondent plus, ou que l'une des deux machines ne possède pas d'état suivant, nous procédons de la façon suivante. Si le nombre d'états différents successifs est inférieur à 5, une vérification a lieu pour déterminer si les machines possèdent ou non un état suivant.
  — Pour chaque machine possédant un état suivant, une nouvelle branche de la comparaison est créée et répète les étapes explicitées ici en ayant au préalable avancé d'un pas dans cette machine. Par exemple, si me possède un état suivant, la comparaison continue en considérant les états e_{a+1} et e'_b.
  — Si aucune machine ne possède d'état suivant, la valeur de correspondance de la branche courante $s(machine)_i$ est calculée en divisant la valeur de $matchTotal$ par le nombre de comparaisons d'états réalisées dans la branche courante.
— Si les états e_a et e'_b ne correspondent pas (ou que l'une des deux machines ne possède pas d'état suivant) et que le nombre d'états différents successifs est supérieur à 5, la branche de comparaison courante est interrompue.
— Finalement, la correspondance entre me et me' est égale à la valeur $s(machine)_i$ maximale calculée parmi toutes les branches. me et me' seront jugées identiques si, et seulement si, leur correspondance est supérieure à un seuil calculé en réalisant la moyenne des seuils de correspondance des états de me.

L'exécution des différentes branches de comparaison est réalisée à l'aide d'un pool de threads grâce aux classes de la librairie `concurrent` de Java. Les différentes branches s'exécutent ainsi en parallèle, ce qui permet de réduire le temps d'exécution nécessaire. En outre, la méthode de comparaison à été conçue récursivement de sorte que les branches non terminales ne doivent pas attendre la fin de l'exécution des branches terminales avant de se

terminer.

Une dernière chose à noter est l'utilisation d'un objet de type `Phaser` afin de s'assurer de la terminaison de toutes les branches avant de continuer l'exécution du programme ainsi que l'utilisation d'un type "MutableDouble" conçu pour gérer les accès concurrents à la valeur de correspondance entre les machines.

6.4.2 Comparer les mouvements

Pour rappel, nous utilisons la méthode du DTW pour calculer la correspondance entre les mouvements. Le DTW nécessite deux phases : le calcul de la matrice des coûts cumulés et le calcul du chemin le moins cher (correspondance optimale). Cependant, dans notre cas, la seconde phase n'a que peu d'intérêt. En effet, pour la comparaison de mouvements, seul le coût de la correspondance importe puisque plus ce coût en bas, plus les mouvements sont semblables. Ce coût correspondra toujours à la valeur se trouvant à la dernière ligne et à la dernière colonne de la matrice des coûts cumulés.

Avant de calculer cette matrice des coûts cumulés, les mouvements sont réalignés afin de comparer ces derniers indépendamment de leur position de départ dans l'espace. Un mouvement est représenté comme une liste de points (x,y). Pour réaligner deux mouvements, les valeurs de x et de y moyennes des deux mouvements $M1$ et $M2$ sont calculées. Ensuite, les marges en x et en y sont calculées :

$$marge_x = moyenne_x M1 - moyenne_x M2$$

$$marge_y = moyenne_y M1 - moyenne_y M2$$

Le second mouvement est alors aligné sur le premier en soustrayant la valeur de $marge_x$ à la valeur x de tous ses point, et la valeur de $marge_y$ à la valeur y de ses points.

La matrice des coûts cumulés est ensuite calculée en déterminant la valeur de chaque élément (i, j). Cette valeur correspond à la somme de la valeur minimale des cellules précédentes et adjacentes et de la distance entre le point (i) du premier mouvement et le point (j) du second mouvement. Le code suivant permet de mieux se représenter la construction de cette matrice.

```
1  double[][]coutCumule= new double[M1.length][M2.length];
2
3      //(0,0)
4      coutCumule[0][0]=distance(M1[0],M2[0]);
5      //for i=0
6      for(int j=1;j<M2.length;j++) {
7          coutCumule[0][j]=(distance(M1[0],G2[j])
8                                  +coutCumule[0][j-1]);
9      }
10     //for j=0
11     for(int i=1;i<M1.length;i++) {
```

```
12          coutCumule[i][0]=(distance(M1[i],M2[0])
13                                  +coutCumule[i-1][0]);
14      }
15      //for i,j
16      for(int i=1;i<M1.length;i++) {
17          for(int j=1;j<M2.length;j++) {
18              coutCumule[i][j]=(distance(M1[i],M2[j])
19                                  +Math.min(coutCumule[i-1][j],
20                                   Math.min(coutCumule[i][j-1],
21                                   coutCumule[i-1][j-1])));
22          }
23      }
```

La méthode renvoie ensuite le coût de correspondance qui est la valeur se trouvant à la dernière ligne et à la dernière colonne de la matrice des coûts cumulés.

Pour finir, il nous a fallu déterminer à partir de quel seuil nous pouvions considérer que les mouvements étaient semblables ou non. Ce seuil a été choisi de manière empirique. Nous avons déterminé que deux mouvements étaient quasi identiques si leur coût de correspondance était inférieure à 100 et qu'ils étaient comparables si leur coût de correspondance était inférieure à 500.

6.5 Apprentissage

La phase d'apprentissage est réalisée grâce aux fichiers de caractéristiques produits par le module d'extraction. Le traitement d'un tel fichier commence par la création d'une liste d'objets `Track` (représentant chaque entrée du fichier) grâce à la classe `XmlTrack` contenant les méthodes de lecture et d'écriture des ces fichiers. Ensuite, pour chaque objet `Track` extrait du fichier, une `Signature` est construite puis ajoutée à la base de connaissances. Cet objet reprend :

- le tag du signe ;
- les mains impliquées dans la réalisation du signe ("MG","MD","MM") ;
- la machine à états du signe ;
- les mouvements des mains impliquées dans le signe.

L'ajout des objets de type `Signature` est effectué grâce aux méthodes de la classe `XmlSignes`. Cette dernière contient les méthodes de lecture et d'écriture du fichier Signe.xml. D'abord, la liste des signatures de la base de connaissances représentant le même signe que la signature à ajouter est récupérée. Ensuite, les informations de la signature en cours de traitement sont comparées avec celles des signatures de la liste.

