Thèse

Présentée pour obtenir le grade de docteur
de l'École Nationale Supérieure des
Télécommunications

Spécialité : **Signal et Images**

José Alonso YBANEZ ZEPEDA

Sujet :

ESTIMATION LINÉAIRE DE LA POSE TRIDIMENSIONNELLE D'UN VISAGE ET DE SES ACTIONS FACIALES.

A LINEAR ESTIMATION OF THE FACE'S TRIDIMENSIONAL POSE AND FACIAL EXPRESSIONS.

Mme. Alice CAPLIER	Rapporteurs
M. Maurice MILGRAM	
M. Francis SCHMITT	Examinateurs
M. Djamel MERAD	
M. Franck DAVOINE	Directeurs de thèse
M. Maurice CHARBIT	

To my family and friends for their support during my studies.

Acknowledgements

First of all, I want to thank Franck DAVOINE from ”Laboratoire Heudiasyc” at ”Université de Technologie de Compiègne”. He was one of my advisors and without his help this work would not have been possible. I want to thank him for his advice, his encouragement and especially for his friendship. I would also like to thank Maurice CHARBIT, my advisor too, for the fruitful discussions we had and also for the fun discussions we had.

I would like to thank the “Consejo Nacional de Ciencia y Tecnología (CONACYT)” for its financial support to perform my doctoral studies in France. I would also like to thank the “Secretaría de Educación Pública (SEP)” for their financial aid at the end of the thesis. Finally, I would like to thank the “Fundación Alberto y Dolores Andrade”, for its support and advice during all my studies.

I’m particularly grateful to my mother and my sisters that believe in me and give me all their support and love.

Thanks also to Peter WEYER-BROWN for helping me with the English language.

I want to thank all the people that helped me during my four years as a PhD student in the TSI laboratory of the ENST in France.

I would like to thanks all my friends, who encouraged me all the time, for the time we spent together and for their friendship.

Finally, I would like to thank Gisèle, for the support and help given at home, especially at the end of my PhD.

Abstract

The aim of the thesis is the face and facial animation tracking in video sequences. After introducing the topic, we propose a method to track the 3D pose and facial animations from a face detected at the beginning of a video sequence. We will then present a method to initialize the face tracking, estimating pose and form of an unknown face, based on a data base containing several faces. To achieve these two objectives (initialization and tracking), different approaches are described, using a geometric model of the face and matching two sets of variables: the perturbations of the geometric model in terms of the 3D pose parameters and the deformation and corresponding residues (errors between the current observation and appearance model of the face to be tracked). The relationship between the two sets of variables is described through a canonical correlation. We will show how to improve detection of the 3D pose and the shape of the face, using an incremental supervised learning. The effectiveness of these methods will be evaluated from a large number of video sequences, and tracking results will be compared to ground truths.

Français :

L'objectif de la thèse est le suivi de visages et d'animations faciales dans des séquences vidéo. Après avoir introduit le sujet, nous proposerons une première méthode permettant de suivre la pose 3D et les animations faciales du visage détecté au début d'une séquence vidéo. Nous présenterons ensuite une méthode permettant d'initialiser le suivi du visage, en estimant la pose et la forme d'un visage inconnu, à partir d'une base de visages vus de face. Pour atteindre ces deux objectifs (initialisation et suivi), différentes approches seront décrites, utilisant un modèle géométrique de visage et une mise en correspondance de deux ensembles de variables: les perturbations du modèle géométrique en terme de pose 3D et de déformations, et les résidus correspondant (erreurs entre l'observation courante et le modèle d'apparence du visage à suivre). La relation de dépendance entre les deux ensembles de variables est décrite à l'aide d'une analyse canonique des corrélations ou d'une analyse canonique régularisée noyaux. Nous montrerons enfin comment améliorer la détection de la pose et de la forme du visage, à l'aide d'un apprentissage supervisé incrémental. L'efficacité des méthodes sera évaluée à partir d'un grand nombre de séquences vidéo, et les résultats de suivi seront comparés à des vérités terrain.

Contents

List of Figures

List of Tables

Chapter 1

Resumé.

Ce travail s'inscrit dans le cadre de la détection et du suivi 3D des visages.

Le visage est un objet tridimensionnel très complexe qui nous apporte beaucoup d'informations, comme l'identité de la personne, le genre, l'âge, la position relative par rapport à la caméra, les mouvements, etc.

Avec le développement d'applications telles que la reconnaissance des expressions d'un visage ou la reconnaissance d'un individu à partir de son visage, et puis des ordinateurs de plus en plus puissants, de nombreux chercheurs se sont intéressés au problème de la détection et du suivi de visages ainsi que l'estimation de leur pose 3D.

Parmi les travaux les plus importants on va seulement citer ceux qui utilisent un modèle pour faire le suivi des visages.

Tout d'abord citons [88], où les auteurs proposent un algorithme qui se sert d'un modèle 2D pour suivre de différents objets. Dans ce cas les auteurs font le suivi de bateaux, de livres, et de visages, à l'aide d'un modèle 2D. Pour faire le suivi, les auteurs se servent d'une approche de type "Relevant Vector Machine" (RVM) pour obtenir les paramètres 2D du modèle.

Ensuite nous pouvons mentionner le travail [48], pour lequel les auteurs utilisent des modèles géométriques 3D. Ils proposent de faire le suivi du visage avec un cylindre. Pour l'algorithme de suivi les auteurs utilisent une approche de moindres carrés.

Egalement dans [51], dans ce cas les auteurs utilisent un modèle rigide 3D du visage. Ils emploient une méthode de détection des points caractéristiques. Puis ils font une classification de ces points pour assigner leur appartenance à une région du visage et ensuite ils s'en servent pour estimer la pose 3D du visage.

Un autre travail qui continue dans cette progression est [31], dans lequel les auteurs se servent d'un modèle géométrique 3D du visage pour suivre la pose et les mouvements faciaux. Pour ces expérimentations, les auteurs utilisent une méthode de descente du gradient.

Enfin nous devons citer les approches dîtes d'analyse par synthèse. Parmi ces approches les deux plus connues et utilisées sont les "Active Appearance Models" (AAM) et les "3D Morphable Models" (3DMM).

Les AAM's ont été proposés dans [21] et ont ouvert un large champ de recherches

qui ont donné lieu à de nombreuses publications. L'idée principale est de noter plusieurs images qui contiennent l'objet à suivre. Cette notation consiste à indiquer la localisation des points caractéristiques de l'objet, dans ce cas, des visages. Une fois que les visages de la base d'apprentissage sont enregistrés, on utilise l'analyse en composantes principales pour obtenir deux sous-modèles : un modèle de forme et un modèle d'apparence.Pour aboutir au suivi nous tentons de reconstruire l'image originale avec une image synthétique créé à partir de ces modèles.

Dans le cas des 3DMM's décrites dans [70] la façon de procéder est similaire, excepté que nous utilisons des scanners 3D pour effectuer l'apprentissage des visages.

Dans notre cas, nous présentons une méthode dite d'apprentissage supervisé, dont l'objectif global est de proposer une méthode pour estimer et suivre les paramètres de pose 3D ainsi que les paramètres des gestes faciaux. Pour cela, nous utilisons deux outils. Tout d'abord nous utilisons le modèle géométrique 3D du visage *Candide* [2], avec lequel nous créons des images du visage synthétiques, et ensuite nous prenons le formalisme CCA (Canonical Correlation Analysis) pour résoudre le problème de l'estimation et du suivi 3D de visages ainsi que l'analyse des mouvements faciaux (déformations de la zone des yeux et de la zone de la bouche). Nous allons aussi montrer plusieurs utilisations de ce formalisme pour faire du suivi : des paramètres d'animation faciale, de l'estimation de pose 3D et de forme de visages inconnus.

Ainsi ce résumé se structure de la façon suivante : dans un premier temps, nous exposerons notre représentation du visage. Pour cela, nous allons parler du modèle Candide et comment s'en servir pour aboutir à un visage normalisé. Ensuite nous formulerons notre problème pour faire l'estimation de la pose et des actions faciales. Dans cette partie nous allons présenter trois algorithmes : un premier pour faire uniquement le suivi de la pose, un second pour faire le suivi de la pose et des paramètres d'animation, et un troisième pour faire l'initialisation automatique. Finalement nous présenterons un algorithme d'apprentissage incrémental.

1.1 Représentation du visage.

Pour représenter le visage, nous utilisons le modèle *Candide* décrit en [2].

Il est constitué de n sommets tridimensionnels. Ce modèle peut être écrit comme un vecteur de dimension $3n$ de la façon suivante :

$$\mathbf{g}(\boldsymbol{\tau_a}, \boldsymbol{\tau_s}) = \overline{\mathbf{g}} + \mathbf{S}\boldsymbol{\tau_s} + \mathbf{A}\boldsymbol{\tau_a}.$$

Ici le premier terme $\overline{\mathbf{g}}$ représente la forme standard du modèle *Candide* et on peut le voir dans la figure 1.1. Le deuxième terme $\mathbf{S}\boldsymbol{\tau_s}$ représente les caractéristiques statiques (morphologie) du visage, telles que la position verticale des sourcils, du nez, des yeux et de la bouche, distance entre les yeux, etc, et qui sont représentées dans la figure 1.2.

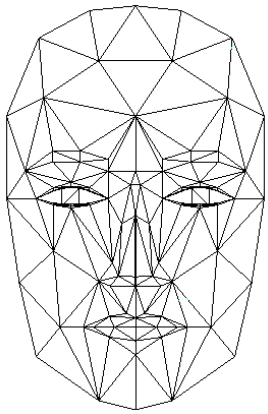


Figure 1.1: Forme standard du modèle *Candide*

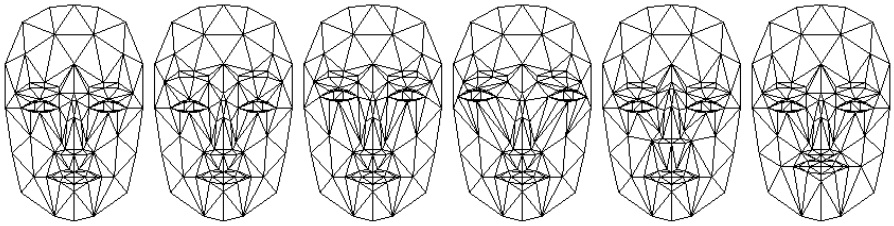


Figure 1.2: Caractéristiques statiques (morphologie) du visage

Finalement, le troisième terme $\mathbf{A}\tau_a$ représente les caractéristiques dynamiques, telles que l'ouverture et fermeture des yeux, de la bouche, etc, et qui sont representées dans la figure 1.3. C'est avec ces deux termes que nous pouvons modifier la forme standard du modèle *Candide*.

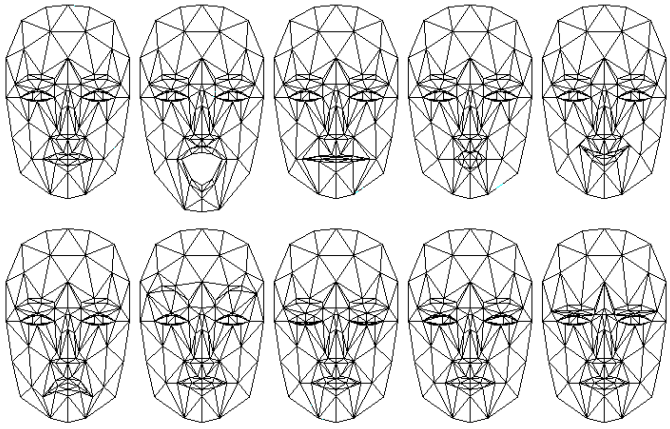


Figure 1.3: Caractéristiques dynamiques

Comme nous nous intéressons à faire le suivi du visage dans un espace 3D, nous considerons des paramètres de rotation et de translation pour pouvoir modifier la pose 3D du modèle. On peut alors définir notre vecteur d'état comme :

$$b = \left[\theta_x, \theta_y, \theta_z, t_x, t_y, t_z, \boldsymbol{\tau}_a^T\right]^T.$$

Ici les paramètres θ de notre vecteur d'état correspondent aux paramètres de rotation, les paramètres t de notre vecteur d'état correspondent aux paramètres de translation et $\boldsymbol{\tau}_a^T$ correspondent aux paramètres d'animations faciales. Les paramètres d'animations faciales choisies correspondent aux principaux mouvements des yeux, ainsi qu'aux mouvements de la bouche.

C'est avec ce modèle que nous créons notre vecteur d'observation. Pour cela, nous plaçons le modèle sur le visage de la personne, modèle que nous adaptons aux caractéristiques de ce visage. Puis nous utilisons la texture de l'image et l'appliquons au modèle 3D. Ensuite, nous faisons la projection du modèle vu de face, avec tous les paramètres de rotation ainsi que les paramètres d'expression $\boldsymbol{\tau}_a$ fixés à zero, autrement dit avec une expression neutre. Finalement, nous ajoutons deux vues de profil, qui vont nous permettre de suivre le visage dans une plage de rotations plus importante. Un exemple du vecteur d'observation $\mathbf{x}_t$, de taille 6912, est montré dans la figure 1.4.

Figure 1.4: Visage sans forme ni expression.

Maintenant que nous avons exposé notre façon de représenter le visage, nous pouvons poursuivre sur les problématiques de l'estimation de la pose 3D.

1.2 Formulation du problème.

Il consiste à trouver la relation qui existe entre une variation du vecteur d'état $\Delta\mathbf{b}_t$, qui correspond au placement du modèle *Candide* sur une image contenant un visage, et le résidu qui existe entre le visage sans expression obtenu avec ce vecteur d'état perturbé comparé avec une référence $\Delta\mathbf{x}_t$. Nous proposons qu'il existe cette relation linéaire entre les deux vecteurs :

$$\Delta\mathbf{b}_t = \mathbf{G}\Delta\mathbf{x}_t$$

avec $\Delta\mathbf{x}_t = \mathbf{x}_t - \mathbf{x}_t^{(ref)}$ et $\Delta\mathbf{b}_t = \mathbf{b}_t - \mathbf{b}_{t-1}$.

Cette relation est alors obtenue avec une méthode d'apprentissage supervisé. Dans une méthode d'apprentissage supervisé on a deux ensembles d'exemples qui vont être utilisés pour apprendre la relation qui existe entre ces deux ensembles :

$$\mathbf{A}_1 = [\Delta\mathbf{x}_1, \Delta\mathbf{x}_2, \ldots \Delta\mathbf{x}_m]$$

$$\mathbf{A}_2 = [\Delta\mathbf{b}_1, \Delta\mathbf{b}_2, \ldots \Delta\mathbf{b}_m]$$

Dans notre cas, nous avons choisi l'analyse canonique des corrélations (CCA). Pour cela on va créer la matrice $\mathbf{A}_2$ qui contient les variations de notre vecteur d'état et la matrice $\mathbf{A}_1$ qui contient les visages sans expression comparés avec la référence. C'est ainsi que nous obtenons la matrice $\mathbf{G}$ qui fait le lien entres ces deux vecteurs.

1.3 Analyse canonique des corrélations.

L'analyse canonique des corrélations est un formalisme mathématique qui est utilisé pour trouver la relation qui existe entre deux ensembles de données. Dans notre cas, ces deux ensembles sont :

$$\mathbf{A}_1 = [\Delta\mathbf{x}_1, \Delta\mathbf{x}_2, \ldots \Delta\mathbf{x}_m]$$

qui contient des résidus obtenus de la différence entre le visage sans expression créé synthétiquement avec un vecteur d'état perturbé et une référence qui correspond au visage sans expression par conséquent au vecteur d'état sans perturbation, et

$$\mathbf{A}_2 = [\Delta\mathbf{b}_1, \Delta\mathbf{b}_2, \ldots \Delta\mathbf{b}_m]$$

qui contient les perturbations du vecteur d'état utilisé pour la création des visages synthétiques. Ces vecteurs sont de dimensions différentes :

$$\mathbf{A}_1 \in \mathbb{R}^{d\times m} \text{ et } \mathbf{A}_2 \in \mathbb{R}^{p\times m}$$

L'idée principale derrière ce formalisme consiste à trouver des couples de direction $\mathbf{w}_1$ et $\mathbf{w}_2$ dans lesquels on va projeter nos données originales de la façon suivante :

$$\mathbf{z}_1 = \mathbf{A}_1\mathbf{w}_1 \quad \text{et} \quad \mathbf{z}_2 = \mathbf{A}_2\mathbf{w}_2$$

La corrélation entre ces projections s'écrit alors :

$$\rho = \frac{\mathbf{z}_2^T\mathbf{z}_1}{\sqrt{\mathbf{z}_2^T\mathbf{z}_2}\sqrt{\mathbf{z}_1^T\mathbf{z}_1}}$$

Avec les contraintes :

$$\|\mathbf{z}_1\| = 1 \text{ et } \|\mathbf{z}_2\| = 1$$

On obtient au maximun $k = \min(d, p)$ couples de directions. On résout le problème avec trois SVD de la façon suivante:

$$\mathbf{A}_1 = \mathbf{U}_1\mathbf{D}_1\mathbf{V}_1^T \ , \ \mathbf{A}_2 = \mathbf{U}_2\mathbf{D}_2\mathbf{V}_2^T \text{ et } \mathbf{V}_1^T\mathbf{V}_2 = \mathbf{U}\mathbf{D}\mathbf{V}^T$$

De façon que les k couples de direction qui maximisent la corrélation sont donnés par l'expression :

$$\mathbf{W}_1 = \mathbf{U}_1\mathbf{D}_1^{-1}\mathbf{U} \text{ et } \mathbf{W}_2 = \mathbf{U}_2\mathbf{D}_2^{-1}\mathbf{V}$$

Si on revient à la formulation du problème on peut montrer que la matrice qui fait le lien entre nos deux matrices peut s'écrire :

$$\mathbf{G} = \mathbf{U}_2\mathbf{D}_2\mathbf{V}\mathbf{U}^T\mathbf{D}_1^{-1}\mathbf{U}_1^T$$

Ce formalisme existe aussi dans une version à noyau, qui est utile dans le cas où nos données présentent un comportement non linéaire.