- Si cette liste est vide, cela signifie que le signe n'est pas encore présent dans la base de connaissances. Une nouvelle entrée est alors créée pour celui-ci avec comme signature, la signature à ajouter.

— Si la liste n'est pas vide, les mouvements et la machine dont le résultat de la comparaison est minimal et inférieur aux différents seuils de correspondance sont déterminés parmi les mouvements et machines des signatures de la liste.
  — Si les mouvements et la machine ainsi trouvés appartiennent tous à la même signature, le nombre d'occurrences de celle-ci est simplement incrémenté de un.
  — Si les mouvements et la machine trouvés n'appartiennent pas à la même signature, une nouvelle entrée "signature" est ajoutée à la base de connaissances pour le signe correspondant.

6.6 Reconnaissance

Comme expliqué au point 5.4, la reconnaissance se fait en deux parties : la réduction de l'ensemble des signatures en ensemble des signatures potentielles puis la sélection du signe le plus probable. Deux méthodes de sélection différentes ont été testées.

6.6.1 Réduction de l'ensemble des signatures potentielles

La réduction de l'ensemble des signatures potentielles commence par la récupération de l'ensemble des signatures de la base de connaissances grâce à la classe `XmlSignes`.

La liste des signatures ainsi récupérée est ensuite filtrée sur base de la caractéristique de mouvement. Pour chaque signature de cette liste, les mouvements des mains gauche et droite sont comparés avec les mouvements du signe observé. Si le résultat de l'une de ces comparaisons est supérieur à 500, la signature est éliminée des signatures potentielles. Cependant, afin de gérer le fait que certaines signatures de la base de connaissances ne possèdent des informations de mouvements que pour une seule des deux mains, un mouvement `null` est considéré comme identique à celui de l'observation.

La liste est finalement filtrée une seconde fois afin d'éliminer les signatures dont la machine à états n'est pas jugée identique à celle du signe observé.

6.6.2 Sélection du signe le plus probable

Une fois filtrée totalement, la liste des signatures potentielles est traitée pour assigner à chaque signature (ou signe selon la méthode utilisée) une probabilité de correspondre à l'observation.

Première méthode

Avec cette première méthode, la liste des signatures potentielles est parcourue et, pour chacune, une probabilité de correspondance avec le signe observé est calculée. La formule pour déterminer cette probabilité utilise le score de correspondance des machines à états et le pondère en fonction du nombre d'occurrences et du fait que le signe soit fait à deux

mains plutôt qu'une. La formule exacte de calcul de la probabilité qu'un signe observé S corresponde à une signature S' est :

$$P(S = S') = correspondanceEtat(S, S') * (1 + 0.01 * S'.nbOccurrence) * \beta$$

où $correspondanceEtat(S, S')$ est le taux de correspondance des machines à états, $S'.nbOccurence$ est le nombre d'occurrences de la signature et β vaut 1.05 si la signature est réalisée à deux mains et un si elle est réalisée à une main. Pour chaque signature ainsi traitée, un objet de type `Prediction` est crée. Ce type d'objet encapsule d'une part l'objet `Signature` traité et, d'autre part, la probabilité $P(S = S')$ ainsi que les valeurs de DTW et de correspondance de machines calculées. Une fois toutes les signatures traitées, la liste des objets `Prediction` est triée par ordre décroissant de $P(S = S')$ et finalement retournée.

Deuxième méthode

Avant d'aborder le fonctionnement de la seconde méthode de calcul de probabilité, le terme "signe distinct" (dans le contexte de cette méthode) doit être défini car il s'agit d'un élément important de la méthode. Par signe distinct, nous entendons les signes possédant un tag différent et dont les mains impliquées dans la réalisation du signe sont différentes. Par exemple, un tag t réalisé une fois avec la main gauche et une seconde fois avec la main droite engendrera deux signes distincts. Nous avons procédé de la sorte afin d'être en mesure d'associer chaque prédiction produite à une (ou deux) main(s) et donc être en mesure de produire le fichier d'annotations correctement. Maintenant que le terme est clairement défini, le fonctionnement de la méthode peut être abordé. Cette dernière procède comme suit.

La liste des signatures potentielles filtrée est parcourue signature par signature. Parallèlement à ce parcours, une `Hashmap` est construite. Elle associe une valeur numérique, appelée `scoreSigne`, à chaque signe distinct rencontré. Pour chaque signature de la liste, une vérification a lieu afin de déterminer si le tag de celle-ci a déjà été traité, c'est-à-dire qu'il est déjà présent dans la `Hashmap`. Si c'est le cas, la signature est simplement ignorée. Si ce n'est pas le cas, la valeur de `scoreSigne` correspondant au tag est calculée. Ce calcul correspond au produit du nombre d'occurrences du tag parmi les signatures de la liste et du nombre d'occurrences de ce tag dans la base de connaissances divisé par le nombre d'occurrences de tous les signes de la base.

$$scoreSigne(s) = (occurrenceBase(s)/nbTotalOccurrences) * occurrencesList$$

Après le parcours de la liste complète, la `Hashmap` créée est traitée pour construire une liste `resultat` d'objets de type `PredictionSigneGlobal`. La classe `PredictionSigneGlobal` permet simplement d'encapsuler le tag et la valeur de `scoreSigne` associée à ce tag. La liste `resultat` construite est finalement triée par ordre décroissant de `scoreSigne` avant d'être retournée.

Chapitre 7

Évaluation

7.1 Méthode d'évaluation

L'évaluation de la solution a été réalisée en limitant un maximum les biais extérieurs. En effet, à la section 4.1, quelques problèmes pouvant nuire au bon déroulement de la reconnaissance ont été présentés. Pour rappel, voici les points qui pouvaient poser problème :

- les occlusions ;
- les sorties de cadre ;
- les angles de vue ;
- la présence du second signeur dans le champ ;
- l'éclairage non-uniforme ;
- l'absence de contrainte sur l'habillage.

L'impact de l'éclairage et des sorties de cadre a pu être réduit. Cependant, les solutions trouvées ne sont pas optimales et des soucis de détection et de suivi persistent. Concernant les autres problèmes, aucune solution efficace n'a pu être trouvée soit parce que, à notre connaissance, aucune solution n'existe actuellement (pour les problèmes d'angle de vue par exemple), soit parce que de telles solutions demandaient niveau d'expertise dans différents domaines (mathématique, physique...) qu'il nous était impossible d'atteindre en quatre mois. De plus, ces méthodes ne fonctionnent généralement que dans un environnement relativement contrôlé. Dans l'optique de vérifier la faisabilité d'une solution de traitement automatique des vidéos du corpus LSFB, nous avons donc choisi de tester les vidéos qui présentaient le moins de problèmes possibles. Si les résultats s'avèrent concluants, les autres vidéos pourront également être testées. Dans le cas contraire, il ne servirait à rien de tester les vidéos de mauvaise qualité.

Dans un premier temps, les signeurs permettant d'éviter le plus possible ces problèmes ont été sélectionnés. Ces signeurs sont ceux possédant des manches longues, un décolleté faible et des cheveux sombres. Comme expliqué au point 4.1, les vêtements laissant paraître beaucoup de peau sont problématiques car un grand nombre de zones de peau découverte peut gêner la reconnaissance. Pour ce qui concerne les cheveux clairs, leur teinte se confondait parfois avec la teinte de la peau, ce qui avait tendance à induire en erreur la détection. Les problèmes

d'angles de vue et de luminosité étant présents quel que soit le signeur, ceux-ci n'ont pas été pris en compte dans la sélection.

Ensuite, les sessions dont les deux signeurs correspondaient à ces critères ont été sélectionnées. La solution mise en place permettant de traiter deux vidéos d'une même discussion en même temps, il était plus intéressant d'extraire les caractéristiques de deux vidéos simultanément plutôt que d'une seule à la fois. De cette manière, un plus large ensemble de données pouvait être extrait pour un temps de travail identique.

Enfin, parmi ces sessions, seules celles disposant de discussions annotées ont été conservées. Plusieurs raisons expliquent ce choix. D'une part, les vidéos utilisées pour construire la base de connaissances doivent disposer d'annotations. D'autre part, du fait de la charge de travail et du temps limité, le processus de reconnaissance fait l'hypothèse que les signes présents dans les vidéos sont déjà segmentés. Cette segmentation étant déjà présente dans les fichiers d'annotations, il est logique d'utiliser ceux-ci. Enfin, pour réaliser l'évaluation de la solution, les résultats produits doivent pouvoir être validés. Dès lors, les annotations des vidéos analysées doivent être connues.

Seules cinq sessions remplissaient toutes ces conditions : les sessions 21, 22, 23, 32 et 40. Parmi celles-ci, nous avons décidé de travailler avec les sessions 21 et 23 qui contenaient respectivement huit et dix tâches annotées. Les autres sessions ne possédaient que une ou deux tâches annotées. Le temps nécessaire à l'étape d'extraction étant relativement long, nous avons choisi de ne pas traiter toutes les vidéos de ces sessions mais uniquement un total d'environ 1h de vidéo, à savoir trente minutes de chaque session. Le tableau 7.1 reprend les tâches ayant été sélectionnées pour chaque session.

Session	Tâche	Durée
21	02	1 :32
	03	9 :05
	04	7 :12
	11	3 :14
	12	7 :59
	14	6 :00
23	03	9 :40
	05	3 :42
	06	6 :53
	13	10 :23
Total		1 :05 :40

TABLE 7.1 – Tableau récapitulatif des vidéos ayant servit à l'évaluation.

Pour construire la base de connaissances, un ensemble d'entraînement a été sélectionné. Pour cela, nous avons choisi d'utiliser environ 2/3 des données extraites, soit environ 40 minutes de vidéo. En outre, nous avons décidé de réaliser deux séries de tests avec chacune

un ensemble d'entraînements différent. Disposant d'un ensemble de données limité, nous avons jugé que cela réduirait les risques de tomber sur un cas particulier qui aurait engendré des résultats très bons ou très mauvais. Selon nous, si les deux jeux de tests produisent les mêmes résultats, il est fort probable que cela reflète les résultats réels de la solution. Nous avons également choisi les ensembles d'entraînement de sorte qu'ils contiennent environ vingt minutes de chaque session. Ce choix a été posé afin que les différents signeurs soient représentés de manière identique dans les données d'entraînement. Les ensembles ainsi choisis sont ceux repris par les tableaux 7.2.

Ensemble d'entraînement 1		
Session	Tâche	Durée
21	02	1 :32
	03	9 :05
	04	7 :12
	11	3 :14
23	05	3 :42
	06	6 :53
	13	10 :23
Total		41 :59

Ensemble d'entraînement 2		
Session	Tâche	Durée
21	04	7 :12
	12	7 :59
	14	6 :00
23	03	9 :40
	13	10 :23
Total		41 :14

TABLE 7.2 – Tableaux récapitulatifs des vidéos utilisées pour les ensembles d'entraînement

Le tiers de données restant (soit vingt minutes de vidéo) a été utilisé pour évaluer la reconnaissance. Les deux méthodes présentées au point 6.6 ont été testées. Dans les deux cas, seuls les trois résultats les plus probables ont été pris en compte et triés pas ordre décroissant de ressemblance. Ce choix de considérer trois résultats a été pris afin de pouvoir mieux tenir compte du fait que certains signes se ressemblent fortement et peuvent donc, pour une observation donnée, avoir des probabilités très proches. Nous nous sommes cependant limités à trois résultats, car, au-delà, nous avons jugé que cela constituait trop de solutions possibles et que la reconnaissance n'était donc pas fiable. Pour terminer, les résultats produits ont été comparés avec les annotations manuelles présentes dans les fichiers ELAN.

7.2 Résultats

Avant de parler plus en détails des résultats obtenus, il est nécessaire de considérer la configuration des machines sur lesquelles les tests ont été réalisés. En effet, cette configuration joue un rôle sur le temps nécessaire à l'exécution des différentes étapes de notre solution. Connaître celle-ci permet donc de mettre en perspective les temps qui seront donnés par la suite. Les machines utilisées sont des machines portables de configuration moyenne. La première possède 6go de ram, un processeur "2-cores" i5-3337U, et un disque dur de type SSD. La seconde possède 4go de ram, un processeur "2-cores" i3-380M et un disque dur de type HDD.