1.4 Analyse canonique des corrélations à noyau.

L'idée principale derrière les méthodes à noyau consiste à faire la projection des données originales dans un espace de dimension plus grande, tel que dans ce nouvel espace on peut trouver plus facilement une relation linéaire entre les données.

$$\phi : \mathbb{R}^p \mapsto \mathbb{R}^s, s > p.$$

De la même façon que pour la CCA nous pouvons écrire :

$$\mathbf{w}_{\phi_1} = \phi_1(\mathbf{A}_1)^T\mathbf{f}_{\phi_1}$$

et

$$\mathbf{w}_{\phi_2} = \phi_2(\mathbf{A}_2)^T\mathbf{f}_{\phi_2}$$

Dans notre cas, nous utilisons le noyau Gaussian :

$$\mathbf{K}(\mathbf{x}_i, \mathbf{x}_j) = \exp\left(-\frac{\|\mathbf{x}_i - \mathbf{x}_j\|^2}{2\sigma^2}\right), \ i = 1 \ldots m \ \ et \ \ \ j = 1 \ldots m.$$

Avec cette méthode, si on utilise la même formulation que dans le cas de la CCA nous pouvons exprimer la mise à jour du vecteur d'état ainsi :

$$\Delta\mathbf{b}_t = \mathbf{K}_t\mathbf{f}_\phi\mathbf{G}.$$

Ici le vecteur $\mathbf{K}_t$ est estimé à chaque instant comme cela:

$$\mathbf{K}_t(\mathbf{x}_t, \mathbf{x}_i) = \exp\left(-\frac{\|\mathbf{x}_t - \mathbf{x}_i\|^2}{2\sigma^2}\right),\ i = 1 \ldots m$$

L'inconvénient de cette méthode est que nous devons garder tous les vecteurs utilisés pendant l'entraînement et comparer à chaque instant le vecteur observé avec tous ces vecteurs.

1.5 Suivi des paramètres de pose.

Une fois que nous avons montré la CCA et comment s'en servir pour lier les variations des paramètres du modèle *Candide* avec le résidu entre le vecteur d'observation et la référence, nous allons présenter l'algorithme de suivi.

Il se décompose en trois parties :

- ⋆ Initialisation.
- ⋆ Entraînement.
- ⋆ Suivi.

1.5.1 Initialisation (manuelle).

L'initialisation consiste à placer manuellement le modèle *Candide* sur le visage de la personne, et de l'adapter aux caractéristiques morphologiques de la personne à l'image. De cette façon nous obtenons alors le vecteur de réference $\mathbf{x}_t^{(ref)}$.

Figure 1.5: Initialisation du modèle *Candide*.

1.5.2 Entraînement.

Pendant l'entraînement nous allons créer des images de synthèse à partir de l'image de référence. Pour cela, nous ajoutons des perturbations au vecteur d'état, qui sont stockées dans une matrice. A chaque perturbation correspond une image sans expression comme celle montrée en haut à gauche de l'image 1.6. On fait la soustraction de l'image de référence et on stocke les résultats dans une deuxième matrice. Les perturbations ont été choisies dans une grille symétrique non uniforme, qui est plus dense près de l'origine et moins dense quand on s'éloigne de l'origine. Une fois les images exploitables, on utilise la CCA pour obtenir la relation linéaire entre ces données.

1.5.3 Algorithme de suivi des paramètres de pose.

L'algorithme résultant est donné par la relation suivante :

$$\hat{\mathbf{b}}_t = \hat{\mathbf{b}}_{t-1} + \mathbf{G}_{Pose}(\mathbf{x}_t - \mathbf{x}_0^{(ref)})$$

Nous avons constaté que si nous gardions la référence constante pendant tout le suivi, au bout d'un moment le suivi devenait moins précis, tel que nous pouvons le voir dans la figure 1.7.

Nous avons constaté que le fait de faire une mise à jour de la référence permettait d'améliorer la netteté du suivi, ce qui rend notre algorithme :

$$\hat{\mathbf{b}}_t = \hat{\mathbf{b}}_{t-1} + \mathbf{G}_{Pose}(\mathbf{x}_t - \mathbf{x}_t^{(ref)})$$

On fait la mise à jour de la référence avec $\alpha = 0.99$:

$$\mathbf{x}_{t+1}^{(ref)} = \alpha \mathbf{x}_t^{(ref)} + (1-\alpha)\hat{\mathbf{x}}_t$$

1.5.4 Résultats.

La vérification du suivi a été faite sur les 45 séquences vidéo de La Cascia [1] [48]. Ces séquences représentent 5 personnes différentes. Pour chaque personne existent 9 séquences de 200 trames à 30 fps, de taille 320×240 pixels. Ces séquences sont fournies avec des données correspondantes aux rotations et translations, et ont été obtenues avec un capteur posé sur la tête de chaque individu. En conséquence, il y a une différence entre l'origine du système de coordonnées car le modèle *Candide* à son origine au nez, tandis que la vérite de terrain fournie à son système sur la tête. Autrement dit, ce qui correspond dans un système à une rotation, équivaut dans l'autre système à une rotation ajoutée à une translation. On peut regarder les résultats dans la figure 1.8, où on observe les valeurs estimées comparées à la vérité terrain fournie. Dans ces séquences, les visages ne sont pas expressifs (pas d'animations faciales).

[1] $http://www.cs.bu.edu/groups/ivc/HeadTracking/$

Figure 1.6: Exemples de perturbations dans les paramètres de rotation utilisés pour l'entraînement. En haut à gauche on peut voir les vecteurs d'observation obtenus avec ces perturbations.

Les résultats obtenus avec cet algorithme pour l'estimation de la pose nous ont mené à étendre cet algorithme pour estimer aussi les paramètres d'animation faciale.

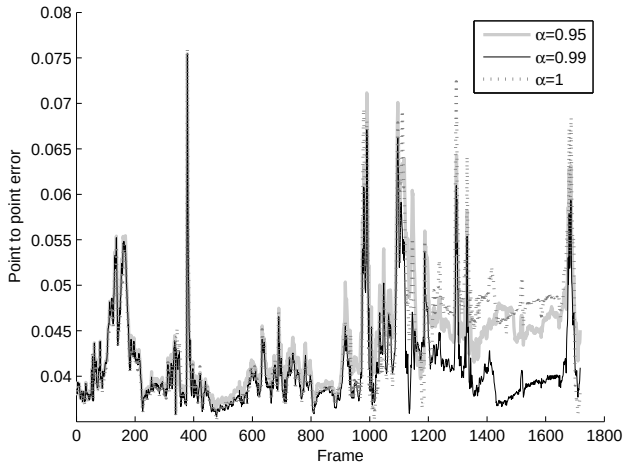


Figure 1.7: Facteur d'oubli utilisé pour la mise à jour de la référence.

1.6 Suivi des paramètres de pose et d'animation faciale.

Nous avons fait l'extension de notre algorithme pour estimer conjointement des paramètres de pose 3D et des paramètres d'animation faciale. Cela a donné lieu à notre algorithme de suivi avec un seul modèle.

1.6.1 Suivi avec un modèle.

L'algorithme de suivi ressemble ainsi à celui utilisé pour l'estimation seule de la pose 3D. Sauf que, dans ce cas, la matrice $\mathbf{G}$ est calculée pour estimer conjointement des paramètres de pose 3D et des paramètres d'animation faciale.

$$\hat{\mathbf{b}}_t = \hat{\mathbf{b}}_{t-1} + \mathbf{G}_{General}(\mathbf{x}_t - \mathbf{x}_t^{(ref)})$$

On fait la mise à jour de la référence avec l'équation :

$$\mathbf{x}_{t+1}^{(ref)} = \alpha \mathbf{x}_t^{(ref)} + (1-\alpha)\hat{\mathbf{x}}_t$$

Néanmoins, nous avons constaté que le fait de réaliser conjointement les paramètres de pose 3D et les paramètres d'animation faciale rendait moins précise l'estimation de la pose 3D. C'est la raison pour laquelle nous avons décidé de séparer l'estimation de la pose 3D des paramètres d'animation faciale. C'est à dire nous faisons, dans un premièr temps, l'estimation de la pose 3D du visage et, dans un second temps, nous estimons les paramètres d'animation faciale.

1.6.2 Suivi avec deux modèles.

Pour cela, nous avons donc utilisé deux modèles : l'un qui contenait exclusivement les perturbations aux paramètres de pose, et l'autre avec les perturbations

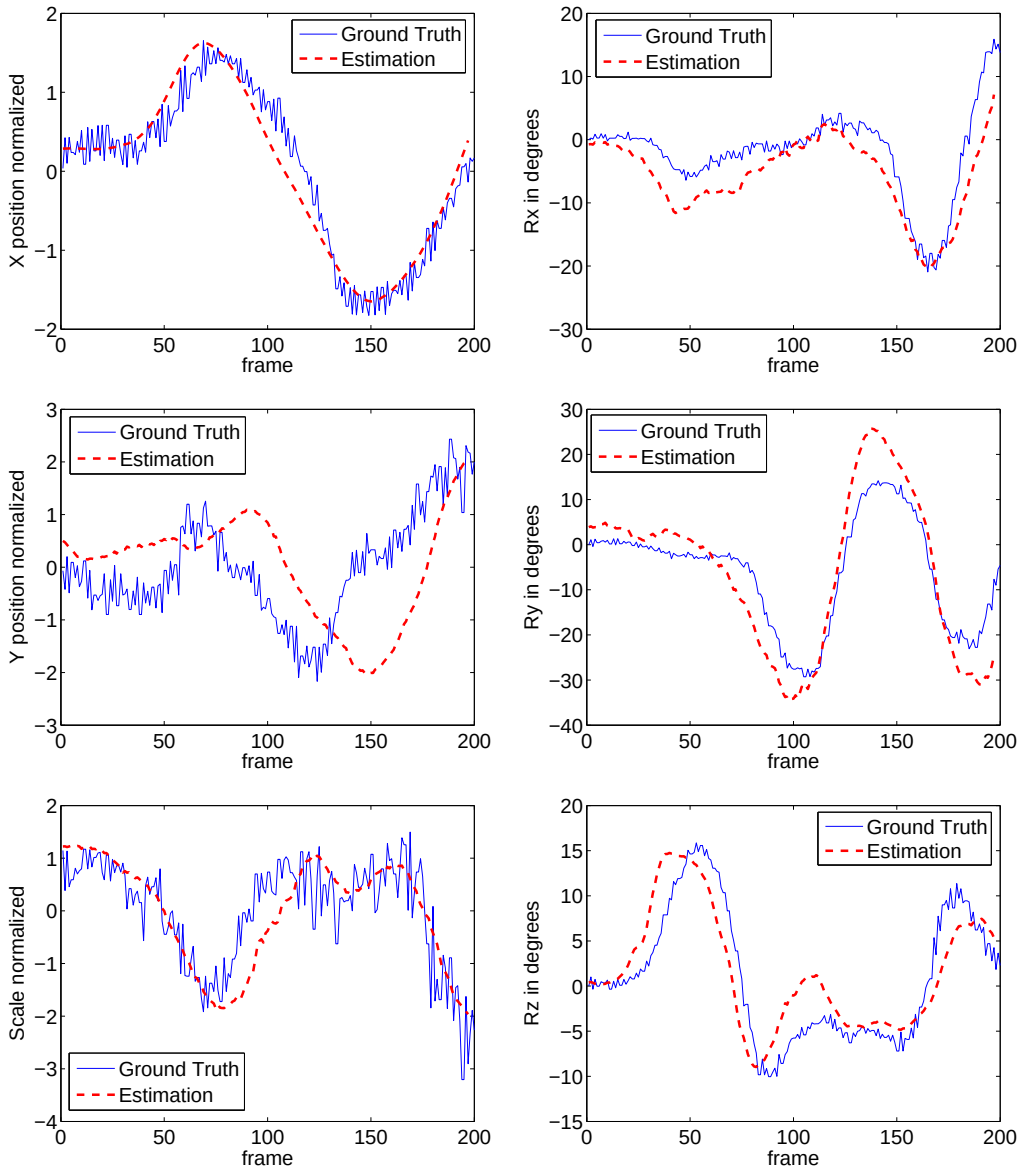


Figure 1.8: Comparaison de la vérité de terrain contre les paramètres de pose estimés.

aux paramètres d'animation. L'algorithme de suivi est devenu alors :

$$\hat{\mathbf{b}}_t = \hat{\mathbf{b}}_{t-1} + \mathbf{G}_{Pose}(\mathbf{x}_t - \mathbf{x}_t^{(ref)})$$

et

$$\hat{\mathbf{b}}_t = \hat{\mathbf{b}}_{t-1} + \mathbf{G}_{Animation}(\mathbf{x}_t - \mathbf{x}_t^{(ref)})$$

On fait la mise à jour de la référence, qui demeure la même pour les deux modèles:

$$\mathbf{x}_{t+1}^{(ref)} = \alpha \mathbf{x}_t^{(ref)} + (1-\alpha)\hat{\mathbf{x}}_t$$

Dans ce cas, les résultats obtenus étaient beaucoup plus nets, et on a décidé de découper encore une fois les paramètres d'animation en deux "sous-modèles".

1.6.3 Suivi avec trois modèles.

Dans ce cas nous avons décidé de découper les paramètres d'animation correspondant à la région de la bouche, des paramètres correspondant à la région des yeux. Nous en avons profité pour augmenter la résolution des vecteurs d'observation pour ces régions, ce qui a donné une nette amélioration surtout au niveau de la bouche, comme cela sera montré dans la partie des résultats. L'algorithme de suivi devient alors :

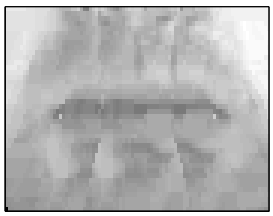
$$\hat{\mathbf{b}}_t = \hat{\mathbf{b}}_{t-1} + \mathbf{G}_{Pose}(\mathbf{x}_t - \mathbf{x}_t^{(ref)})$$
$$\hat{\mathbf{b}}_t = \hat{\mathbf{b}}_{t-1} + \mathbf{G}_{Bouche}(\mathbf{x}_t - \mathbf{x}_t^{(ref)})$$
$$\hat{\mathbf{b}}_t = \hat{\mathbf{b}}_{t-1} + \mathbf{G}_{Yeux}(\mathbf{x}_t - \mathbf{x}_t^{(ref)})$$

On fait la mise à jour de chaque référence :

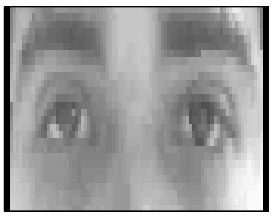
$$\mathbf{x}_{t+1}^{(ref)} = \alpha \mathbf{x}_t^{(ref)} + (1-\alpha)\hat{\mathbf{x}}_t$$

Dans la figure 1.6.3 nous pouvons voir ces trois modèles qui correspondent au paramètres pour l'estimation de la pose 3D, pour l'estimation des paramètres de la bouche, et finalement pour l'estimation des paramètres des yeux.

a) 96×72,



b) 88×42,



c) 86×28.

1.6.4 Suivi avec des points d'interêt.

L'utilisation des points d'intérêt, aussi appelé méthode locale, présente plusieurs avantages, tels que mentionnés dans [51] et [24]. Parmi les avantages par rapport à une méthode globale comme celle présentée précédemment, nous pouvons citer les suivantes :

⋆ Robustesse par rapport aux variations d'échelle,

- ⋆ robustesse par rapport aux variations de point d'observation,
- ⋆ robustesse par rapport aux variations d'illumination,
- ⋆ robustesse par rapport aux bruits du fond,
- ⋆ robustesse par rapport aux occultations partielles,
- ⋆ robustesse face aux variations de pose de l'objet par rapport aux images de l'ensemble d'apprentissage.

Dans notre cas, nous avons utilisé des points du modèle *Candide* qui correspondent aux points les plus importants du visage. Ces points peuvent être visualisés dans la figure 1.9, où on peut comparer l'image pour l'approche locale comparée à l'image utilisée pour l'approche globale. Dans le cas de l'image pour cette approche locale nous avons decidé de ne pas ajouter les deux images de profil.

On peut alors dire que la différence entre cette approche et l'approche décrite précédemment est la façon dont on construit le vecteur $\mathbf{x}_t = \mathcal{W}(\mathbf{g}(\mathbf{b}_t), \mathbf{y}_t)$. Dans ce cas, nous utilisons le modèle 3D *Candide* pour créer une image du visage vu de face sans expression, c'est-à-dire, avec tous les paramètres de rotation fixés à zéro et les paramètres d'expression $\tau_{\mathbf{a}}$ également mis à zéro. Ensuite nous utilisons des fenêtres de taille 6×6 pixels autour de 96 points choisis du modèle. Cette taille à été obtenue de façon expérimentale. De plus, on normalise chaque fenêtre de façon indépendante. Finalement nous effectuons la concaténation de ces fenêtres dans un seul vecteur $\mathbf{x}_t$ de taille 3456.

(a) (b)

Figure 1.9: (a) Image du visage stabilisée pour une approche locale (b) Image du visage stabilisée pour une approche globale.

Pour le suivi, l'algorithme reste le même que celui décrit précédemment.

1.6.4.1 Résultats.

Le premier test que nous avons fait a été de vérifier l'estimation correcte des paramètres 3D de la pose. Le résultat peut être apprécié dans la figure 1.10. Ici sont montrés les résultats quand sont estimés les paramètres 3D de pose dans les séquences de LaCascia, également quand sont estimés les paramètres de pose et

d'animation faciale. On peut voir que les résultats obtenus avec les deux algorithmes sont similaires, ce qui montre que le fait d'estimer tous les paramètres affecte peu l'estimation de la pose.

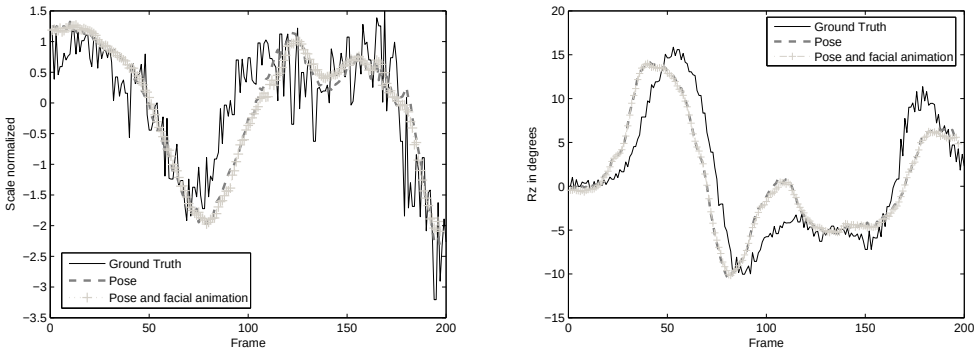


Figure 1.10: Comparaison de l'algorithme sans et avec paramètres d'animation faciale.