Pour ce qui est du temps d'exécution, de nets progrès ont pus être observés. Pour rappel, l'annotation manuelle nécessitait environ 8 heures de travail de la part d'une personne bilingue pour annoter 2 minutes de vidéo. La phase d'extraction de notre solution nécessite quant à elle environ 1h30 pour 2 minutes de vidéo. Évidemment, cette phase ne couvre pas toute l'annotation de la vidéo. Il faut ainsi rajouté à ce temps d'extraction environ 20 minutes pour une vidéo de durée moyenne (comportant environ 600 signes) pour la phase de reconnaissance. Il faut également considérer le temps nécessaire à la construction de la base de connaissances (bien que celle-ci ne doit pas être construite à chaque vidéo traitée). Pour nos tests, la construction de cette base à l'aide de 40 minutes de vidéo à nécessité environ 90 minutes. Au final, le temps nécessaire au traitement d'une vidéo avec notre solution est bien inférieur à celui nécessaire pour réaliser une annotation manuelle. De plus, là où une annotation manuelle nécessite qu'un annotateur bilingue reste concentré sur l'annotation du début à la fin, notre solution ne nécessite une intervention humaine que pour la phase d'extraction. La personne réalisant cette intervention ne doit pas nécessairement comprendre la langue des signes et la charge de travail qu'il doit fournir est bien moindre puisqu'il ne doit prêter attention à l'annotation que lorsque le programme lui demande son avis.

Les résultats montrent que les deux méthodes testées ne reconnaissent respectivement que 5% et 9% de l'ensemble des signes présents dans les vidéos (en ne considérant que les trois propositions les plus probables), ce qui n'est malheureusement pas très satisfaisant et l'objectif des 70% évoqué au point 4.3 est loin d'être atteint. Cependant, ces chiffres sont à nuancer. En effet, il s'avère qu'en moyenne, les discussions analysées contiennent 662 signes dont un peu moins du tiers ne se trouve pas dans notre base de connaissances. Cela peut s'expliquer par la petite taille de celle-ci et la rareté de certains signes. Dès lors, il serait plus correct de ne considérer que les signes présents dans la base de connaissances. Le taux de reconnaissance moyen des deux méthodes monte alors respectivement à 7.54% et 12.29% ce qui reste insuffisant. Les tableaux 7.2 et 7.4 reprennent les résultats obtenus suite aux tests réalisés. Il est à noter que les pourcentages présentés dans ces tableaux sont calculés en fonction du nombres de signes présents dans la base de connaissance (Nb signes en BC).

	Ensemble d'entraînement 1						Moyenne	
Tâche	2112		2114		2304			
Nb signes au total Nb signes en BC	612 401		761 550		1219 864		864 605	
Méthode	1	2	1	2	1	2	1	2
1ère correspondance	6	16	14	40	13	50	11	35
%	1.50%	3.99%	2.55%	7.27%	1.50%	5.79%	1.85%	5.68%
2nde correspondance	9	13	19	33	12	29	13	25
%	2.24%	3.24%	3.45%	6.00%	1.39%	3.36%	2.36%	4.20%
3ème correspondance	6	8	25	24	15	15	15	16
%	1.50%	2.00%	4.55%	4.36%	1.74%	1.74%	2.59%	2.70%
Signes reconnus	21	37	58	97	40	94	40	76
%	5.24%	9.23%	10.55%	17.64%	4.63%	10.8%	6.80%	12.58%
Signes non reconnus	380	364	492	453	824	700	565	529
%	94.16%	90.77%	89.45%	82.36%	95.37%	89.12%	93.20%	87.42%

TABLE 7.3 – Résultat de la reconnaissance avec l'ensemble d'entraînement 1

TABLE 7.4 – Résultat de la reconnaissance avec l'ensemble d'entraînement 2

	Ensemble d'entraînement 2										Moyenne	
Tâche	2102		2103		2111		2305		2306			
Nb signes au total	167		986		270		402		796		520	
Nb signes en BC	121		643		173		290		566		358	
Méthode	1	2	1	2	1	2	1	2	1	2	1	2
1ère correspondance	4	16	19	42	2	6	10	18	15	27	10	22
%	3.31%	13.22%	3.95%	6.53%	1.16%	3.47%	3.45%	6.21%	2.65%	4.77%	2.70%	6.84%
2ndè correspondance	2	5	23	24	2	4	7	7	13	14	9	11
%	1.65%	4.13%	3.58%	3.73%	1.16%	2.31%	2.41%	2.41%	2.30%	2.47%	2.22%	3.01%
3ème correspondance	6	4	21	16	2	2	8	8	18	9	11	8
%	4.96%	3.31%	3.27%	2.49%	1.16%	1.16%	2.76%	2.76%	3.18%	1.59%	3.06%	2.26%
Signes reconnus	12	25	63	82	6	12	25	33	46	50	30	40
%	9.92%	20.66%	9.80%	12.75%	3.47%	6.94%	8.62%	11.38%	8.13%	8.83%	7.99%	12.11%
Signes non reconnus	109	96	580	561	167	161	265	257	520	516	328	318
%	90.08%	79.34%	90.20%	87.25%	96.53%	93.06%	91.38%	88.62%	91.87%	91.17%	92.01%	87.89%

Au vu de ces chiffres, une observation intéressante concerne la performance des deux méthodes testées. Pour rappel, la première méthode (colonnes '1' dans les tableaux) avait pour objectif d'attribuer une probabilité à chaque signature de chaque signe. Elle retournait donc une liste de signatures potentielles. La seconde méthode quant à elle attribuait directement une probabilité aux différents signes, retournant donc une liste de signes potentiels. En nous basant sur les résultats de nos tests, il est possible d'affirmer que la seconde méthode fournit de meilleurs résultats. En effet, le taux de reconnaissance moyen de la première méthode est d'environ 7,54% avec un taux maximal de 10,55% tandis que le taux moyen de la seconde méthode atteint 12,29% avec un taux maximal d'environ 20%.

7.3 Explication des résultats

Selon nous, plusieurs facteurs peuvent expliquer le taux assez faible de reconnaissance observé. Par ordre d'impact décroissant, ces facteurs sont les suivants.