En revanche ces séquences vidéo ne contiennent pas d'animation faciale. Pour vérifier le comportement de cet algorithme face aux animations faciales, nous avons utilisé la séquence vidéo "Talking Face" du Réseau européen FGnet[2]. Cette séquence est celle d'une personne engagée dans une conversation. Il y a 5000 trames de taille 720 x 576 pixels. Il est fourni avec une vérité terrain qui contient des coordonnées 2D de 68 points du visage. Nous avons choisi 52 points qui étaient les plus proches des points du modèle *Candide*. Dans la figure 1.11 on voit l'erreur moyenne pour chaque point. On montre la comparaison quand on utilise un, deux et trois modèles pour faire le suivi des paramètres de pose et d'animation. Dans ce cas, on peut voir une nette amélioration quand on passe d'un à deux modèles, et le fait de passer à trois modèles améliore le suivi de la bouche. On montre aussi dans cette figure les 52 points du modèle *Candide* le plus proche des points fournis comme vérité de terrain.

Dans l'image nous pouvons apprécier l'évolution de l'erreur moyenne dans une partie de la séquence talking face. On voit l'apparition de sommets dans la courbe qui correspond aux images où des fortes rotations ont lieu. Si on regarde en détail le visage qui correspond à ces images, on voit que le suivi est assez net.

L'évolution temporelle de l'écart type aux sommets dans la figure 1.12 correspond aux paramètres de rotation :

⋆ Au temps 476 : $R_x = -3.6°$, $R_y = 11.2°$, $R_z = 2.8°$

⋆ Au temps 993 : $R_x = -3.7°$, $R_y = 36.6°$, $R_z = -1.4°$

⋆ Au temps 1107 : $R_x = -14.42°$, $R_y = 18.8°$, $R_z = -10.83°$

[2] $http://www-prima.inrialpes.fr/FGnet/data/01-TalkingFace/talking\_face.html$

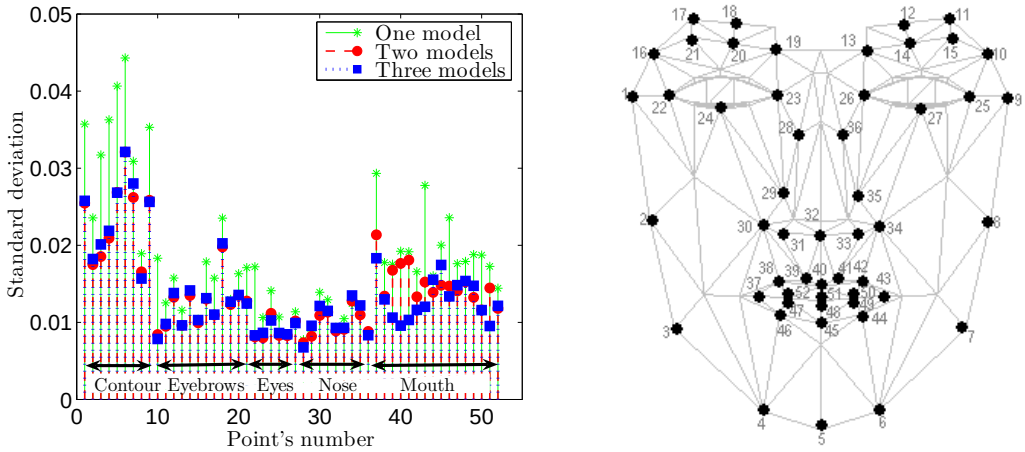


Figure 1.11: Écart type de 52 points sur la séquence vidéo "talking face".

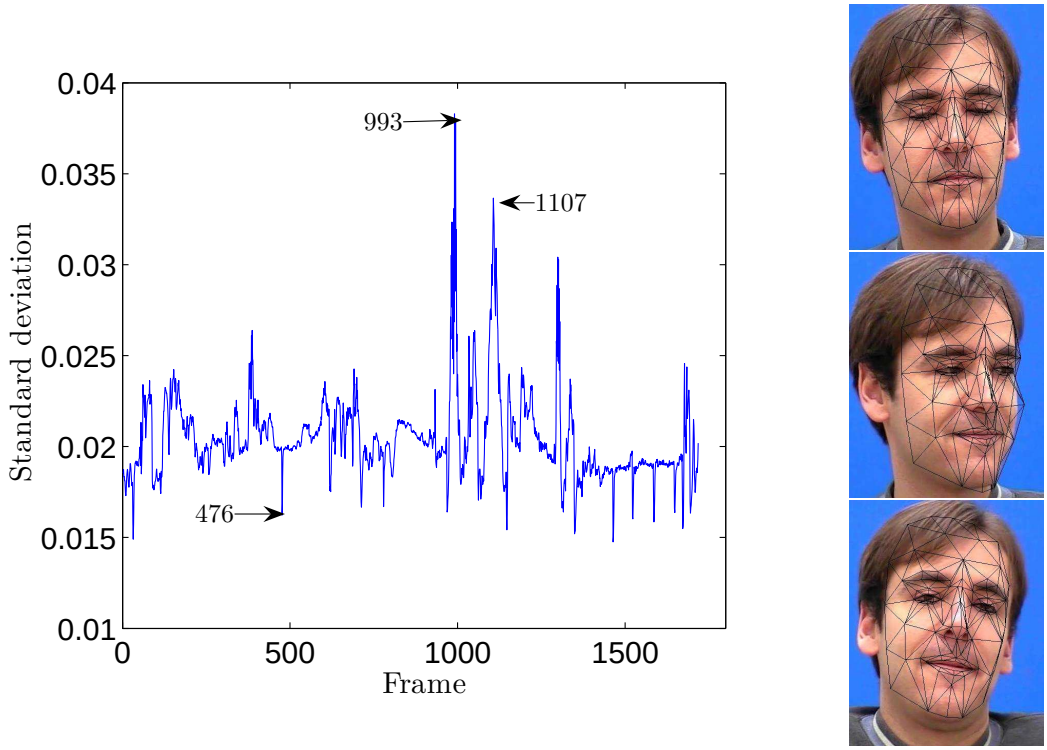


Figure 1.12: Écart type pour la séquence "talking face". On montre à droite les visages correspondants aux sommets dans la courbe à l'image 476, 993, et 1107.

On peut voir que malgré le fait d'avoir des sommets qui correspondent à de très fortes variations de pose et d'animation, le suivi du visage est assez précis. En revanche, si on essaie d'aller plus loin au niveau des rotations, comme l'apprentisage a été fait pour une vue de face, l'algorithme ne fonctionnera pas.

Finalement nous avons observé la robustesse de l'estimation des paramètres d'animation quand on introduit à l'estimation de la pose un bruit Gaussian. Pour

faire cela, nous avons bruité les paramètres de pose avec un écart type de 1° pour les rotations et de 1% de la largeur du visage pour les translations. Ensuite nous avons estimé avec les paramètres de pose bruités les paramètres d'animation faciale. On peut voir les résultats dans l'image 1.13. En haut nous remarquons le paramètre de translation correspondant au paramètre $\mathbf{t}_z$ avant et après le fait d'avoir ajouté le bruit, et en bas, le paramètre qui contrôle les mouvements des sourcils quand il est estimé avec les paramètres de pose bruités. On voit que malgré le bruit ajouté aux paramètres de pose, les résultats obtenus sont très ressemblant à ceux obtenus quand on n'ajoute pas de bruit.

Dans l'image 1.14 nous pouvons voir des images qui correspondent à plusieurs séquences vidéos différentes, parmi celles de LaCascia, la "talking face", et nos séquences vidéos propres.

Pour faire la validation de l'algorithme local, nous avons utilisé la séquence vidéo "talking face". nous pouvons apprécier dans la figure 1.15 que l'écart type moyen est très similaire entre les deux approches, néanmoins, il y a des sommets qui correspondent aux mouvements importants du visage ou à des expressions faciales. On peut bien voir que le sommet dans l'image 992 correspond à une rotation en y de 36.62°. Pour le sommet correspondant à l'image 1102, les rotations en x, y et z correspondent dans l'ordre à -13.3°, 18.9° et -10.5°. On peut voir dans la figure que les deux approches présentent un comportement similaire, mais dans le cas de l'approche locale, il y a plus d'oscillations. Cela est dû au fait que cette approche locale est plus sensible aux fortes rotations et aux expressions faciales.

La figure 1.15 présente aussi l'écart type pour chaque point. Nous pouvons apprécier que les points qui présentent un écart type plus important correspondent aux points du contour du visage. On peut voir que le comportement de l'approche locale est inférieur au comportement de l'approche globale. Cela est dû au fait que dans l'approche globale nous utilisons plus d'information que dans le cas local, surtout si on considère que deux vues de profil sont ajoutées pour le cas global. Ces vues de profil améliore la robustesse face aux rotations, ce qui est montré dans la figure 1.16. Au niveau du temps de calcul, l'approche locale est plus rapide avec un temps moyen de calcul par image de 26 ms tandis que l'approche globale à un temps moyen de 46 ms. Aussi au niveau de l'apprentissage on voit une amélioration car les temps sont de 29.1 et 33.2 secondes pour l'approche locale et globale.

Finalement pour voir la robustesse face aux variations d'illumination, nous avons utilisé la séquence vidéo de l'Université Polytechnique de Madrid qui contient 967 images [11][3]. Quelques images sont présentées dans la figure 1.16. Avec ces images nous constatons encore une fois que l'approche globale est plus robuste que l'approche locale.

[3]http://www.dia.fi.upm.es/∼pcr/downloads.html

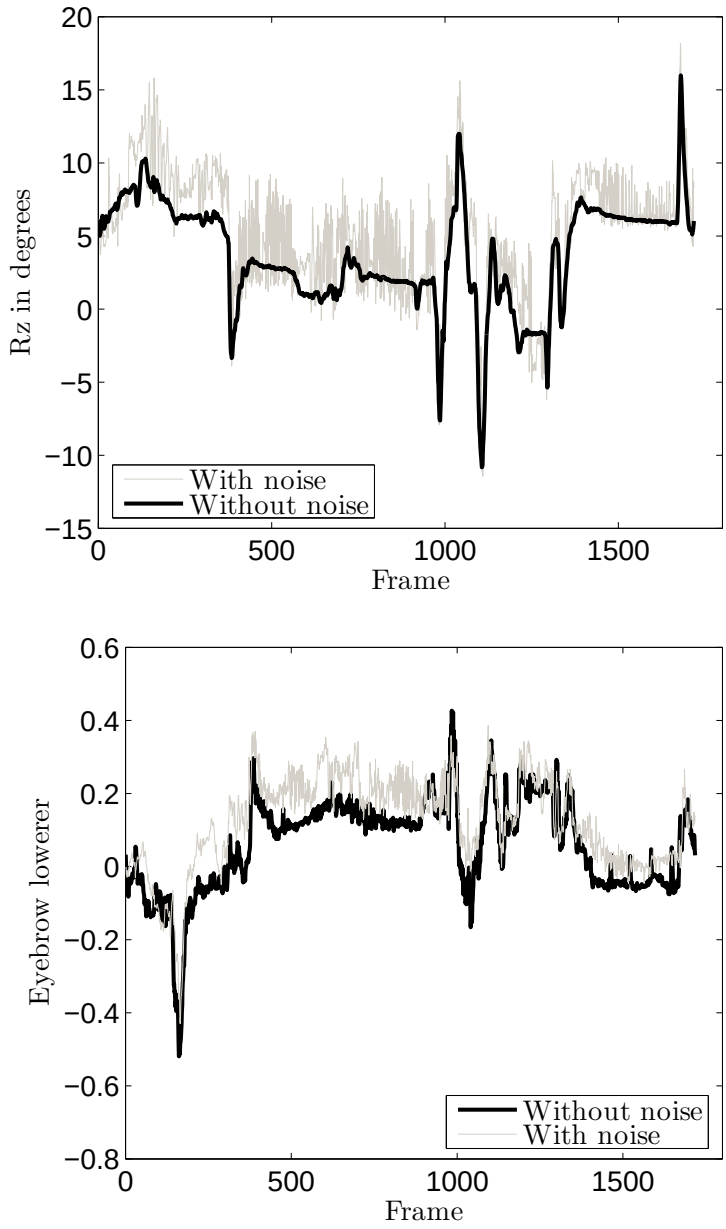


Figure 1.13: En haut : paramètre $\mathbf{t}_z$ avec et sans bruit. En bas : paramètre d'animation faciale qui contrôle les sourcils.

1.7 Initialisation automatique.

Dans cet algorithme on s'est proposé d'estimer de façon automatique la pose 3D des visages dans les séquences vidéos. Pour cela, nous avons utilisé une base de données publique qui contient 37 visages vus de face avec un éclairage homogène. La figure 1.17 nous présente quelques exemples. Cette base de données contient les visages de 7 femmes et de 30 hommes vus de face, avec une expres-

Figure 1.14: Résultats obtenus avec quelques séquences vidéos différentes.

sion neutre. Chaque image est de dimension 640×480.

Pour faire cet apprentissage, nous devons appliquer une normalisation, qui doit adapter les visages de telle façon que toutes les parties des visages se trouvent toujours à la même place, c'est à dire, nous allons placer le nez, la bouche et les yeux dans la même position pour tous les visages, tel que nous pouvons le voir dans la figure 1.18. Dans cette image on voit bien la modification du visage due à cette normalisation.

L'entraînement a été réalisé de la même façon que dans le cas de suivi expliqué précédemment, sauf que nous utilisons une grille autour de l'origine, comme il a été expliqué précédemment, mais on a ajouté du bruit blanc Gaussian pour chaque paramètre, de façon à ne pas avoir les mêmes points pour toutes les personnes. Dans ce cas, on a estimé un visage moyen à partir de toutes les images

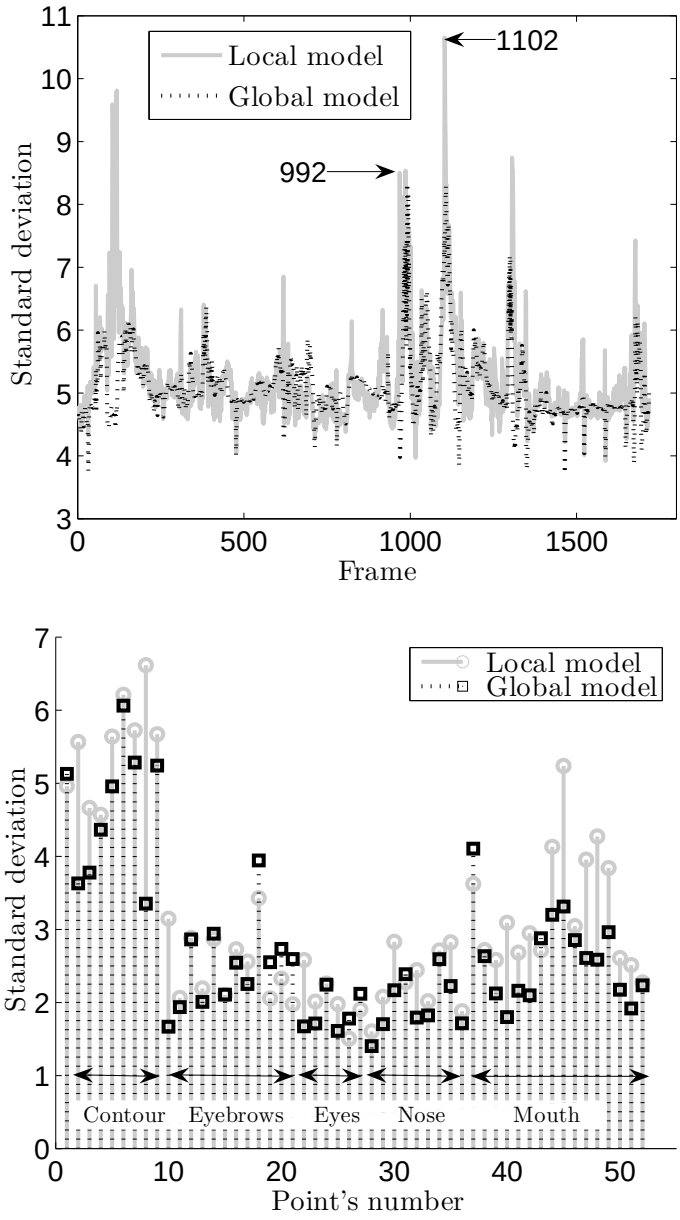


Figure 1.15: En haut: Evolution temporelle de l'écart type moyen. En bas: Écart type pour chaque point.

utilisés pour l'entrainement. Ce visage ne contient que la vue de face, car nous proposons un algorithme qui estime la pose pour des visages vus de face. En plus on utilise la version de l'algorithme Adaabost distribué dans OpenCV pour les visages vus de face. Les visages moyens obtenus pour chaque modèle, à partir des 37 visages utilisés pour l'apprentisage sont montrés dans la figure 1.19. Le vecteur moyen obtenu pour chaque région est de dimension 58×72, 88×42 et 86×28 respectivement.

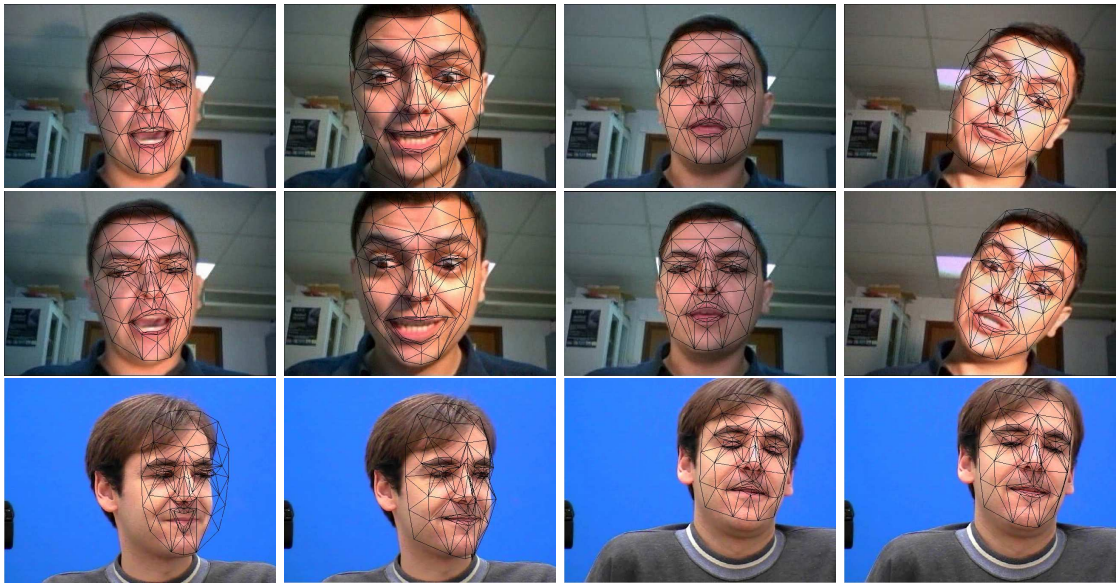


Figure 1.16: En haut: Images obtenues avec l'approche locale. Milieu: Images obtenues avec l'approche globale, En bas: Images obtenues de la séquence talking face avec l'approche locale et globale en alternance.

Figure 1.17: Exemples des visages utilisés pour l'apprentissage.

Figure 1.18: Visage sans forme ni expression.

La procédure utilisée pour faire l'initialisation automatique est illustrée dans la figure 1.20.

Elle se compose en plusieurs étapes. La première consiste a détecter la position d'un visage dans une image. Pour cela, nous faisons appel à l'algorithme Adaboost de Viola et Jones [86], qui est distribué avec la bibliothèque OpenCV. Avec cet algorithme on obtient une fenêtre. Avec cette fenêtre on fait ensuite une

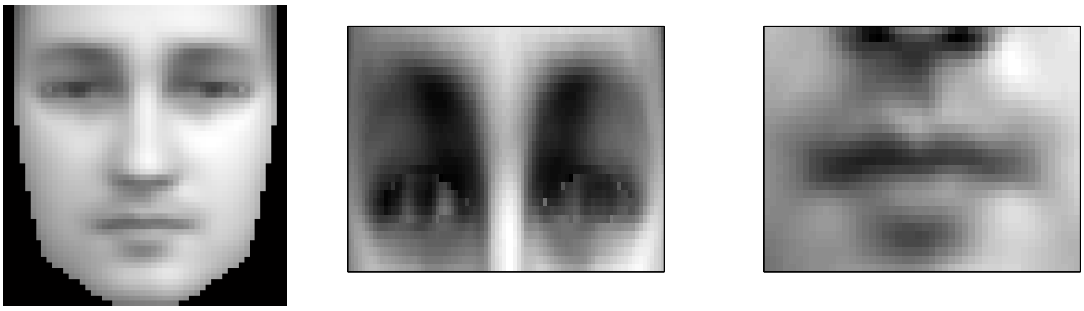


Figure 1.19: Visage moyen obtenu pour chaque modèle.

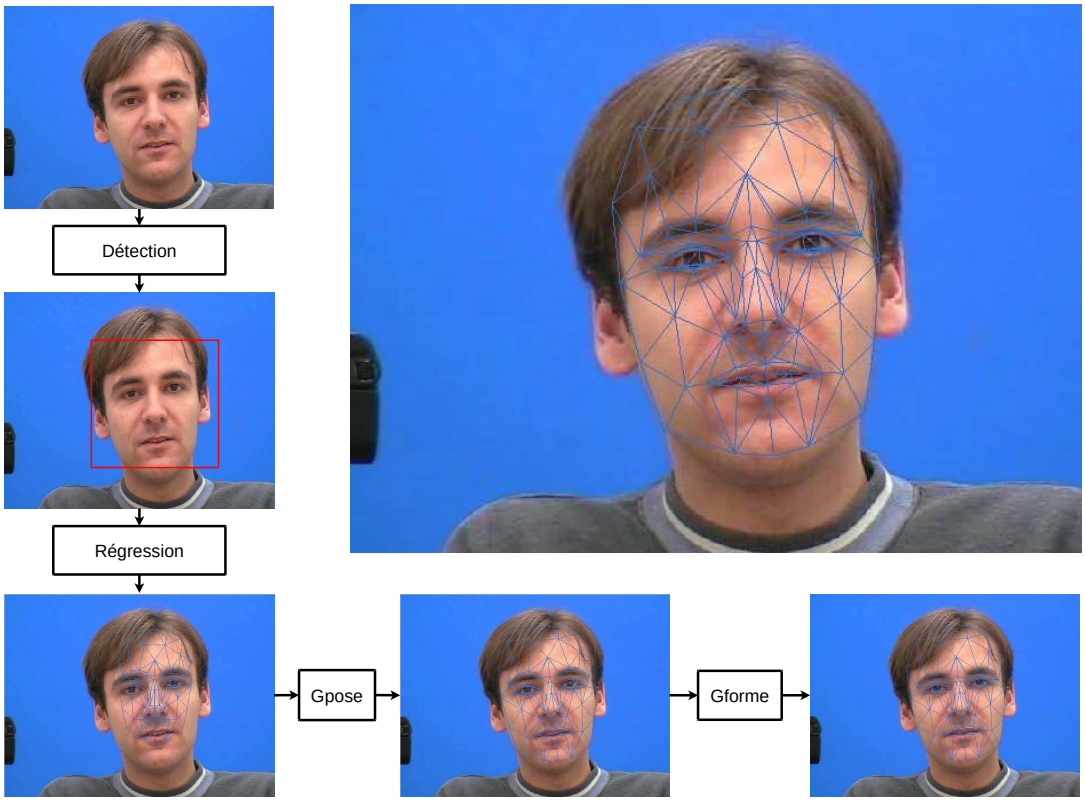


Figure 1.20: Procédure pour l'initialisation automatique.

régression linéaire pour obtenir les paramètres de translation du modèle *Candide*, tel que nous pouvons l'apprécier dans l'équation suivante :

$$[t_x, t_y, t_z]^T = \mathbf{A}\,[x_1, y_1, x_2, y_2]^T$$

Avec ces paramètres on place le modèle *Candide* dans la fênetre où se trouve le visage détecté avec l'algorithme Adaboost.

Une fois que nous avons placé notre modèle, nous utilisons notre algorithme pour faire l'estimation des paramètres de pose tel que nous le voyons dans l'équation suivante :

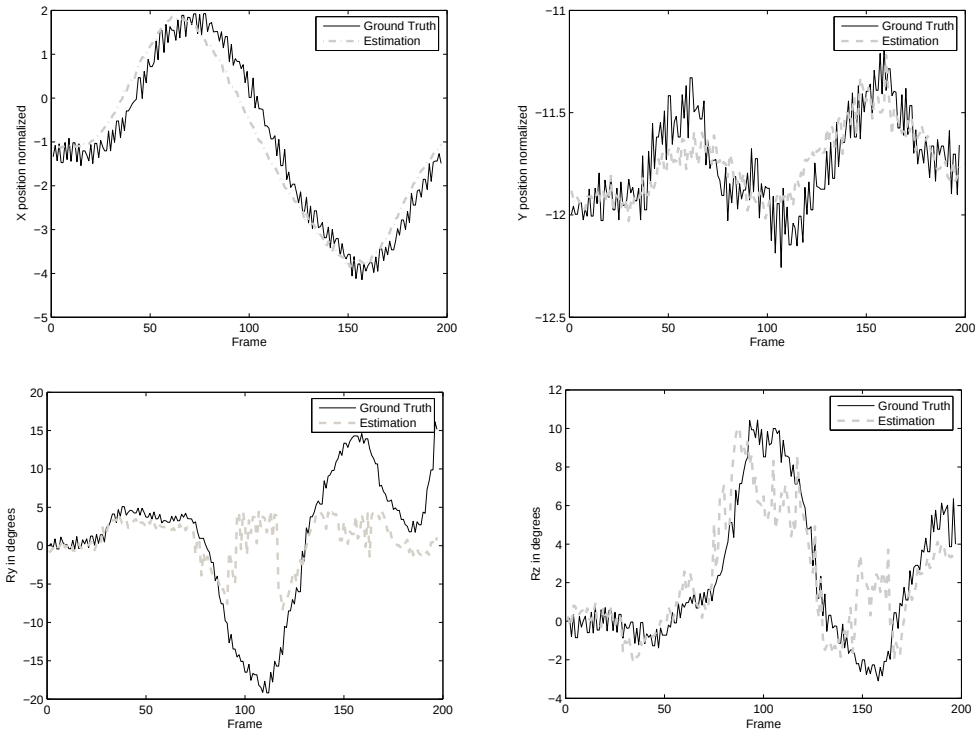
$$\hat{\mathbf{b}}_i = \mathbf{b}_{i-1} + \mathbf{G}_{\mathbf{pose}}(\mathbf{x}_i - \overline{\mathbf{x}})$$

Une fois que nous appliquons notre algorithme pour estimer les paramètres des poses du modèle *Candide* nous estimons les paramètres de formes appris pendant l'apprentissage dans un modèle indépendant des paramètres de pose. Dans ce cas, nous avons découpé en deux regions : une region pour la bouche et une autre pour les yeux :

$$\hat{\mathbf{b}}_i = \mathbf{b}_{i-1} + \mathbf{G}_{\mathbf{forme}}(\mathbf{x}_i - \overline{\mathbf{x}})$$

1.7.1 Résultats

Pour l'estimation de visages inconnus, on montre des résultats des séquences vidéo de LaCascia. On a vu que l'estimation de la pose dans chaque image était proche de la vérité de terrain fournie, mais dans le cas où on s'éloigne d'une vue frontale, l'algorithme ne parvient pas à suivre ces déplacements, comme on peut le voir dans l'image 1.7.1. On observe, quand la rotation est supérieure à 5 degrées, que l'algorithme ne parvient plus à estimer la pose de la personne. Cela est dû au fait qu'on utilise une plage plus réduite de points pour l'entrainement de cet algorithme qui va de -5 à +5 degrés. Ceci correspond à une vue de face, pour laquelle nous avons construit cet algorithme.



Ensuite nous avons utilisé la séquence talking face. Dans ce cas on voit qu'il existe un offset par rapport à la version de suivi, où l'adaptation du modèle est

effectué à la main et le plus soigneusement possible pour approcher les points du modèle *Candide* aux points donnés avec la séquence. On voit que la forme correspond aussi avec les sommets obtenus dans le cas de suivi. Ce qui nous montre que l'algorithme obtenu pour des visages inconnus est assez robuste, comme il peut l'être apprécié dans la figure 1.21.

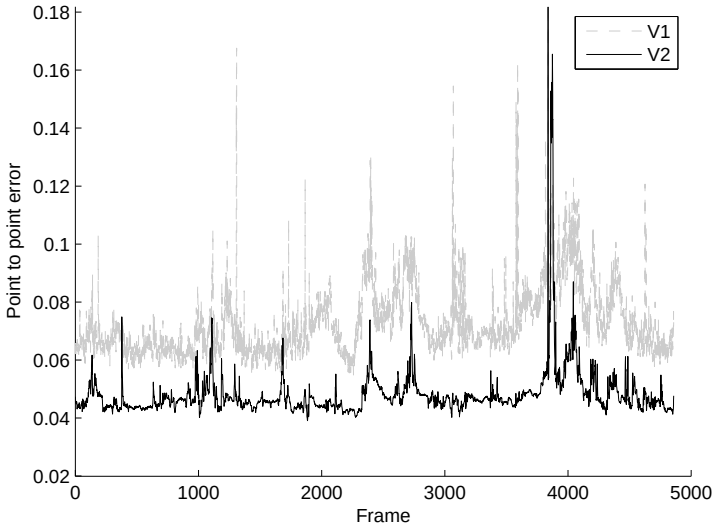


Figure 1.21: V1 : visage inconnu. V2 : suivi basé sur une seule image.

1.8 Un algorithme incrémental

L'idée principale sur laquelle se base notre algorithme incrémental est sur le fait que si à un instant t on a des nouvelles données qui arrivent, et que l'on veut ajouter ces données à une matrice déjà existante, on peut faire une mise à jour de la SVD de cette matrice tout en conservant la SVD déjà estimé à l'instant $t-1$.

$$\mathbf{A}^{(total)} = \left[\begin{array}{cc} \mathbf{A}^{(old)} & \mathbf{A}^{(new)} \end{array} \right] = \mathbf{U}^{(total)}\mathbf{D}^{(total)}\mathbf{V}^{(total)T}$$

Pour appliquer cette idée nous avons utilisé l'algorithme de la SVD incrémentale décrit dans [72], de façon qu'on effectue la mise à jour de la SVD en terme de la SVD estimée précédemment :

$$\mathbf{U}^{(total)} = f_1(\mathbf{U}^{(old)}, \mathbf{U}^{(new)})$$

$$\mathbf{D}^{(total)} = f_2(\mathbf{D}^{(old)}, \mathbf{D}^{(new)})$$

$$\mathbf{V}^{(total)} = f_3(\mathbf{V}^{(old)}, \mathbf{V}^{(new)})$$

Alors, comme la méthode pour résoudre la CCA est basée sur la SVD des deux matrices de données, nous allons utiliser la SVD incrémentale pour calculer les coefficients de la CCA de façon incrémentale, pour faire la mise à jour des coefficients.

On applique alors la SVD incrémentale aux matrices $\mathbf{A}_1$ et $\mathbf{A}_2$ et on fait la mise à jour des coefficients de la CCA de la façon suivante:

$$\mathbf{W}_1 = \mathbf{U}_1^{(total)} \mathbf{D}_1^{-1(total)} \mathbf{U}^{(total)}$$

et

$$\mathbf{W}_2 = \mathbf{U}_2^{(total)} \mathbf{D}_2^{-1(total)} \mathbf{V}^{(total)}$$

1.8.1 Résultats

Pour l'algorithme incrémental, on a utilisé les séquences vidéo de Lacascia, néanmoins, du fait qu'elles n'ont que 200 images, aucun résultat important n'a pu être obtenu de ces images. En revanche, dans le cas de la talking face, on voit une légère amélioration de l'erreur moyenne obtenue autour de l'image 1400. Cependant, visuellement, les résultats entre l'un et l'autre algorithmes sont identiques.

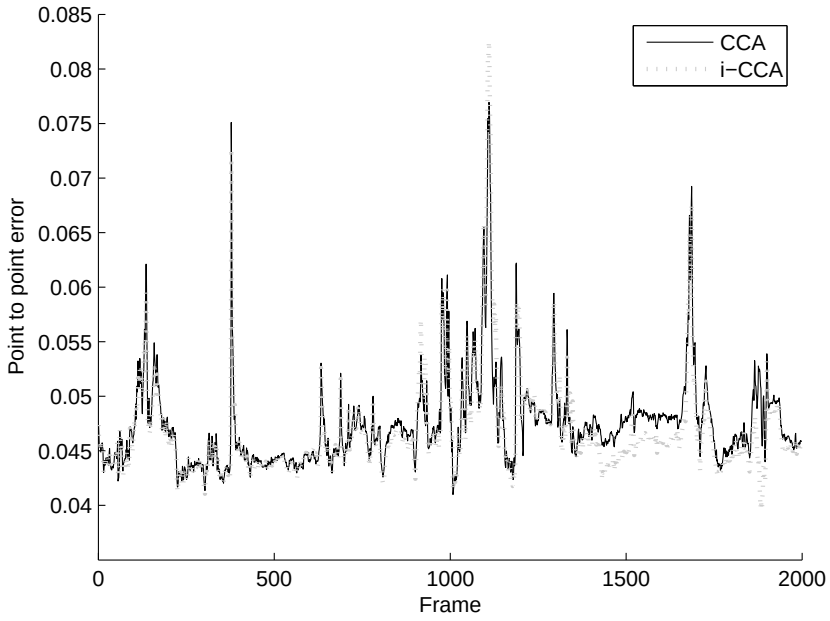
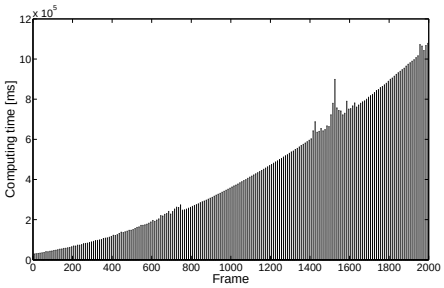
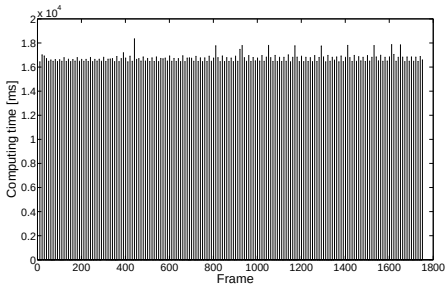


Figure 1.22: Erreur moyenne point par point.

En utilisant l'algorithme incrémental





1.9 Conclusion et perspectives

Ce travail de recherche nous a amené à développer un algorithme de suivi particulièrement robuste aux mouvements de rotation comme aux mouvements de translation du fait qu'on a ajouté deux vues de profil synthétisées au vecteur d'observation. De plus, nous estimons le modèle de suivi pendant la phase d'apprentissage contenu dans une matrice $\mathbf{G}$. Cette matrice $\mathbf{G}$, qui fait le lien entre les vecteurs d'observation et les paramètres de suivi des poses, est conservée sans aucun changement pendant tout le suivi. De ce fait, l'algorithme est très rapide.

Dans toutes les implémentations, les résultats simplement visuels sur les séquences vidéos semblent identiques. Néanmoins si on compare avec une vérité terrain nous avons vérifié l'intérêt de décomposer le visage en plusieurs "sous-modèles", dans notre cas : trois.

Sans ajout de mécanisme ou modèle spécifique pour prendre en compte les changements de variations de lumière ou d'éclairage, notre algorithme est relativement robuste. Egalement, les mouvements de caméras ne semble pas perturber le processus. L'algorithme qui fait le positionnement du modèle *Candide* automatiquement sur les visages inconnus est relativement robuste et améliore donc considérablement la détection de la pose et de la forme du visage. En continuité de ce projet, il serait possible d'accoupler l'algorithme d'initialisation automatique avec le suivi, ainsi il serait possible d'obtenir la pose automatique du modèle sur le visage à l'initialisation du processus sans l'intervention d'un utilisateur. Alors il serait possible d'utliser cet algorithme en cas d'occultation, gêne ou masquage partiel du visage et n'importe quel autre facteur qui pourrait entraîner la perte du suivi.

La version incrémentale de notre algorithme possède de nombreux avantages. Notamment, en exploitation, il s'enrichit sans cesse dans l'apprentissage et l'acquisition de données nouvelles relevées au fur et à mesure du traitement de nouveaux visages caractéristiques. Autrement dit quand l'algorithme découvre un visage pour lequel l'apprentissage à été incorrect alors, après correction, il peut être ajouté à la base de données.

Chapter 2

Introduction

Object tracking is an important part of modern complex systems. It permits to enhance the performance of a person in doing some specific labor (augmented reality or people identification), to make tedious labors automatically (counting vehicles passing in a crossway), to locate persons in the crowd, vehicles in the traffic jam or stolen vehicles, following merchandise, etc. To do this, there exists a panoply of features that can be tracked from the interest object, as following sounds, in the case of people following the voice, or electric signals from devices fixed to, or making part of the object, and more recently from video sequences. The latest have developed in the last decades with the growing computer power and with the reduction in cost and size of digital video cameras, especially because the video vision offers a non-intrusive solution. In consequence, many applications based on video object tracking have seen the day, as face-based biometric person authentication, human computer interaction, teleconferencing, surveillance, augmented reality, behavior understanding, vehicle tracking, etc.

The problem of object tracking in a video sequence can be defined as knowing the object's location relative to the camera, its shape and trajectory from frame to frame. To do this there are three processes involved in object tracking: track initialization, track update, and track termination [13].