Les problèmes de détection et de suivi dus aux occlusions ayant lieu durant les discussions sont probablement une cause majeure. En effet, en analysant les ensembles d'entraînement, nous avons pu constater qu'environ 70% des signatures observées impliquaient au moins une occlusion avec la tête durant leur exécution. Si ces occlusions n'impliquent pas systématiquement une mauvaise reconnaissance de la signature, il est indéniable que cela impacte le taux de correspondance entre deux occurrences de cette signature. Dans le même ordre d'idée, les erreurs de détection de la forme de la main pouvant survenir en raison de l'éclairage sont très probablement aussi la cause d'une diminution du taux de détection. En effet, deux signes réalisés de manière absolument identique pourraient ne pas être reconnus si les formes observées pour l'un sont déformées en raison d'un éclairage incorrect. Dans les vidéos, l'impact de l'éclairage change selon la manière dont se tient le signeur, ce scénario peut donc arriver.

Les problèmes de confusion entre signes en raison de leur ressemblance sont sans doute la deuxième cause principale du taux de reconnaissance faible. Comme nous l'avons vu au point 4.1.3, certains signes sont fort semblables du point de vue de leur configuration et de leur mouvement. Dans des conditions normales, la distinction entre les deux peut être faite assez simplement. Cependant, dans le cas des vidéos du LSFB, cette distinction est rendue très complexe principalement en raison de l'angle de vue choisi pour le tournage. Cela est d'autant plus complexe que cet angle de vue varie en fonction de la façon dont le signeur s'oriente sur sa chaise. D'autres facteurs limitant cette distinction sont les problèmes de détection et d'occlusion. En effet, une mauvaise détection peut avoir pour conséquence que les informations de formes extraites des images ne soient pas correctes et fassent penser à un signe autre que celui réellement réalisé. De même, les occlusions peuvent fausser le pourcentage de correspondance entre deux machines à états en raison de l'impossibilité de comparer des formes provenant d'une occlusion avec la tête.

Certaines erreurs peuvent également provenir de soucis dans l'implémentation. Durant les différentes étapes, certains seuils interviennent (seuil de correspondance, d'appartenance a une classe de pixels, etc.). Les valeurs de ces seuils ont fait l'objet de tests avant d'être

définies. Cependant, il n'est pas à exclure que certaines de ces valeurs ne soient pas optimales. En outre, nous ne possédions pas de connaissance particulière en matière de reconnaissance gestuelle avant de commencer notre stage. Ayant du acquérir et mettre en pratique celles-ci dans des délais assez courts, nous n'avons pas la prétention de considérer notre solution comme parfaite.

Enfin, le fait que la base de connaissance soit relativement petite impacte également la reconnaissance. En effet, les 40 minutes de vidéo sélectionnées ne contiennent pas un échantillon de signes suffisant. En moyenne, un tiers des signes présents dans une vidéo ne sont pas présents dans la base de connaissance. Cela pourrait s'expliquer principalement par la structure des vidéos. En effet, chaque vidéo représente une tâche particulière. Or, chaque tâche correspond à un sujet de discussion particulier. Il peut donc arriver que certains signes n'apparaissent que dans certaines tâches. Il est donc normal que certains signes n'apparaissent pas dans la base de connaissances. Auquel cas, il est normal qu'ils n'aient pas été reconnus durant nos tests. Toutefois, cela n'est probablement pas la cause majeure de l'imprécision de la détection. En effet, comme expliqué au point précédent, le taux de reconnaissance reste très faible même en ne considérant que les signes présents dans la base de connaissances. Cela pourrait s'expliquer par le fait que ces signes possèdent un grand nombre de signatures différentes.

7.4 Évaluation vis-à-vis des exigences

Au regard des exigences énumérées au point 4.3, nous ne pouvons pas considérer la solution comme satisfaisante.

Concernant l'accélération du processus d'annotation, le temps nécessaire à la production du fichier a bel et bien été réduit. Cependant, comme l'exigence de reconnaissance des signes est très loin d'être remplie, nous ne pouvons considérer l'exigence comme atteinte. En effet, produire un fichier d'annotations en très peu de temps n'a pas d'intérêt si celui-ci doit subir de très nombreuses corrections de la part des annotateurs.

L'exigence de production d'un fichier compatible avec ELAN, n'a pas non plus été atteinte. Cependant, il s'agit ici d'un choix. Nous n'avons pas jugé utile de produire un fichier respectant la structure des fichiers .eaf sans savoir si l'objectif de reconnaissance allait être atteint. Ainsi, au lieu de produire un fichier .eaf, nous avons décidé de produire un fichier possédant une structure nous permettant de vérifier le plus facilement possible les résultats obtenus. L'ajout de la production d'un fichier compatible ELAN n'est cependant pas une tâche complexe et aurait pu être ajoutée dans le cas où les résultats obtenus étaient positifs.

Finalement la dernière exigence, à savoir l'utilisation possible par une personne non bilingue, a bien été atteinte. Il ne s'agit cependant pas de l'exigence la plus importante.

7.5 Pistes d'amélioration

Plusieurs pistes pourraient être explorées dans le but d'augmenter l'efficacité du programme. Certaines de ces pistes nécessiteraient cependant plusieurs mois de recherche à elles seules. C'est essentiellement pour cette raison que celles-ci n'ont pas été explorées, du moins en profondeur, durant le développement de la solution initiale.

Dans un premier temps, les problèmes d'occlusions seraient intéressants à résoudre car ceux-ci diminuent fortement l'efficacité de la comparaison. En effet, il est impossible à ce stade de comparer efficacement les caractéristiques de deux mains qui se trouvent devant le visage. De ce fait, la comparaison de deux signes est parfois approximative ce qui résulte en une reconnaissance moins précise. Par exemple, dans le cas extrême où deux signes différents sont réalisés exclusivement devant le visage, ceux-ci seront considérés comme identiques car aucune information ne peut-être donnée quant à leur réelle similitude. La résolution des occlusions constitue cependant un champ de recherche à part entière. Si des méthodes ont été proposées pour résoudre ces problèmes d'occlusions, en général, celles-ci ne fonctionnent que dans certains cas. Paul Smith & Al [29] proposent par exemple une solution basée sur les champs de force d'une image. Cela produit cependant des résultats approximatifs lorsque la personne est trop de profil.

Ensuite, les problèmes de détection et de suivi dus à l'éclairage, à l'absence de contrainte d'habillage ou à l'apparition du second signeur dans le champs, mériteraient une attention particulière. En effet, ces problèmes empêchent dans certains cas l'acquisition de la forme complète de la main ou le bon suivi de celle-ci. Les caractéristiques extraites sont alors erronées, ce qui engendre par la suite une reconnaissance moins précise.

De plus, comme expliqué à la section 5.1, tous les paramètres des signes n'ont pas été utilisés. Dès lors, il serait peut-être intéressant de rajouter ceux ayant été omis, comme les expressions du visage. D'autres paramètres que les paramètres d'un signe évoqués au point 1.2.2 pourraient également être ajoutés. Par exemple, la probabilité qu'un signe soit suivi par un autre pourrait servir à pondérer les résultats de la phase de reconnaissance. L'ajout de ces paramètres pourrait permettre de résoudre certains problèmes d'ambiguïté entre signatures. Toutefois, sans l'ajout des deux améliorations précédentes, il est probable que le gain de précision ne soit que minime. En outre, dans le cas où le dernier signe observé est considéré, il serait également important d'éviter la propagation des erreurs, sans quoi, la précision de la reconnaissance pourrait être pire qu'actuellement.

Enfin, tout au long du développement, beaucoup de valeurs ont été définies empiriquement. Pour chacune, nous avons réalisé une série de tests sur un petit ensemble de données. Les résultats ainsi obtenus ont ensuite été analysés afin de déterminer la valeur reflétant le mieux notre vision de la correspondance entre les éléments de l'ensemble utilisé. Dans le cas des valeurs nécessaires aux différents calculs (marge d'erreur du nombre d'angles, distance maximale entre deux angles, etc.) nous avons sélectionné les valeurs en fonction de leur impact sur les résultats de ces calculs. Pour les seuils de correspondance, nous les avons déterminés en analysant les résultats des différentes comparaisons et en sélectionnant

les valeurs permettant de faire correspondre un maximum les éléments que nous jugions similaires sans pour autant faire correspondre des éléments que nous jugions différents. Cette façon de procéder à deux inconvénients principaux. D'abord, la similarité entre deux mouvements ou deux formes est difficile à déterminer au préalable et fortement subjective. Ensuite, nous aurions idéalement du utiliser de plus grand ensemble de données afin de définir plus précisément les valeurs. Toutefois, nous ne disposions pas du temps nécessaire à la réalisation de test à grande échelle pour chaque valeur.

Une fois ces quatre points améliorés, la détection devrait être plus efficace et le taux de reconnaissance plus élevé. S'il s'avérait que ce taux ne soit toujours pas suffisant, une piste d'amélioration supplémentaire serait l'amélioration des algorithmes de comparaison de formes et de reconnaissance des signes à l'aide de technique de machine learning. En effet, les algorithmes mis en place reste relativement simplistes. L'utilisation de méthodes de machine learning poussées pourrait peut-être permettre de réduire la confusion entre certains signes et donc permettre d'atteindre un meilleur taux de reconnaissance. Au final, si le taux obtenu est suffisamment bon, la mise en place de la segmentation automatique pourrait être explorée. Pour cela, une méthode envisageable serait de capturer l'ensemble des configurations des mouvements de la main pour une séquence donnée. Sur base de cet ensemble, la combinaison de signes la plus probable pourrait ensuite être recherchée. Le début et la fin de chaque signe pourraient être détectés en se basant sur les états initiaux et finaux des machines à états présentes dans la base de connaissances. Les informations de mouvements pourraient également servir à cette segmentation. Par exemple, si les mains sont inactives depuis un certain moment, cela signifie probablement que plus aucun signe n'est réalisé. Cela peut donc marquer la fin de la séquence à traiter.

Quatrième partie

Conclusion

Conclusion

L'un des projets du laboratoire de Langue des Signes de Belgique Francophone (LSFB-lab) est la création du premier large corpus informatisé illustrant l'usage actuel de la langue des signes de Belgique francophone (LSFB). Pour concevoir ce corpus, de nombreux sourds et malentendants ont été filmés lors de dialogues. Ainsi, 150 heures de vidéos ont été produites. Toutes ces vidéos doivent cependant être annotées afin de pouvoir servir de matériel de recherche. Parmi les 150 heures, le LSFB-lab a, à l'heure de l'écriture de ce mémoire, annoté environs 12 heures et produit la traduction pour 2 de celles-ci. Le principal problème rencontré par le laboratoire est le temps nécessaire au processus d'annotation. En effet, ce processus est actuellement réalisé manuellement et demande environ 8 heures de travail pour annoter 2 minutes d'une conversation. Pour tenter de palier ce problème, nous avons exploré la faisabilité de l'automatisation de ce processus grâce aux techniques de reconnaissance gestuelle. En raison du contexte particulier (absence de contrainte, éclairage, etc.) dans lequel les vidéos ont été tournées, l'application des techniques de reconnaissance gestuelle existantes était incertaine.

Pour déterminer cette faisabilité, nous avons intégré le LSFB-lab durant quatre mois. Tout d'abord, nous avons observé les annotateurs travailler et analysé les vidéos à notre disposition. De ces observations, il est ressorti qu'une phase de vérification avait lieu lors de l'annotation manuelle. La vérification des résultats produits par une annotation automatique ne constituait dès lors pas un problème. Nous avons également déterminé un certain nombre de difficultés présentes dans les vidéos et devant être prises en compte autant que possible. Ensuite, nous avons décidé de développer une solution de traitement automatique des vidéos. Pour cela, nous nous sommes renseignés sur le processus de reconnaissance gestuelle et des techniques permettant de mettre en place chaque étape de celui-ci. Les étapes de ce processus sont :

- la détection des zones de l'image contenant la main ;
- le suivi image par image des mains ;
- la reconnaissances des signes ;

Pour chacune de ces étapes, nous avons mis en place une méthode se basant sur les méthodes existantes mais tentant de tenir compte de la plupart des difficultés identifiées dans les vidéos. En outre, nous avons déterminé, sur base de la documentation sur la langue des signes, les caractéristiques devant être considérées pour reconnaitre un signe. Les caractéristiques ainsi choisies reprennent la configuration de la main, son emplacement ainsi que la trajectoire réalisée.