- ⋆ Track initialization consists in saying whether or not there is an instance of the object we want to detect, and if it is the case, to determine its location.

- ⋆ Track update consists of two components: Object detection and finding the correspondences between the detected objects across frames, sometimes called filtering step. These two tasks can be performed separated or jointly. For the first case, possible object regions in each frame are obtained by means of an object detection algorithm, which can be the same algorithm used for initialization, and then the tracker looks for correspondences of the objects across frames. In the second case, the object region and the correspondences can be estimated jointly by iteratively updating the object location obtained from previous frames [100]. The objective of the filtering component is to add temporal continuity constraints across frames and to deal with dynamics of the tracked object, as in [7; 42; 59; 61; 65; 82]. The

principal advantage of using this component is that in case of complete occlusions, we can estimate the object position as a function of the dynamic model, or limit the wrong estimations to go far away from the real position of the tracked object.

⋆ Track termination consists in detecting when a tracked object gets out off the camera visual region.

To deal with the tracking problem, vision research groups have used in the last years several approaches. These approaches can be classified in terms of the object's description, in terms of the object's feature used for the tracking, or in terms of the method used for tracking. Some examples of the object's description are points, primitive geometric shapes, textures, colors, etc. For the features used to perform the tracking we can find edges, gray level intensities, color histograms, interest points, etc. Finally for the tracking methods we can find deterministic ones, like gradient descent based methods, Kalman filters, and stochastic ones, like the particle filter also known as Sequential Monte Carlo (SMC) algorithm, and learning-based approaches. In chapter 3 we will talk about these characteristics of a tracker.

The nature of the tracked object is of great influence in the choice of the tracker. If we want a general tracker, we will choose to track for example a specific color present on the desired object. However, if we want a specific tracker, we can introduce a geometric or a parametric model that will perform better for tracking this kind of objects but that will be useless if there is an object that can not be described with our model. This model can be a 2D or a 3D model and depending on that we can track our object not only as a 2D projection over an image but also in real world coordinates. In our case the object of interest is the human face.

The face is a very complex three dimensional surface, that is flexible, usually contains creases, and it has color variations, and it conveys many pieces of information such as identity of a person, the gender, his age, the relative position with respect to the camera. It is because of this that faces have aroused the interest of computer vision community, besides they are very familiar to us, i.e., we have a well developed sense of what expressions and motions are natural for a face. So, human face image processing is currently an active research area in the computer vision community. There exist several classes of image processing, being some of the most important nowadays[3]:

⋆ Face Detection: It consists in determining the presence of faces in images, and the localization of it. For the case of localization, several parameters can be obtained depending on the representation of the face, going from a simple window containing the face, to 3D pose parameters as well as expression parameters, as for the case when a 3D geometric model is used. A face detection algorithm is vital for most face image processing applications. Face detection has been the subject of research by numerous groups as it lies at the core of many applications.

- ⋆ Facial Feature Extraction: The principal features that can be obtained from faces are specific points and contours. These features can be used to determine the shape, motion and expression of the face. To obtain these features, it is necessary to know the location of the face.

- ⋆ Face Tracking: Tracking of a face is the process of following a detected face or extracted facial features through an image sequence. The algorithm used in this process can be identical to face detection or facial feature extraction (by applying those techniques on each image in the sequence). However, as previously explained we can perform jointly the detection and tracking by taking into account the knowledge acquired from the earlier image(s) in the sequence. There is a panoply of applications behind face tracking, like biometrics, human-computer interactions, behavior analysis, driver monitoring systems, marketing and advertising, interactive information system, computer games, etc.

- ⋆ Face Synthesis: Face synthesis is the task of rendering (moving) images of faces from a set of parameters. Recently face synthesis has been used for face tracking and face detection, as the Active Appearance Models and 3D Morphable Model. The applications are several, going from video coding to multimedia modeling and cinema special effects. In order to do that a parametric model should be available and it will fix the quality of the results.

- ⋆ Face Recognition: Given the location of a face in an image, the face recognition task consists in identifying a person or verifying the person's identity (face authentication).

2.1 Contributions and publications.

In this thesis, we will present a method for tracking faces. This method uses a parametric 3D model to estimate the 3D pose and the facial gesture parameters. It uses the Canonical Correlation Analysis (CCA) as a supervised learning method that finds the regression parameters between the model parameters and the residual between an input vector and a reference image. We extend also this method to estimate the 3D pose and the facial shape parameters for initializing, to perform automatically the 3D model's initialization. Finally, we will present an incremental method to estimate the CCA coefficients as the data arrives.

We can compare our work with the work realized in [25; 30; 31], where the *Candide* 3D head model is also used to track people's 3D head pose and facial expressions, but in their case, the tracking method used is based on a gradient descent to determine the perturbations. Our approach can also be compared with the Active Appearance Model described in [21; 22; 23; 32]. For the AAM's, they use a linear regression to estimate the relation between the parameters of a model learned from a training set, and the difference between a synthesized image and

the input image. To obtain the model, they use a set of multiple manually labeled images. Then they train this model to estimate 2D parameters, represented by horizontal and vertical translation, scale and rotation. In [29] they improve this algorithm by using the Canonical Correlation Analysis instead of the linear regression to learn the relationship between the parameters of a model and the difference between a synthesized image and the input image. Finally, we can compare our work with the 3D morphable models explained in [68]. In this case 3D scans are used to obtain a general model to describe variations of illumination, pose, and facial gesture. This method performs also the analysis by syntheses, where the input image is compared with the synthesized one until there is not improvement of the error.

The principal difference is that in our work we do not estimate a shape model from the training images, but we used the *Candide* 3D geometric model that we adapt for the person we want to track. We use the 3D model to create a synthetic training database containing perturbations based on the first image of the video sequence. We use the Canonical Correlation analysis to obtain a linear relationship between the 3D pose perturbation and facial gesture parameters, and the difference between a reference normalized patch and the patch obtained from the current image at the last known position. We will show that our proposition performs well in estimating the 3D pose and up to eleven facial gesture parameters using for training the first video frame that is manually initialized. It is important to remark that our work uses the input image to create the normalized patch contrary to analysis by syntheses schema used by the AAM's and the morphable models.

Another important contribution of our work is the development of an incremental algorithm for the Canonical Correlation Analysis, based on the incremental Singular Value Decomposition (SVD) proposed in [10; 57; 72]. Although this algorithm is slower than the CCA, because we update the CCA coefficients every m frames, it has the advantage of adding new information as the tracking evolves.

Finally, we have extended this work to 3D face and shape detection. To do this, we use the Adaboost algorithm to detect the 2D window containing a face, and then we use an algorithm similar as the one used for tracking, but training over a database of several people, to estimate the 3D pose parameters and the shape parameters of the face.

We restrict this work to the use a single fixed webcam, because multi-camera approaches require calibration of all the cameras and stereo cameras are more expensive than traditional webcams, and moving cameras are not only more expensive but they require an additional alignment as explained in [90]. We have tested our tracking algorithm with moving cameras and the results are encouraging.

The structure of this thesis is the following: in chapter 3 we will present the object's representation used for tracking, the most important features that are used for this, and finally the most important methods used for tracking objects, and particularly faces. In chapter 4 we will present our algorithm published in

[96], where we compare the use of a linear method based on the Canonical Correlation Analysis and a non-linear method based on the Kernel Canonical Correlation Analysis. In this case we limit the tracking to the pose parameters only. In chapter 5, we extend the algorithm to track not only the pose, but also the facial gesture, as explained in [98]. Then we propose a local based approach that uses small patches around some facial regions of interest to track the 3D pose and facial gesture, as described in [99]. Finally we compare both methods and study the robustness of the algorithm with respect to noise in the pose estimation, as described in [97]. In chapter 6, we propose an algorithm that estimates the 3D pose and the shape of faces in images. This algorithm is a complement to the tracking algorithm, as it can be used for automatic initialization and recovery. The principal advantage of this algorithm is that the model is general enough to work with persons that are not present in the training database. Finally, in chapter 7 we present an iterative algorithm to update the CCA coefficients as new data arrives. Our motivation is to improve the efficiency of our tracking algorithm.

Chapter 3

Tools and Methods for Object Tracking.

Object tracking through the frames of a video sequence is required in many applications of computer vision, such as augmented reality, surveillance devices, biometrics, visual servoing of robot arms, computer-human interfaces, smart environments, meteorological and medical imaging, etc. Moreover, with the always increasing computing power and the cost and size reduction of high quality cameras, computer vision offers solutions that are cheap, practical and non intrusive to the tracking problem. This has developed the use of video analysis not only to track objects, but also to use the object tracks to recognize their behavior.

Tracking an object in a video sequence can be defined as knowing the object's location relative to the camera. To do this, there are two processes involved in object tracking. Object detection and finding the correspondences between the detected objects across frames. These two tasks can be performed separately or jointly. For the first case, possible object regions at each frame are obtained by means of an object detection algorithm and then the tracker looks for correspondences of the objects across frames. In the second case, the object region and the correspondences can be estimated jointly by iteratively updating the object location obtained from previous frames [100].

The approach that will be used to track a specific object must take into account the characteristics and degrees of freedom of the object and the camera, and the features that we want to track of the object for a specific application. This means that the combination of tools used to accomplish a given tracking task will depend on whether we want to track a kind of objects, like cars, people, faces, etc, groups of these objects, like black cars, walking people, persons, or unknown objects with a particular characteristic, like moving objects in a security area. The problem of tracking can be further simplified if we have more priors about the object, like a velocity model, an acceleration model, etc.

For any case, it will be necessary to compare the input video frame with a given representation that describes the object of interest. This representation can be based on image points, on contours, on geometric models, on appearance, on skeletal models, on articulated shape models, on geometric models, etc. All

these approaches can refer us to a 2D tracking or to a 3D tracking. 2D tracking consists in following the projection of the 3D movements of an object by means of a transformation. This kind of tracking requires a model that can handle with the appearance changes due to perspective effects, self occlusions and deformations. However this kind of tracking can not recover the true position in the space. In the other hand, 3D tracking aims at recovering the six degrees of freedom that define the position of an object with respect to the camera [52], as well as the degrees of freedom that describe a non-rigid tracked object.

Different approaches exist for tracking moving objects, two of them being feature-based and model-based. Feature-based approaches rely on tracking local regions of interest, like key points, curves, optical flow, or skin color [24; 54]. Model-based approaches use a 2D or 3D object model that is projected onto the image and matched to the object to be tracked [31; 48]. These approaches establish a relationship between the current frame and the information that they are looking for. Some popular methods to find this relation use a gradient descent technique like the active appearance models (AAMs) [22; 91], others a statistical based technique using support or relevant vector machines (SVM and RVM) [5; 88; 89], or a regression technique based on the Canonical Correlation Analysis (CCA) (linear or kernel based).

In this chapter we will present relevant tools and methods for tracking. To do that, we will start showing some of the possible ways to represent the object that we want to track. An important characteristic for tracking is to determine if we are working with a rigid or non-rigid object. This will be determinant to choose the object representation. Moreover, sometimes we can approximate deformable objects with rigid models, as it is the case of faces. After these object representations, we will talk about the more common features that the computer vision community has used to track these objects. Finally we will present some of the principal methods used to perform the tracking task.

3.1 Representation of an object.

The representation of an object is of great importance for the choice of the tracking approach that we will use, because some representations that are robust for a kind of object can be useless when we use them to track another kind of object. This is why the first important thing to do when implementing a tracker is to choose how to represent the tracked object. Before introducing the most common object's representation, we will talk about rigid and non rigid objects, because this property of the objects will determine the choice of the way to represent it. After that, we will present some of the most common representations of an object by means of points, of primitive geometric shapes, of silhouette and contours, of models, and of appearance.

3.1.1 Rigid and deformable objects.

Tracking rigid and non-rigid objects are two very different tasks, especially from a complexity point of view. Rigid objects present the advantage of having only six degrees of freedom when working in a 3D space. Some examples of rigid objects are cars, detached pieces in a production line, trains, etc. In the other hand, non rigid objects have more degrees of freedom depending on the object and the degrees of freedom inherent to it. In addition, the deformation that can be present for a non rigid object, can be confused as a 3D movement if it is not taken into account. Some examples of non rigid objects are living beings, especially persons. It is because of this difference in complexity that the first thing that we should know about an object is to classify it in either of these categories.

In [90] a method for determining if an object is rigid or non rigid in a video sequence is proposed. The central idea to distinguish between this kind of objects is that over short time intervals, the appearance of rigid objects, under viewing conditions similar to orthographic projection, changes more much slowly than this of most non-rigid objects. In their case, they have compared humans and vehicles. The results reported show that the variations in non-rigid objects, walking humans in this case, present a periodic motion with sharp peaks, while the rigid objects vary in a more stable curve. However, as they compare pixels to pixels within a motion window, movements toward or away from the camera are reported to present an unpredictable behavior.

Some examples of works tracking movements of non-rigid objects can be found in [18; 67; 74]. In [67] they propose an algorithm to track deformable objects of any kind, based on feature points and classification trees. Given an input image, they look for correspondences between the features learned of a given object. They use 2D meshes that are deformed in order to follow highly textured surfaces, and can cope with a high number of outliers. In [18], points are matched between brain MRI images to find geometric correspondences of these points, and in synthesized 2D objects. They propose an approach that takes soft decisions. This means, instead of discarding a relation between two points, it assigns a score that will be improved in further iterations. Finally we can talk about the work presented in [74], where the author proposes the active blobs. He uses a texture-mapped triangular mesh model for tracking deforming shapes in color images. It uses a gradient approach to estimate the variations.

3.1.2 Points.

We can represent an object by a point or by a set of points. In the case of a single point, it can represent the centroid of the object, and this kind of representation is generally used when the object to be tracked is small with respect to the image. This kind of representation usually models translations. We can use this representation for both, rigid an no-rigid objects as the object represents a small area of the image. Examples of this are presented in [84] for a rigid turning object with points drawn over its surface and tracked independently, and in [85] for points of

faces, and in this case also each point is tracked independently of the others. For the case of multiple points, they can be interest point distributed over the object to be tracked. In this case the points have a geometric relation between them. One example for rigid objects can be found in [54] where interest points are detected and compared with a geometric model to determine the 3D pose. In [67] we can find the extension to deformable objects of a similar tracker. The use of a relation between several points permits the estimation of translations and rotations in 2D and 3D spaces, as well as the estimation of non rigid movements. One advantage of using a set of points is that we are not suppose to detect all points, but the maximum number of them that gave a valid configuration, making this kind of approaches robust with respect to outliers.

3.1.3 Shapes.

In this case the object is represented by primitive geometric shapes, like rectangles, ellipses, etc. This kind of representation can be used to estimate rotations, translations, affine, or projective transformations, but in a 2D space. More complex representations can be achieved if several of these geometric shapes are used together to represent a specific kind of object. In [102], the authors use ellipses to represent the human body. In [47] moving objects are tracked with rectangles across multiple non overlapping cameras. Also a rectangle is used in [62] to track hockey players. In [46] a polygon is used to represent the persons in a particle filter context.

3.1.4 Silhouette and contour.

The boundary of an object is the contour. The region inside the contour is the silhouette. Contour or silhouettes are suitable for tracking complex non rigid shapes. To track these non-rigid objects active contour models, also known as snakes, are used. Nevertheless the presence of partial occlusions can diminish the tracker performance.

In [14] the authors perform a segmentation and look for regions that are classified as foreground. Then they use these regions to match previously detected objects. In this case the motion blobs are used as silhouettes to represent moving cars. In [75] a machine learning approach is used to detect texture transition and then the contour is obtained to track rigid and non-rigid objects. In [1] the authors use a relevance vector machine to estimate the pose of a human body represented in terms of silhouettes.

3.1.5 Geometric models.

Another common method used for representing objects is by means of a model. This model approximates the geometry of the object, as for instance, boxes for cars, ellipsoids for heads, or a geometrical model that can take into account the possible deformations of non-rigid objects. For the particular case of non-rigid

objects there exists also the use of articulated models and skeletal models, where every part of the object is represented by simple geometric shapes that are held together with joints. For example, the human body can be represented by ellipses, or cylinders, for the torso, head, and limbs. These models can be used for 2D or 3D movements depending on the application.

For the case of rigid objects we can cite [54], where planar objects as the sail of a ship and the face of a book are represented by a 2D model, or a box and a human face are represented by means of a 3D model. We can also cite [53], where the 3D model of a teddy tiger is obtained from several images, all of them used for the training of a classifier. Another example of rigid object tracking is the work of [78], where three car models are used to determine the vehicles that pass through a certain road by means of a Monte Carlo approach. As an example of non-rigid objects we can cite the 2D model presented in [45], where a shape model is used to track the contour of a hand by means of a Monte Carlo approach. In the shape model used they implemented twelve degrees of freedom for the fingers and possible hand movements. In [67], the authors use features of the image to determine the shape of a deformable object, in this case a t-shirt with a logo. In [101] the authors use a skeletal model to represent the locomotion of walking people, in order to decide whether or not a person is walking, running or standing in a video sequence.

The principal advantage of using models to represent objects is that we can be very discriminative, and add some robustness to the tracker when facing occlusions, outliers, changes of illumination, etc. However, this kind of approach limit the tracker to a very specific kind of object, and had to be initialized and well adapted to the application that it is intended to do. They are also expensive in computational cost, because of the transformations that the model can be subject to, as scaling, rotations, translations, and deformation in the case of non rigid objects.

Human face models will be explained more precisely in the chapter 4, where we will present the parametric model used in our approach.

3.1.6 Appearance models.

It is possible to represent the appearance of objects in several ways. One of the most popular methods consists in using the active appearance models, where shape and appearance are combined to track faces. To do this, a training database containing manually placed landmarks is used. The shape and the appearance are associated by means of the principal Component Analysis [20; 21; 22; 23; 32]. Another way to represent the appearance is by means of templates, which are simple geometric shapes containing spatial and appearance information. However, this kind of approaches is generated for a simple view, limiting it to objects whose appearance does not change during the course of tracking. In [72], the authors propose an algorithm that makes an update of the template, allowing the tracking of complex object under varying conditions.

3.1.7 Others.

It is important to say that a combination of these representations can be used as in [50], where an articulated body model of 10 joints and 14 segments is used to represent the torso, the limbs and the head, having 32 degrees of freedom. To estimate and adapt a human body model, the silhouette is used, and a head shoulder contour is used as a template to locate the head from the silhouette obtained. This information is combined by means of a particle filter.

As we have seen, there exist several ways to represent an object, and depending on the application and the characteristics of the object and on the tracking information needed, we have to choose between these multiple representations.

3.2 Features used for tracking.

The feature selection, as well as the object representation, plays a critical role in the implementation of a tracker. Both are closely related, because the choice of a way of representing an object will limit the features that we can use to track the object. However, many of the recent tracking approaches use a combination of features in order to have more robust results. In this section we will present the most important features used for tracking objects as well as some of the advantages and disadvantages of using them.