Finalement, nous avons évalué l'efficacité de la solution développée sur base des annotations manuelles à notre disposition. Le résultat de cette évaluation est que très peu de signes ont pu être reconnus. Le taux moyen de reconnaissance se situait aux alentours de 7.5% pour la première méthode et de 12.3% pour la seconde méthode. Dès lors, la solution a été jugée non satisfaisante. En effet, même si certaines exigences énumérées au point 4.3 ont effectivement été atteintes, obtenir un taux de reconnaissance de 70% était l'une des exigences prioritaires. La solution ne permettant pas de reconnaitre une majorité de signes, le résultat produit nécessitera une phase de correction beaucoup trop importante. Le gain de temps et de charge de travail obtenu par l'automatisation sera alors perdu en raison des corrections nécessaires. Au final, le processus d'annotation risque même de demander plus de temps et de travail pour les annotateurs bilingues que lors d'une annotation manuelle. Au vu de ces résultats non satisfaisants, nous avons porté un regard critique sur les causes de ce faible taux de reconnaissance. Suite à cela, nous avons déterminé qu'une solution visant à automatiser le processus d'annotation des vidéos composant le corpus du LSFB-lab n'était probablement pas réalisable dans les conditions actuelles.

Les causes de ce faible taux ont été identifiées comme étant essentiellement liées au contexte dans lequel les vidéos ont été tournées. Les particularités de celles-ci rendent l'application des techniques de reconnaissance gestuelle extrêmement difficile. L'amélioration de ces différents points permettrait probablement d'améliorer le taux de reconnaissance. D'abord, de nombreuses occlusions ont lieu durant les discussions. Cela implique que pour être en mesure d'extraire efficacement les différentes caractéristiques, ces occlusions doivent pouvoir être gérées. Or, la résolution de ces occlusions constitue un champs de recherche à part entière. Si des méthodes ont été proposées, celles-ci ne fonctionnent en général que dans certains cas. La méthode proposée dans [29] a par exemple des difficultés lorsque la personne est trop de profil. Ensuite, l'éclairage utilisé, l'absence de contrainte quant à l'habillage des personnes filmées et l'apparition du second signeur dans le champs de la caméra provoquent des problèmes pour la détection et le suivi des mains et de la tête. Concernant l'éclairage, celui-ci est responsable d'une augmentation du nombre de pixels de peau mal détectés ce qui engendre une mauvaise détection des formes des mains. L'habillage quant à lui a pour conséquence qu'un grand nombre de zones de peau est visible et donc détecté. Cela entraînent alors une augmentation du nombre d'occlusions ou une mauvaise détection de la forme de la main lorsque le bras est visible. L'apparition du second signeur dans le champs de la caméra est également un problème impactant le suivi des parties du corps. En effet, lorsque les mains de ce second signeur passe devant le signeur filmé, le module de suivi peut être amené à suivre celles-ci provoquant alors une détection et un suivi erroné pendant une certaine durée. Pour finir, la distinction entre certains signes visuellement proche est parfois difficile à réaliser en raison de l'angle de vue choisi et de la variation de celui-ci durant les vidéos. C'est par exemple le cas des signes "tu" et "il". Ce problème de distinction est d'autant plus fort qu'il arrive que cet angle de vue change d'un signeur à l'autre mais également pour un même signeur. Si ces changements sont généralement légers, ils sont suffisant pour que, combinés aux autres difficultés, ils impactent les formes perçues. Á l'avenir, il serait intéressant d'analyser plus en profondeur les résultats obtenus afin d'identifier les corrélations entre certaines particularités des signes (fusion, deux mains, etc.) et la reconnaissance de ceux-ci. Par exemple, les signes non

reconnus sont-ils majoritairement des signes impliquant des occlusions avec la tête ? Sont-ils majoritairement fait à une main ? Sont-ils plus long que les signes reconnus ? L'identification de ces corrélations permettrait d'orienter les recherches futures sur les points plus critiques.

D'autres causes du taux de reconnaissance faible ont été identifiées. Dans un premier temps, un bon nombre de valeurs ont été définies empiriquement. Des tests plus approfondis et à plus grande échelle seraient nécessaires pour avoir des valeurs plus adaptées. Dans un deuxième temps, les algorithmes de reconnaissance et de comparaison sont relativement simplistes. L'utilisation de méthodes de machine learning poussées pourrait permettre de réduire la confusion entre certains signes et donc permettre d'atteindre un meilleur taux de reconnaissance.

Dans le futur, si le laboratoire désire absolument accélérer son processus d'annotation, d'autres pistes devront être explorées. Nous en avons identifiées deux principales : l'une se basant toujours sur l'automatisation mais nécessitant d'enregistrer de nouvelles vidéos, l'autre se basant sur un système collaboratif permettant de garder les vidéos actuelles.

Lors de la mise en place de notre solution, nous avions décidé d'écarter les technologies de type Kinect. Nous avions fait ce choix en raison de la nature des vidéos actuelles. Ces dernières ayant été tournées avec de simples caméras, l'utilisation de Kinect pour leur annotation aurait demandé un travail bien trop conséquent. Dans un premier temps, les 150 heures de vidéos auraient du être reproduites devant Kinect. Ensuite, le lien entre ces nouvelles vidéos et les vidéos actuelles aurait du être réalisé en segmentant manuellement [1] les signes dans ces dernières. Au final, utiliser Kinect pour les vidéos actuelles aurait alourdi le travail d'annotation. Cependant, dans le cas où le laboratoire accepterait de tourner de nouvelles vidéos, l'utilisation de Kinect pour réaliser celles-ci serait idéal. En effet, outre les informations de couleur, Kinect est capable de fournir des informations de profondeur permettant de résoudre, entre autre, les problèmes d'occlusion constituant un frein majeur à la solution actuelle. De plus, Kinect est capable d'identifier et de suivre nativement les différentes parties du corps. Si le laboratoire accepte de tourner de nouvelles vidéos mais ne peut utiliser Kinect pour une raison ou une autre, une attention particulière devrait être prêtée aux problèmes énumérés aux chapitres précédents. Cependant, il faut être conscient qu'il est peut probable de réussir à résoudre l'entièreté de ceux-ci. Par exemple, même en contraignant un maximum l'environnement, il n'est pas possible d'éviter toutes les occlusions pouvant survenir. Par conséquent, une telle solution serait moins optimale que Kinect.

Si le laboratoire ne souhaite pas enregistrer de nouvelles vidéos mais désire absolument traiter les vidéos actuelles, la solution à une annotation plus rapide de celles-ci se trouvera probablement du coté du travail communautaire. En effet, une possibilité serait de permettre à la communauté des sourds et malentendants (ou plus largement des personnes comprenant la LSFB) de participer au travail d'annotation via, par exemple, une interface web. Si, fondamentalement, le temps nécessaire à l'annotation d'une vidéo serait toujours aussi long, le temps total d'annotation du corpus se verrait réduit en raison de l'augmentation massive du nombre d'annotateurs. Cependant, la faisabilité d'une telle solution dépend fortement de l'implication de la communauté.

1. Une segmentation automatique n'est bien sûr pas envisageable car cela reviendrait à être en mesure de reconnaitre les signes de manière automatique dans les vidéos actuelles.

Bibliographie

[1] Très chère langue des signes de Belgique francophone, où en es-tu ? *Sournal journal de la FFSB*, (117) :4–6, 2013.

[2] Edward J. Delp Alberto Albiol, Luis Torres. Optimum color spaces for skin detection. In *Proceedings 2001 International Conference on Image Processing*, volume 3, 2001.

[3] Aristote. *Metaphysique*. Pocket, 1995.

[4] Serge Belongie, Jitendra Malik, and Jan Puzicha. Shape matching and object recognition using shape contexts. *IEEE transactions on pattern analysis and machine intelligence*, 24(4) :509–522, 2002.

[5] Gary R Bradski. Computer vision face tracking for use in a perceptual user interface. 1998.

[6] David Brown, Ian Craw, and Julian Lewthwaite. A som based approach to skin detection with application in real time systems. In *in Proc. of the British Machine Vision Conference*, 2001.

[7] cambridge in color. Digital camera sensors. [En ligne ; accédé le 11 février 2017].

[8] Monica COMPANYS. *Prêt à Signer*. Editions Monica Companys, 2006.

[9] Jean Giot et Jean-Claude SCHOTTE. *Surdité, Différences, Ecritures*. DeBoeck Université, 1997.

[10] Ramsey Faragher. Understanding the basis of the kalman filter via a simple and intuitive derivation [lecture notes]. *IEEE Signal processing magazine*, 29(5) :128–132, 2012.

[11] Keinosuke Fukunaga and Larry Hostetler. The estimation of the gradient of a density function, with applications in pattern recognition. *IEEE Transactions on information theory*, 21(1) :32–40, 1975.

[12] Pierre Guitteny. *Entre sourds et entendants*. Editions Monica Companys, 2009.

[13] Michael Isard and Andrew Blake. Condensation—conditional density propagation for visual tracking. *International journal of computer vision*, 29(1) :5–28, 1998.

[14] M.J. Jones and J.M. Rehg. Statistical color models with application to skin detection. *International Journal of Computer Vision*, 46 :81–96, 2002.

[15] Yves Rybarczyk João Gameiro, Tiago Cardoso. Kinect-sign, teaching sign language to "listeners" through a game. In Alessandro Fantoni and Artur J. Ferreira, editors, *Conference on Electronics, Telecommunications and Computers – CETC 2013*, volume 17, page 384–391, 2014.

[16] Mohamed Kaâniche. *Gesture recognition from video sequences.* PhD thesis, Université Nice Sophia Antipolis, 2009.

[17] Adam Kendon. *Gesture : Visible Action as Utterance.* Cambridge University Press, 2004.

[18] Marie Zegers de Beyl Laurence Meurant. *Dans les coulisses d'un enseignement bilingue (langue des signes - français) à Namur.* Presses universitaires de Namur, 2009.

[19] Jae Y Lee and Suk I Yoo. An elliptical boundary model for skin color detection. In *Proc. of the 2002 International Conference on Imaging Science, Systems, and Technology*, 2002.

[20] Raman Maini and Himanshu Aggarwal. Study and comparison of various image edge detection techniques. *International journal of image processing (IJIP)*, 3(1) :1–11, 2009.

[21] Laurence Meurant. Un corpus informatisé en libre accès de vidéos et d'annotations de la langue des signes de belgique francophone (lsfb)., 2015.

[22] Wanli Ma Monish Parajuli, Dat Tran and Divya Sharma. Senior health monitoring using kinect. In *Communications and Electronics (ICCE), 2012 Fourth International Conference on*, pages 309–312.

[23] Takashi Masuda Jun-ichi Mizusawa Naofumi Kitsunezaki, Eijiro Adachi. Kinect applications for the physical rehabilitation. In *Medical Measurements and Applications Proceedings (MeMeA), 2013 IEEE International Symposium on.*

[24] N. Bourbakis P. Kakumanu, S. Makrogiannis. A survey of skin-color modeling and detection methods. *Pattern Recognition*, 40 :1106–1122, mars 2007.

[25] Platon. *Cratyle.* GF-Flammarion, 1967.

[26] Porphyre. *traité de l'abstinence.* Les Belles Lettres, 1977-1995.

[27] Reis L.O. Amorim P.H.J. et al. Ruppert, G.C.S. Touchless gesture user interface for interactive image visualization in urological surgery. *World Journal of Urology*, 30(5) :687–691, 2012.

[28] Anupam Agrawal Siddharth S. Rautaray. Vision based hand gesture recognition for human computer interaction : a survey. 2012.

[29] Paul Smith, Niels da Vitoria Lobo, and Mubarak Shah. Resolving hand over face occlusion. In *International Workshop on Human-Computer Interaction*, pages 160–169. Springer, 2005.

[30] Vladimir Vezhnevets, Vassili Sazonov, and Alla Andreeva. A survey on pixel-based skin color detection techniques. In *IN PROC. GRAPHICON-2003*, pages 85–92, 2003.

[31] Robert Y. Wang and Jovan Popović. Real-time hand-tracking with a color glove. *ACM Transactions on Graphics*, 28(3), 2009.

[32] Yushun Lin Zhihao Xu Yili Tang Xilin Chen Xiujuan Chai, Guang Li. Sign language recognition and translation with kinect, 2013.

[33] Quan Yuan, Stan Sclaroff, and Vassilis Athitsos. Automatic 2d hand tracking in video sequences. In *Application of Computer Vision, 2005. WACV/MOTIONS'05 Volume 1. Seventh IEEE Workshops on*, volume 1, pages 250–256. IEEE, 2005.

[34] Xenophon Zabulis, Haris Baltzakis, and Antonis Argyros. Vision-based hand gesture recognition for human-computer interaction. In *The universal access handbook*, pages 1–30. CRC Press, 2009.

Table des figures

Liste des tableaux