3.2.1 Color.

Color is a feature that is frequently used to track objects. There are several advantages of describing an object by a color. It lets us track any kind of object independently of its geometry, if it is rigid or not. It is robust with respect to self occlusions when strong pose variations are present. Color based processing is often faster than processing other features. Finally, under certain lighting conditions, color is orientation invariant, which means that the motion model required will consider only translation parameters and scale. However, the principal disadvantage of this kind of tracking arises when a same color object moves close to the object we are tracking, or even worse, when the background has a similar color.

To describe a color, the more usual approach is to estimate a reference color histogram of a fixed-shape window, as those depicted in figure 3.1. Then, we will look for scaled windows of the same shape that contain a color histogram close to the reference. Typically the Bhattacharyya distance is used in order to determine whether or not a color histogram is close to another one. In [65], the authors use a Monte Carlo approach that uses color as the principal cue to track objects in 2D. They apply a kind of geometry model that divides the fixed-shape window in rectangles where each rectangle has a particular reference color histogram. They use then the color histogram distances of the sample windows to obtain the color likelihood for the particle filter they use. The principal advantage of using this

kind of approaches, as explained in 3.3.3, is that the tracker is more robust to distractions and to occlusions, even if they are partial or total. Color histograms have the property of being stable object representations unaffected by occlusions and changes in view, and they had been used to differentiate among a large number of objects.

We can also use the a priori knowledge of the tracked object, to look for particular colors, like skin's color, as in [94; 95], or more specifically lips' color as in [34] by means of transformations of the color input image.

In [94] we can find an extensive report presenting a statistical skin-color model and its adaptations to different lighting conditions. An important remark of this report is that tracking a specific color, such as skin color presents several problems. For example, the color representation of the skin obtained by a camera is influenced by many factors as ambient light, movement, etc. Moreover, different cameras produce different color values even under the same external conditions. Finally skin colors are different from one person to another. Nevertheless, the authors use their approach to track faces and report a good performance in real time, with 30 frames per second with 305×229 input sequences.

In [34] the authors propose a color transformation for lips segmentation. The motivation of this work is that skin and lip colors are more characterized by chromatic than by brightness components. That means that under varying illuminations, chromatic skin features are relatively constant. Thus, the proposed transformation separates the lips from the skin. One application of this kind of algorithms is for lip reading in noisy environments.

It is known that the RGB representation has the disadvantage of not being uniform from a perceptually point of view [34], and also there is a high correlation between the different components of this space, which makes the images stored in this format to have a big size. This is why other color spaces have been proposed, as the Lab, HSV, YCrCb. Lab is a space created to be perceptually uniform, with dimension L for luminance and a and b for the color-opponent dimensions, while HSV (hue, saturation, value) is another space created to be perceptually uniform. These two spaces are reported to be more sensitive to noise. YCrCb is a transformation of the RGB space where the Y component stands for the luma, and it contains the most of the image information, that corresponds to a grayscale image. The Cr and Cb stand for the red and blue chroma components. This space was designed to compress digital images. Frequently, we transform the color image to a grayscale image, by means of this transformation, and we keep only the Y component. Depending on the object to be tracked, the choice of one or other space, or even other particular transformation will be used.

An important algorithm used for color segmentation and color tracking is the mean shift algorithm. It was first proposed by Fukunaga in [35] and then resumed by Cheng in [17]. It is a non parametric approach for the estimation of gradient of density function and it was developed to find modes in a set of data samples. Although the method has been applied for clustering, segmentation, background subtraction, more recently it has been applied to track objects based in the color histograms and the Bhattacharyya distance. In [19] the authors use

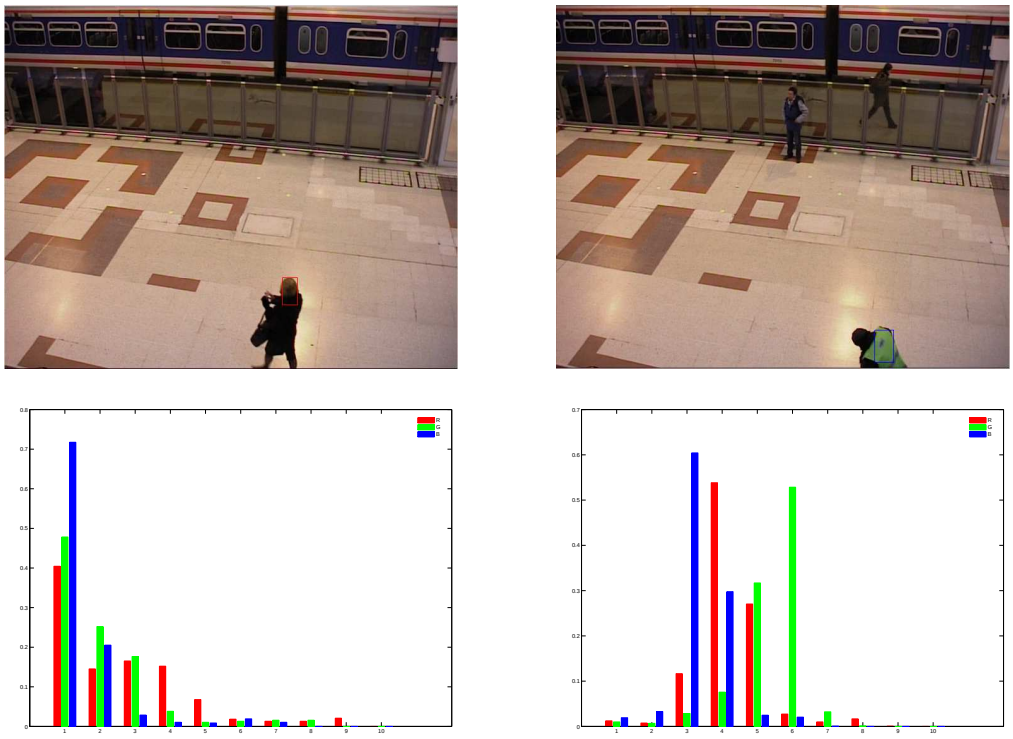


Figure 3.1: Top images: Two rectangles containing a color reference. Bottom images: Corresponding color Histograms.

the mean-shift algorithm to track faces, hands, football players, people in the subway, and a mug. They probe the robustness of this approach and it convergence properties. In [16], the authors use the mean shift algorithm as a key part of an autonomous vehicle navigation to detect the road. They use the mean shift algorithm to segment the incoming image, and then, they apply a threshold to discriminate between the road and the other objects present in the image.

3.2.2 Motion.

Motion tracking can be described in a simplified form as detecting changes between images. These changes can be produced due to illumination changes, from moving objects, or from camera movements. This means that motion-based trackers tend to be simple and fast, but they do not guarantee that the tracked regions have any meaning.

One widely technique used for change detection, due to its simplicity, is frame differencing, which consists in subtracting two images and obtaining a binarised difference map. These two images can be the incoming frame with a background image or two or three consecutive frames of a video sequence. The resulting frame difference will classify then those pixels presenting a significant variation into foreground for the first case or as motion blobs for the second, as can be

seen in figure 3.2. The difference map is generally binarised by thresholding it at some predetermined value to obtain a change/no-change classification. Thus, the value assigned to thresholding is of great importance, because giving a very low threshold will include spurious changes while a high value will suppress changes [71].

Background subtraction is a core component in applications where stationary cameras are used, and where the background scene is a static structure. In this case one has to obtain a representation of the background, or reference image, update this representation over time and compare it with the input image. The reference image is a frame of the sequence where no foreground objects appear in the scene. This kind of approach is not always practical, because for some applications a reference image without foreground objects is not available. Moreover, if the background corresponds to an outdoor sequence, the principal problems are due to illumination changes and oscillating objects like trees. To cope with that, recent works have modeled the background as a dynamic image. In [92] and in [79] the authors model the background as a mixture of adaptive Gaussians, in order to cope with global and local illumination changes, positions of the light sources and moving objects in the background of an outdoor scene. The principal advantage of this approach is that it uses several Gaussians in order to describe multiple backgrounds, and it allows long standing objects to become part of the background without forgetting the last background, and then, when the objects moves, recover the original background. In [61] the authors propose a method for modeling the background dynamically by means of the principal component analysis and an autoregressive model. They use an iterative PCA to describe all the possible background's movements by predicting the following frame.

In [12], to update the background, the authors propose an iterative process that assigns a probability to 5×5 pixels' window to belong or not to the background by means of an error map that takes into account changes with respect to previously computed background and with respect to the previous frame.

In [42] the pixels of some regions are considered as clusters and used to predict the object motion, where adjacent clusters following the same trajectory are considered as an object hypothesis. However, they consider only parallel movement, which restricts this particular tracker to rigid objects that do not present rotating movements about the optical axis. This algorithm was used for cameras on moving vehicles to detect other vehicles and to detect pedestrians. In this case, they did not consider a preprocessing for the case of a moving camera, but sometimes the blob detection may be accomplished using background alignment techniques and change detection algorithms as in [90], where a three image differencing method is used to detect motion blobs when the camera moves. However, using consecutive images can pose several problems when the the object is not sufficiently textured.

In general one important problem of this kind of tracking is when there are multiple objects, because there are four important tasks that should deal with: appearance, disappearance, splitting and merging. To deal with them, in [7] the

authors add a kinematic model as well as color model for each target, in order to discriminate them after a merging process or a splitting.

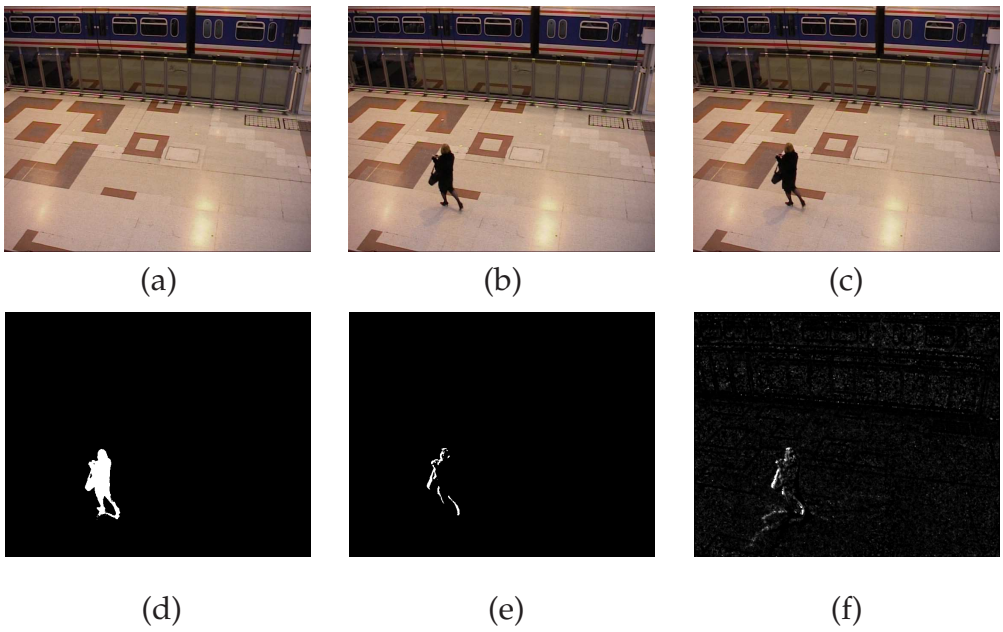


Figure 3.2: (a)Background image, (b-c) two consecutive frames of a video sequence, (d) resulting motion Map of the difference between the background image and a frame,(e) resulting difference map between two consecutive images, and (f) resulting difference map between two consecutive images taking into account the gradient of the image.

In [44] the authors present a general study for tracking motion of multiple objects without the assumption of motion constancy, this means, they consider noisy images due to camera vibrations and noise added due to the quality of the camera. The motion estimation they propose takes into account the fact that between two frames, the difference will not give the moving objects but the contours of these moving objects, and will consider that pixels lying inside the object are static.

In [59] the authors propose a 2D appearance-based approach, that fuses motion and the obtained model, and that uses Kalman filters to robustly predict the position, motion and shape of the desired object.

3.2.3 Contours.

The contour of an object is another characteristic that is often used for tracking. Object boundaries usually generate strong changes in image intensity. An important property of edges or contours is that they are less sensitive to illumination changes compared to color features or motion. For this feature, the objective is to track outlines, from foreground objects, that are modeled as curves, while they

are moving through clutter. This clutter makes the problem more challenging, because elements of the background clutter may look like parts of the tracked object. What can be worst, the background could be a collection of objects like the one being tracked, like for example, a person being tracked when passing a crowded place.

In [45], the CONDENSATION algorithm is used to track object's contours. To have a more robust tracker, the authors use models of shape and motion, in order to obtain a more robust tracker, but they do it taking particular care of not losing generality of application for this model. They train the tracker with sequences where there is not clutter and once they have estimated the coefficients of their models, they use them in sequences where clutter is present.

The principal problem when working with contours is that they must be applied in cases where there is a good contrast between the object to be tracked and the background. They are also sensitive to highly textured surfaces and clutter. To deal with these problems, in [75], the authors propose a model-based method to track contours of rigid and deformable objects by means of a machine learning approach. With this method they estimate the conditional probability of a texture discontinuity, which let them to enforce a connectivity of contours in a quantitatively way. When tracking a rigid object, they use the geometry of the object to find the optimal pose, while in the case of the deformable models, they use a Hidden Markov Model to calculate the joint law of the conditional probabilities of contour points. They train weak classifiers in order to obtain a final weight to determine if there is a cut in the middle of a image band or not. Depending on the complexity of the video sequence, they report to have used between 4 and 10 classifiers.

3.2.4 Interest points.

Interest points are often employed for 3D reconstruction and absolute localization 3D environment, as well as initial pose estimation for tracking purposes. The principal characteristics of these points are that they must be invariant to illumination, to 3D projective transforms, and common object variations. They should be also sufficiently distinctive to identify objects among multiple alternatives. This is why they allow an object to be tracked independently of background clutter and partial occlusions. To handle partial occlusions, we can consider that having a subset of the interest points is enough to estimate its location. This can be done with the aid of a 3D model of the object, or by geometric constrains.

To do that, we should match interest points extracted from gray level or color images. A work very used to detect interest points is [40], where corners are estimated in gray level images. In order to do that, it uses a moving window to estimate the gradient of each pixel in 8 directions and then, by means of a threshold, it determines if it corresponds to a corner or not. This work has been widely used in the computer vision community, because the features obtained with this algorithm were not only corners, but points with large gradients in all directions at a predetermined scale. The principal disadvantage of this approach

is that it is sensitive to scale variations.

In [83] the authors extend the work of [40] to color images, and improve the algorithm to make it more robust. They use a bilateral filter in a smoothing step in order to reduce noise present in the image taking into account not only the distance between pixels, but also the chromatic information between them. They also proposed an iterative version of their filter. The advantage of this kind of filter is that it reduces the localization error generally presented when a smoothing filter is applied to an image.

Other work that uses interest points and that is also based on [40] is [54], where the authors use a classification approach, based on K-means and nearest neighbor, to determine if a point detected is member of a class learned during a training. They use a geometric model to perform the training, and a RANSAC-based method to estimate the 3D pose of the objects. They also propose in [51] an ameliorated version of their algorithm but using classification trees to perform more efficiently the classification of the features obtained with their own algorithm to obtain interest points, based in pixels inside a circle, and comparing the characteristics of all the neighbor pixels. Finally [67] proposes a method similar to the last one, but oriented to deformable objects. They use not only the Harris-corner estimator, but also the one proposed in [58], where the author proposes a method for generating image features invariant to translation, rotation, scaling, and partially invariant to illumination changes and affine or 3D projections. With these features, he makes an index of objects that allows different objects in images to be identified. The principal advantage of this kind of method is that it is more robust to scale changes, because the corners detected at different scales with the Harris detector, as we said before, are not necessarily the same.

3.2.5 Other features.

Some preprocessing of the image can give interesting information about the object that we want to track. Some of these methods include the use of many examples of the object in order to obtain good descriptors of it, like the Principal component Analysis. Another example can be found in the Adaboost, where a preprocessing step creates features that are then used to determine whether or not a face is present in an image window.

As we have seen, there is a panoply of features that can be extracted from an image. All of them imply a preprocessing of the image in order to highlight the desired characteristic. However, for most of the cases, a simple feature can not guarantee the robustness of a tracker for all the possible cases. This is why hybrid trackers are used to fuse information of multiple features.

3.3 Tracking methods.

Once we have defined the principal representations of an object, as well as the most common features used to characterize it, we will present the methods used

to perform the tracking of the object. There exist several methods for tracking, but we are going to present the most popular for face tracking. We will start presenting the gradient descent method, which is widely used because it is simple and easy to implement. Then we will talk about the supervised learning methods, which need a training step to learn the desired object characteristics. For this supervised learning part we will talk about the classification and the regression methods. For the regression methods we will talk about the CCA, the Active Appearance Models, and the morphable models. Finally we will present the particle filter, which is used to model non linear and non Gaussian processes.

3.3.1 Gradient based approaches.

Using the gradient is a very common and easy technique to be implemented. In the case of a known model of the problem treated, we can obtain a closed form of the gradient, indicating us the direction where the maximum variation is located. Gradient descent is an optimization algorithm. To find a local minimum (or maximum the algorithm is then called gradient ascent) of a function using gradient descent, one takes steps proportional to the negative (when looking for the minimum) of the gradient, or the approximate gradient, of the function at the current point. Gradient descent works in spaces of any number of dimensions, even in infinite-dimensional ones.

In [25; 31] the authors propose a method based on the gradient descent algorithm to track the motion and facial gesture of faces. To accomplish this, the authors use a geometric face model coupled with a gradient approach to handle the transition model of the face tracker. They compare the fact of using a fixed gradient matrix with a gradient matrix that evolves over time. They use the geometric model to estimate the gradient matrix, by adding some perturbations to the pose and animation parameters of the geometric model. They showed that their approach when the gradient was estimated only at the first time is not robust to out of plane rotations nor to facial animation, while their approach that estimates the gradient matrix every n frames is more robust to these variations. This method can be compared with ours, but in our case we use the geometric model and the perturbations in a supervised learning context, that will be explained further.

Another example that uses a gradient descent technique is the active blob algorithm proposed in [74]. In this example the object to be tracked is modeled by a triangular mesh and a color texture, and it can handle non-rigid objects. The optimization criterion uses a gradient approach to estimate some partial derivatives with respect to the model parameters, and the result of this gradient is then used in addition to a Hessian matrix to converge to the desired solution.

However there are some known drawbacks to the gradient descent approach. Some of them are that it can need many iterations to converge toward a local minimum, if the curvature in different directions is very different, and finding the optimal step at each iteration can be time-consuming. Conversely, using a fixed step can yield poor result if it is too big, or to a very slow convergence if the step is too small.

3.3.2 Supervised learning.

Supervised learning is a method that has three components. A set of learning images containing different views of the selected object features to be learned, a set of labels that are manually assigned to every image in the set of images, and a mechanism to learn the relation between these object features and the labels assigned. Basically it consists in creating a function that maps inputs to desired outputs [100]. This mapping can be of two forms. If we have an output in the form of a class label, then we will talk about a classification. In the other hand, if the output parameters can take continuous values we will talk about a regression. These two approaches are next explained.

3.3.2.1 Classification.

Classification consists in determining whether or not an object is an element of a class. This object can be a point, a patch, or a whole image. This kind of approaches presents the advantage of being very fast to determine the belonging of an object to a class, but to have this classifier, it is required to perform a training. During the training we can use two sets, one containing examples of the target object or objects, and one with other kind of objects, such that we can have at least two classes. Selection of these classes represents a hard task, and the training of the classifier also, because we need to have a lot of information, images in our case, that contain the object or feature class, and also lot of information of non-objects. Additionally, this input information must be aligned, which is often done manually.

A widely used classification method is the Adaboost algorithm [100]. Adaptive Boosting is an iterative method of finding a very accurate classifier by combining many weak classifiers. For each weak classifier, a weight is associated and it depends on the misclassified data. The classifiers are put together in a cascade, so that the classifiers that are highly discriminant are used at the first stages. The following stages are chosen to be more discriminant for the misclassified data that can be obtained from previous stages. In this way an image that does not correspond to a given object has a very high probability of being rejected in the first stages of the test and thus, further computation is not performed. Viola and Jones proposed in [86] the most known version of the Adaboost algorithm. It uses Haar-like features as weak classifiers. This classifier has been used to detect frontal view faces, profile faces and some other objects and features. The resulting detection is commonly used for tracking purposes, as an initialization step or for recovery, as we will present in chapter 6.

Another widely used classifier approach is the support vector machine (SVM). It consists in finding the maximum marginal hyperplane that separates one class from the other. A certain measure is obtained from the distance between the hyperplane and the data points. The data points that are closer to the hyper plane are called the support vectors. This method can be extended to a non linear by means of the kernel trick and a kernel function, that will project the data to a

high dimensional space where an hyperplane could be estimated when the linear version can not estimate this hyperplane. In [5], the author proposes the support vector tracking (SVT) that uses the classification scores of the support vector machine (SVM) in order to track optical flow. To accomplish this, the author fuses the tracking and the classification in such a way that the cost function optimizes the score of the classification rather than the brightness difference between two consecutive images. The SVT algorithm uses a kind of iterative gradient descent optimized over the SVM's scores. This algorithm is used to track vehicles from a camera mounted inside a car and it estimates the horizontal and vertical displacement of the target. The principal advantage that is reported of doing this, is that fusing the classifier and the tracker information gives better results than doing it sequentially, tracking first and then obtaining classification scores, because the tracker can lead to regions where the classification scores are trapped in a local minimum.

However, the task of tracking can also be done using a regression scheme instead of a classification and in [88], the authors improve the work of [5]. First, they perform a training of the SVM by moving a known object of some pixels in a four dimensional space, corresponding to 2D vertical and horizontal translation, scale and rotation. Then, with the SVM scores obtained, the authors perform a linear regression in order to link the SVM scores to location's variation. They improve also this algorithm and propose to use the relevance vector machine (RVM) in [89]. The principal advantage of RVM is that it is well fitted for regression task and introduces automatically a probabilistic measure of the results. They complement this tracker with a SVM for initialization and recovery. This approach is used to track faces and car plates.

In [51] the authors use a feature point detector based on [40], and use a classification to determine whether or not each feature point detected corresponds to one of the class selected during training. The selection of the classes during training is done by choosing those feature points more representative and present over the whole set of training images. Then, only those that do not lead to a misclassification over a certain limit are kept. In order to reduce the dimension of the patches, a principal component analysis is performed and then a k-mean estimation is used to compact the representation of the view-set. This algorithm was reported to be very fast and accurate. In order to ameliorate their approach, a classification tree method is proposed in [52; 53]. This kind of approach is more complicated to implement, but it is more robust and adapted to classification problems. They also proposed in this approach their own feature detector, that quickly determines if a pixel is a feature point or not. This point detector is reported to be fast and stable, with a reduced complexity with respect to the traditional methods. To do this, the authors scan a circle around a point, and if they find that the neighbors have a similar intensity, then it is not an interest point. This method attributes also an orientation to each point, in order to introduce robustness to 2D rotations. The results reported show the robustness of the method for 2D affine movements in real-time applications.

As we have shown, classification is used to locate objects in video sequences,

and as previously explained, a filtering step is needed to find the correspondences between the detected objects across the frames. However, in the approaches proposed by [52; 53; 88], they do not use this component to find the correspondences, but assume that only one instance of the object is present in the image, so they approach the tracking process by the detection of the object in each image independently of the previous results.

3.3.2.2 Regression.

The regression is a statistical tool widely used to describe the relation existing between two variables. The mathematical model of their relationship is the regression equation. This equation contains estimates of one or more hypothesized regression parameters. To obtain these parameters, we have to use samples of both variables.

Uses of regression include curve fitting, prediction (including forecasting of time-series data), modeling of causal relationships, etc.

CCA. Due to the high dimensionality that arises when working with images, the use of a linear mapping to extract some linear features is common in the computer vision domain. One of the most prominent methods for dimensionality reduction is *Principal Component Analysis* (PCA) which deals with one data space and identifies directions of high variance. However, faces represented by principal components are sensitive to illumination, scale, translation and rotation. Moreover, using first a PCA for dimensionality reduction and then trying to find the linear relation between the reduced data set and the corresponding parametric data set can lead to a loss of information, from a regression point of view, as PCA-features might not be well suited for regression tasks, as is demonstrated in [60]. In our case we propose to use a *Canonical Correlation Analysis* (CCA) to find linear relations between two sets of random variables, because the dimensionality reduction is performed jointly with the two data sets. [6; 9; 87].

Canonical correlation analysis (CCA) is a very powerful and statistical tool that is especially well suited for relating two sets of measurements, that has been little known in the field of signal processing and pattern recognition until recently. CCA can be seen as the problem of finding the direction vectors for two sets of variables such that the correlations between the projections of the variables onto these direction vectors are mutually maximized. The number of direction vectors is equal to or less than the smallest dimensionality of the two variables. These direction vectors are calculated by using jointly the two data sets in order to obtain the relationship that exists between them.

CCA has recently been used for appearance based 3D pose estimation [60], appearance-based localization [76] and to improve the AAM search [29]. These works highlight the advantages of the CCA to obtain regression parameters that outperform standard methods in speed, memory requirements and accuracy (when the parameter space is not too small).

An important property of canonical correlations is that they are invariant with respect to affine transformations of the variables. This is the most important difference between CCA and ordinary correlation analysis which highly depends on the basis in which the variables are described [8]. We will present more in detail the definition of the CCA, and then we will present the formulation of the problem and the procedure to obtain the solution, in the chapter 4, where we will present our approach. A detailed optimization formulated by [87], is presented in appendix A which due to the nature of our data, was the formulation used for our implementation.

Kernel Canonical Correlation Analysis (KCCA). The principal idea behind kernel methods is better understood if we consider a simple binary classification problem in $\mathbb{R}^p$ with non-overlapping classes, i.e. we have two classes that can be perfectly delimited by a hyper plane of dimension $p-1$. In this case, a linear method will perform correctly the classification task. However, if the training data $\mathbf{x}_i \in \mathbb{R}^p$ is not linearly separable, we can still perform a linear classification but first we need to map this training data into a higher dimensional feature space [37; 38; 39; 43; 60]:

$$\phi : \mathbb{R}^p \mapsto \mathbb{R}^s, s > p. \tag{3.1}$$

Thus, using kernel-functions we can formulate our problem as a non-linear version of the original one with the advantage that the complexity of the transformed problem is not linked to the feature space dimension, but to the training data set dimension, which means that we can use kernel transformations to feature spaces of high dimensionality.

We can apply the *kernel-trick* if we can express the classifier in such a way that it only uses dot products of the transformed input data. This is useful because we can express the dot product in the feature space in terms of kernel functions in input space, i.e., $\phi(\mathbf{x})^T\phi(\mathbf{y}) = k(\mathbf{x}, \mathbf{y})$. A sufficient condition for a kernel-function to correspond to a dot product in a feature space is given by *Mercers Theorem*. In [60], it is shown that CCA can be completely expressed in terms of dot products.

KCCA is used in[43] to measure the independence between two data sets, in [37] to learn a semantic representation of web images and their associated text, in [38] to infer brain activity by learning a semantic representation of functional magnetic resonance imaging of brain scans and their associated activity signal, in [60] it is used to create appearance models in order to estimate the pose of different objects and in [103] it is used to estimate the facial expression.

Like for the CCA, we will describe more explicitly the KCCA in chapter 4.

3.3.3 Particle Filter.

Particle Filter is a Monte Carlo methodology for sequential signal processing that has become of great interest in the last decades with the fast advances in computers. The principal advantage of this kind of approach is that it can cope with

difficult nonlinear and/or non-Gaussian models, or when an analytical solution can not be easily obtained, or can not be obtained at all. The underlying principle of this methodology consists in approximating relevant distributions with random measures composed of particles (samples from the space of the unknowns) and their associated weights. A tracking example is depicted in figure 3.3. A very complete tutorial explaining particle filters can be found in [28].

In [82], the authors propose an algorithm that can cope with multiple targets, by using the number of targets as a component of the state vector that is estimated at each iteration as well as an occlusion variable. To do this, the algorithm decouples the sampling process in two parts, one local used to track the motion of individual objects, and one global, used to track addition or deletion of objects. This algorithm uses 300 particles and is capable of processing one frame per second on video frames of size 320x240.

In [46], the authors propose also an algorithm capable of tracking multiple people, and in this case the number of moving objects is also unknown. The difference is that in this algorithm there is an observation model which accurately reflects the likelihood of differing numbers of objects being present. Once this model is present, the particle filter obtains the posterior distribution over the number and configuration of the objects.

In [31], the authors use a geometric model and a particle filter to track the 3D pose of human faces in a 3D environment. They obtain good results for the six parameters of the 3D pose tracking, but to extend the work to track facial gesture they were constrained to use another approach. The principal inconvenient of this kind of approaches is that they are not well adapted to conditions where the state vector to be tracked is of high dimensionality, because the number of particles required to do that increases exponentially.

Figure 3.3: Results from a particle filter that uses color to track a person. From left to right, reference image and two tracking frames where the blue rectangle represent the particles and the red rectangle represents the tracking estimation.

3.3.4 Some examples.

In recent years there have been multiple propositions used for tracking faces. Two of the most popular are the Active Appearance Models (AAM) and the 3D Morphable Model. These approaches model the shape and the appearance of faces.

The principal difference between these two approaches is that for the AAM, 2D images are used to create the shape model, while in the case of the 3D morphable model, 3D laser scans are used [91].

3.3.4.1 Active Shape Models and Active Appearance Models.

The description of Active Shape Models (ASM) can be found in [20], where the characteristics of objects are learned during a training phase by building a compact statistical model representing shape variation of the object. During the training phase, local features are annotated by hand in several images, intended to contain all the possible variations that can perturb the shape of the object. The chosen features are in this case corners and some of the boundary points lying between these corners, which can be found in all the image samples. These points are put together in a vector, for each image, and all the vectors are used to create a training dataset. Then, the principal component analysis (PCA) is performed to reduce the dimensionality and to obtain a reduced set of parameters that describes the perturbations of the model from a reduced number of eigenvectors. The goal of ASM is to find the model parameters that best match the shape of the object in a new image. To do this, the first step is to use a feature detector, and then try to find the model configuration that best fits the feature points obtained. This is done iteratively until the fit measure is not improved.

Active Appearance Models (AAM) can be seen as an extension of ASM [21]. This method obtains not only a statistical model of the shape, but also one of the appearance (grayscale texture), and these models are fused to create one that describes the variation in shape and texture that an object in an image can be subject to. In this case the training step consists in using the notated face images as explained before, to obtain a statistical shape model, and also the texture that lies under this model. From this texture another model is obtained. As these two models are highly correlated, they are put together and by means of a PCA a new set of parameters is obtained. With this model the relation between displacement of a synthesized face and the original image is learned by means of a linear regression. In this way the algorithm performs an analysis by synthesis, creating a frame that tries to be as similar as the original frame by means of the learned AAM, estimating the perturbation of the parameters that corresponds to the difference between the frame and the synthesized image. In [29], they use the CCA to obtain the linear model between the perturbations parameters and the error between the synthesized image and the current frame.

Constrained Local Model described in [24] is a very similar method, but the principal difference is that only particular features of the face are used, and the model is learned only for patches around these local features.

There are some similarities between AAM and our approach, especially because both methods find the regression between a residual and pose parameters, and in our case this regression is performed by means of the CCA. However, our approach uses a geometric model to model the shape, and that extends the tracking capabilities to 3D pose parameters. Another key difference is that we do not

perform an analysis by synthesis, but we compare the face in the current image with a reference face to correct the pose parameters.

3.3.4.2 3D Morphable Model.

The idea behind the 3D Morphable Model (3DMM) is to synthesize an image of a face that resembles as much as possible as the face of the input image [68], by means of a parameterized model. It consists then of two parts, a model and a fitting algorithm. The model should handle the conditions that modify the appearance face image.

In order to construct the 3D morphable model, it is required to have a set of 3D faces obtained from several persons. In his work, Romdhani [68] uses 3D laser scans. Then, he finds correspondences between the scans of one chosen as the reference model and the other, and then the principal component analysis is performed to estimate the statistics of the 3D shape and the texture of the faces. This model intends to describe the parameters in such a way that they are independent. These parameters should describe the principal sources of these variations, which are, as described in [69]:

- ⋆ Pose changes produce changes of faces images. They produce also occlusions of inner features.
- ⋆ Illumination changes alter also the face image. Even with a fixed pose, changes of illumination produce important changes in a face image.
- ⋆ Facial expression can also produce significant changes on the face appearance.
- ⋆ Aging in the longterm is also a source of image changes.

To take into account model changes due to illumination and pose, morphable models impose the physical laws of nature based on the 3D geometry of faces and the interaction of their surfaces with light. The other parameters are obtained by exploiting the statistics of faces.

Once we had this model, we use it to synthesize faces. This is a method of analysis by synthesis. This means, given an image containing a face, we synthesize an image until we obtain an image that is close to the original one. This will give us a set of parameters of the morphable model that could be used for multiple purposes next. This approach is used in [70], to estimate the pose, the texture and the variations in illumination.

In [91] we can find a method that compares the AAM and the 3DMM, and that creates a method that is a mix of both approaches, extending the AAM approach for 3D models.

The principal differences between the 3DMM and our approach are that we do not use 3D scans, but 2D images to construct a representation of the face by means of a 3D geometric model and we do not use an analysis by synthesis approach.

3.4 Conclusions.

We have presented the principal criteria that need to be considered before building a tracker, which are the choice of the object representation, the choice of the features to be used for tracking purposes, and some of the most used tracking methods. The choice between these criteria choice will depend on the kind of analysis that we want to perform from the obtained data. The selection of the method will then depend on the characteristics and the possibility to model the tracked object and the possible movements that it is expected to do. Another characteristic that should be considered for the implementation of the tracker is to decide if the tracker will be specific, or general, i.e., if we want to track a particular kind of objects, as persons for instance, or a more general tracker able to track every moving object in the visual area of the camera. Finally, but not less important, we have to know the external factors that will face our algorithm. We need to know if the tracker will be used in outdoor or indoor environments, or if it will be applied to the video obtained from a fixed or a moving camera, because these factors will also determine the choice of the approach.

In our case we have selected an appearance representation for the face that is obtained by means of a 3D geometric model. With this representation and the parametric model we create synthesized face images by adding some perturbations to the model parameters. The use of a model makes the tracking of a specific object more robust and discriminant. In our case, we also use the model to create synthetic views of the face using a single face image Then, we use a supervised learning approach that uses the CCA to find the linear relationship that exists between the constructed face images and the model parameters perturbations. The CCA is a statistical tool with very interesting characteristics that has not been used in signal processing until last years, that finds the linear relations between two data sets. We will present our approach in more detail in the following chapters as well as the principal advantages of it.

It is important to say that although there are algorithms with very good performances, there is a lot of work to do, and the fact that the technology evolves every day, will led us to develop algorithms more and more complex that will be applied in real time applications. These applications are also growing as the algorithm become more and more complex, letting us not only to estimate simple 3D parameters, but also detect emotions or strange behaviors, and they will change the way human interact with computers.

Chapter 4

3D Pose Tracking of Rigid Faces.

Nowadays video face processing is a popular component in video applications, especially because a facial image conveys many pieces of information such as identity of a person, the gender, his age, the relative position with respect to the camera, the deformation of an individual face due to changes in expression and speaking, variations in the lighting, etc. This is why faces has aroused the interest of computer vision community, besides they are very familiar to us, i.e., we have a well developed sense of what expressions and motions are natural for a face, and also because human faces are very complex three dimensional surfaces, that are flexible, usually contain creases, and they have color variations, and thus, they represent a real challenge. So, human face perception is currently an active research area in the face image processing community.

There are several related topics of analysis, synthesis and processing of images containing human faces [3], but in our study we are more interested in locating and tracking faces in video sequences. The interest of locating and tracking human faces is that it is a prerequisite for face recognition and or facial expression analysis, even if it is assumed in most of the cases that a normalized face image is available. There is a panoply of applications behind face tracking, like face-based biometric person authentication, human-computer interactions, behavior analysis, driver monitoring systems, marketing and advertising, interactive information system, computer games, teleconferencing, surveillance or behavior understanding, etc.

So, in order to deal with face tracking, vision research groups have proposed in the last years several approaches that can be classified in two main groups: model-based and learning-based approaches. In the first category, tracking algorithms rely on a parametric model of the object to be tracked. In the second category, algorithms presuppose the availability of a training set of object examples, and use pattern recognition/classification techniques. We can also make a classification by the data derived from the video frame, as edges, interest points, gray level intensities or color histograms, etc, that were briefly presented in chapter 3.

Nearly all parametric facial feature tracking techniques (ASMs, AAMs, etc.), also introduced in chapter 3, operate the central idea of finding a relationship

between shape and appearance. Some people do this explicitly through gradient techniques or others have done this through learning techniques like regression (linear or kernel based).

Due to the high dimensionality that arises when working with images, the use of a linear mapping to extract some linear features is common in the computer vision domain. One of the most prominent methods for dimensionality reduction is *Principal Component Analysis* (PCA) which deals with *one data set* and identifies directions of high variance. In our case, we are interested in identifying and quantifying the linear relationship between *two data set*: the change of the pose parameters of the *Candide* model and the residuals of the face appearance normalized to a neutral pose. If we use first a PCA to reduce the dimensionality of the data sets and then we try to find the linear relation between them, a loss of information containing the relationship between these two data sets may be produced, as PCA-features might not be well suited for regression tasks. In our case we propose to use a *Canonical Correlation Analysis* (CCA) to find linear relations between two sets of random variables and make the dimensionality reduction of both data sets jointly [9; 87]. CCA finds pairs of directions for two sets of vectors, such that the projections of the variables onto these directions are maximally correlated. CCA is a statistical method which relates two sets of observations, and that is well suited for regression tasks. CCA has recently been used for appearance based 3D pose estimation [60], appearance-based localization [76] and to improve the AAM search [29]. These works highlight the advantages of the CCA to obtain regression parameters that outperform standard methods in speed, memory requirements and accuracy (when the parameter space is not too small).

When tracking a rigid object in a 3D space, the pose of an object may be explained by a state vector $\hat{\mathbf{b}}_t$ at frame t of dimension six, three translation parameters and three rotation parameters. This chapter addresses the problem of tracking in a video the global motion of a face as a rigid object. In our case, we propose a deterministic approach based on Canonical Correlation Analysis (CCA), on raw brightness images, and in a similar way as in [25], we use a 3D face model to track people's 3D head pose.

Although model-based methods and CCA are traditionally used in the computer vision domain, these two methods together were not already used in the tracking context until very recently [29; 60]. We will show experimentally on different public and own video sequences that, indeed, our CCA approach is well suited to obtain a simple and effective facial pose estimation.

A tracker, as explained in chapter 3, usually consists of two components: a detecting component, and a filtering component to add temporal continuity constraints across frames and to deal with dynamics of the tracked object. As we said before, these components can be done jointly or separately [100]. In our case we do it jointly by taking into account the last known position in order to estimate the variation of this position with respect to a reference vector, which is updated from frame to frame.

This chapter is structured as follows. In the first section we will talk about

the geometric models, and particularly about the *Candide* model that is a crucial piece of our work, then we will introduce the Canonical Correlation Analysis and Kernel CCA used in our work. After that, we will present how we link these concepts to track a face in a video sequence to finally present experiments on long video sequences.

4.1 Parameterized geometric models.

Geometric facial models for computerized facial animation were proposed by Parke [63; 64] for computer graphics purposes in the 70's. To obtain these kinds of models, the author drew on a person polygons and took frontal and profile photos of the drawn face to obtain a 3D model. In this early work the author already suggested that facial models might be used for data compression, previews of the effects of corrective surgical or dental procedures, interactive applications, tracking, etc. This makes people to get interested in representing facial movements in a parametric way, in order to code all the possible deformations of a face with a very reduce number of parameters. This interest gave origin to parameterized models which became powerful tools for facial image synthesis and analysis. Geometric face models have been used in the computer vision community for several purposes, from 3D synthesis, compression of video sequences, to human behavior analysis.

As the use of these geometric models grew, also did the necessity of a standard for facial movements, and this gave origin to the Facial Action Coding System (FACS) [33] as a method for measuring and describing facial behaviors. Paul Ekman and W.V. Friesen developed the original FACS in the 1970s by determining how the contraction of each facial muscle (singly and in combination with other muscles) changes the appearance of the face. They examined videotapes of facial behavior to identify the specific changes that occurred with muscular contractions and how best to differentiate one from another. They associated the appearance changes with the action of muscles that produced them by studying anatomy, reproducing the appearances, and palpating their faces. Their goal was to create a reliable mean to determine the category or categories in which to fit each facial behavior. It is the most popular standard currently used to systematically categorize the physical expression of emotions, and it has proved useful both to psychologists and to animators.

FACS measurement units are Action Units (AUs), not muscles, for two reasons. First, for a few appearances, more than one muscle was combined into a single AU because the changes in appearance they produced could not be distinguished. Second, the appearance changes produced by one muscle were sometimes separated into two or more AUs to represent relatively independent actions of different parts of the muscle.

FACS defines 32 AUs, which are a contraction or relaxation of one or more muscles. It also defines a number of Action Descriptors, which differ from AUs in that the authors of FACS have not specified the muscular basis for the action

and have not distinguished specific behaviors as precisely as they have for the AUs.

Using FACS, human coders can manually code nearly any anatomically possible facial expression, decomposing it into the specific AUs and their temporal segments that produced the expression. As AUs are independent of any interpretation, they can be used for any higher order decision making process including recognition of basic emotions, or pre-programmed commands for an ambient intelligent environment.

With the success of the MPEG-4 standard for face animation which provides a standardized way for storing and transmitting animation parameters, model-based coding and face animation have been increasingly popular research topics. The basic idea of model-based coding is that the appearance and motion of a human face are analyzed, in order to extract compact parameters. These parameters can be transmitted at very low bit rates, potentially allowing video face-to-face communication over narrow channels.

Thus, at the beginning, the purpose of the face models was to represent the face with the minimum amount of complexity, using the least quantity of points and polygons, to reduce the computing time of deforming it. One example of these models can be seen in the model developed by Rydfalk [73], known as the original *Candide* model. *Candide*is a parameterized model consisting of some vertices, that are connected by means of polygons. Its low number of polygons, approximately 100, allows fast reconstruction with small computing power. Based on this version, Ströemberg [81] created the standard *Candide* model, which is a slightly modified model with 79 vertices, 108 surfaces and 11 Action Units AUs. In our work, we used the wireframe model *Candide 3*, proposed by Jörgen Ahlberg[2]. This model consists of a small number of vertices, 114, and a small number of triangles, 178 after some personal modifications, and it represents a face statically and dynamically by means of shape animation units with an acceptable realism. *Candide* model is still widely used, since its simplicity makes it a good tool for image analysis tasks and low complexity animation. It was used in the head and facial tracking context in [31], in [36] to restore human faces in noisy video sequences, in [4] it was modified automatically from a profile and a frontal view in order to capture the physical characteristics of a person, in [15] to estimate the 3D pose from planar images. In the following section we will describe in detail the Modified *Candide* model used in our work and how we used it to obtain a facial texture.

4.1.1 *Candide* model.

In figure 4.1 we can see the extended *Candide* model. The principal difference between our version and that proposed by Jörgen Ahlberg [2] is that we added some points to better describe the eyebrows, and the nose, as well as some shape units, to better adapt it to face characteristics. The resulting model consists of 120 3D vertices, that are used to form 198 triangles. They can be modified by 69 animation units and 16 shape units.

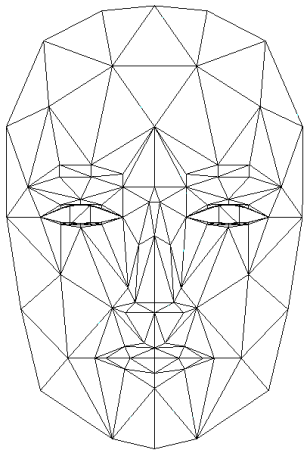


Figure 4.1: Extended *Candide* model.

The *Candide* model consists of a group of 3D vertices interconnected to describe a face by means of the triangles formed between these vertices. We can write these 3D vertices as $P_i, i = 1, ..., n$ being n the number of vertices. Thus, we have a $3n$-vector $\mathbf{g}$ which is the concatenation of all the vertices $\mathbf{P_i}$. This vector can be written in terms of the modifications that can be applied as:

$$\mathbf{g}(\tau_\mathbf{s}, \tau_\mathbf{a}) = \overline{\mathbf{g}} + \mathbf{S} \cdot \tau_\mathbf{s} + \mathbf{A} \cdot \tau_\mathbf{a} \tag{4.1}$$

where $\overline{\mathbf{g}}$ is the standard shape of the *Candide* model, $\tau_\mathbf{s}$ and $\tau_\mathbf{a}$ are shape and animation control vectors, and the columns of $\mathbf{S}$ and $\mathbf{A}$ are the Shape and Animation Units respectively [73], [2]. Thus $\mathbf{S} \cdot \tau_\mathbf{s}$ accounts for the shape variability and $\mathbf{A} \cdot \tau_\mathbf{a}$ accounts for the facial animation, both of them assumed to be independent. The model (4.1) can be seen as one static part and one dynamic part acting independently. The model's static part corresponds to characteristics such as nose size, eye separation distance, eyebrows vertical position, etc, and can be written as:

$$\mathbf{g_s}(\tau_\mathbf{s}) = \overline{\mathbf{g}} + \mathbf{S} \cdot \tau_\mathbf{s} \tag{4.2}$$

and it is inherent to a given person. An example of these parameters is shown in figure 4.2, where some shape units corresponding to eyebrows and eyes vertical position are illustrated.

To acquire the static part, it is necessary to obtain $\tau_\mathbf{s}$. In this part of our study $\boldsymbol{\tau_s}$ is obtained manually. This can also be done automatically as proposed in [4], and as we will present in chapter 6.

To perform the manual initialization, we place the *Candide* model over the first video frame $\mathbf{y}_0$. Then we modify the control vectors $\boldsymbol{\tau_s}$ and $\boldsymbol{\tau_a}$, by fitting the *Candide* shape over the face in the first video frame.

Once the mask is modified to the person's characteristics, we can simplify our model equation (4.1) as:

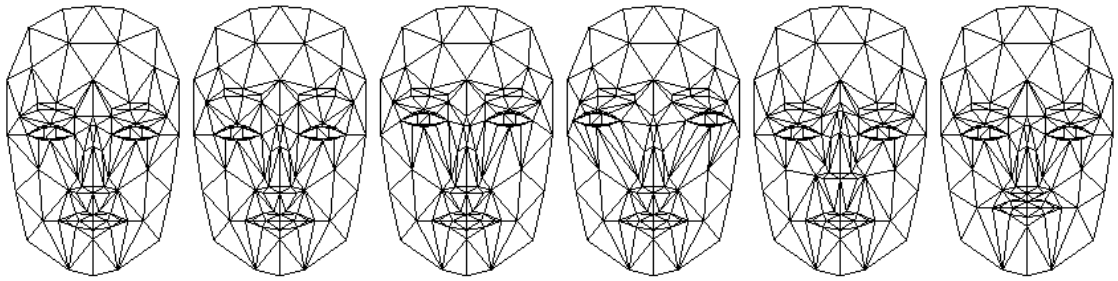


Figure 4.2: Example of shape units. From left to right, standard shape of the *Candide* model, eyebrows' vertical position set to maximum, eyes' and eyebrows' vertical position set to maximum.

$$\mathbf{g}(\tau_\mathbf{s}, \tau_\mathbf{a}) = \mathbf{g_s} + \mathbf{A} \cdot \tau_\mathbf{a} \tag{4.3}$$

It is important to say that there is also a difference between the proportions of the face for every people, so it is necessary to adapt also the vertical, horizontal and deep proportions by a scalar factor in each direction. Although this is adapted at the initialization step with the shape parameters, this scaling also affects the animation parameters. We can then write equation (4.3) as:

$$\tilde{\mathbf{g}}(\tau_\mathbf{s}, \tau_\mathbf{a}) = \mathbf{P}(\mathbf{g_s} + \mathbf{A} \cdot \tau_\mathbf{a}) \tag{4.4}$$

being $\mathbf{P} = P(p_x, p_y, p_z)$ a scale matrix that modifies each 3D point of the *Candide* model.

Since we also want to perform global motion, we need a few more parameters for rotation and translation. Thus we replace (4.4) by:

$$\check{\mathbf{g}}(t_x, t_y, t_z, \theta_x, \theta_y, \theta_z, \tau_s, \tau_a) = \mathbf{RP}(\mathbf{g_s} + \mathbf{A} \cdot \tau_\mathbf{a}) + \mathbf{t} \tag{4.5}$$

where $\mathbf{R} = R(\theta_x, \theta_y, \theta_z)$ is obtained from a rotation matrix and $\mathbf{t} = t(t_x, t_y, t_z)$ is obtained from a translation vector.

As $\mathbf{g_s}$, $\mathbf{S}$ and $\mathbf{P}$ remain constant after the initialization step, we can state that given the *Candide* model, our 3D tracking problem consists in estimating the 3D head pose and the control vector $\tau_\mathbf{a}$, what makes the state vector to be written as:

$$\mathbf{b} = \left[\theta_x, \theta_y, \theta_z, t_x, t_y, t_z, {\tau_\mathbf{a}}^T\right] \tag{4.6}$$

For simplicity when referring to the *Candide* model we will use the notation $\mathbf{g}(\mathbf{b}_t)$, where the t stands for the time index.

When positioning the *Candide* model over the current frame $\mathbf{y}_t$, we can obtain the texture that lies under it, in order to warp it to the 3D *Candide* model. To do this, we used a well known technique called texture mapping. It consists in mapping a source 2D image onto a 3D surface, which is then mapped to the destination 2D image by means of a perspective projection. This means that we

have three coordinate spaces. The 2D texture space labeled as (u, v), the 3D object space labeled as (x_o, y_o, z_o) and the 2D screen space that is labeled (x, y) [41]. In other words, taking the *Candide*'s 3D vertices we estimate for each vertex the corresponding screen coordinates (x, y). Then from these screen coordinates (x, y) we estimate also for each vertex the texture coordinates of the current frame (u, v). Finally, we warp the texture contained in each triangle of the texture image to each triangle of the 3D *Candide* model.

To verify that the texture obtained is correct, we draw the obtained 3D model with the texture warped on it in a frontal view, as can be seen in top of figure 4.3(b) and in a three rotated views, as can be seen in the bottom of figure 4.3 (b). The three synthesized rotated views are useful to adjust the z-component and the parameters of $\tau_{\mathbf{s}}$ and $\tau_{\mathbf{a}}$. The vector $\boldsymbol{\tau_a}$ is supposed to be constant for the pose estimation: the face is seen as a rigid object during the pose estimation process, with a fixed expression.

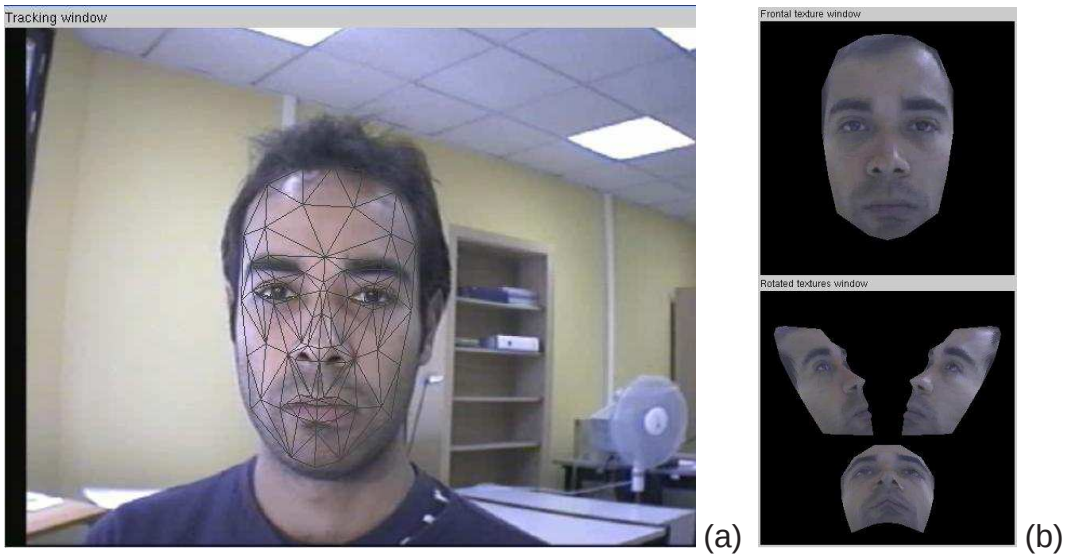


Figure 4.3: (a): *Candide* model placed over the target face in the first video frame. (b): frontal texture views of the target face in the top and profile synthesized texture views of the target face in the bottom, used to help the initialization step.

4.2 Normalized face's texture.

Although normalization process is often treated as a separate preprocessing step, it is an inherent part of face recognition. The ability of a system to produce normalized face sequences implies that it recognizes faces as a unique class of objects in a manner which exhibits invariance under many possible factors, such as changes in illumination, orientation, position in a 3D space, etc.

In our case we performed a geometrical normalization as described in [3] and later in [31]. In our case, we use the geometric model to produce a normalized

face. To proceed like this we use a transformation of the texture obtained at each frame. It consists in several steps:

- ⋆ First we take the 2D texture lying underneath the 3D *Candide* model, and warp it to the 3D model.
- ⋆ Then, we draw the 3D *Candide* model with the texture warped, with all the rotations and translations fixed to a predefined value, in this case, zero, except for the scale parameter, for the frontal view. Then, we add two synthesized profile views at each side with a rotation of ± 60 with respect to the vertical axis. All the expression parameters $\tau_{\mathbf{a}}$ are set to zero.
- ⋆ Next, we load a fixed size window containing this draw and transform it to a greyscale image, obtaining finally an expression-free patch of size 58×72 pixels as the one depicted in figure 4.4.
- ⋆ This patch is then normalized by the norm of all the pixel values and reshaped, giving the vector $\mathbf{x}_t = \mathcal{W}(\mathbf{g}(\mathbf{b}_t), \mathbf{Y}_t)$ of size $d = 4176$ by taking each column and putting it under the previous one ($\mathcal{W}$ can be seen as the texture mapping operator).

The reason of doing this geometrical normalization is to obtain the face features, for instance the eyebrows and the lips, always at the same place.

Figure 4.4: Expression-free patch.

In our case we have used the OpenGL library to implement the texture mapping between the frame image and the *Candide* model to create the expression-free patch.

This vector $\mathbf{x}_t = \mathcal{W}(\mathbf{g}(\mathbf{b}_t), \mathbf{Y}_t)$ is called stabilized face images (SFI), and it will be considered as our observation. The vector obtained from the initialization step will be called reference vector and it is denoted as $\mathbf{x}_0^{(ref)} = \mathcal{W}(\mathbf{g}(\mathbf{b}_0), \mathbf{Y}_0)$, where $\mathbf{b}_0$ corresponds to the vector state manually initialized, $\mathbf{g}(\mathbf{b}_0)$ corresponds to the

geometric model placed at position $\mathbf{b}_0$ and $\mathbf{y}_0$ corresponds to the first video frame where initialization takes place.

To make this observation model robust to most of the environment variations during the tracking process, due to lightning changes, grimaces, occlusions, and even the pose, it was then necessary to introduce an adaptation to the reference. In order to do that, we perform an update of the reference at each time using the following expression:

$$\mathbf{x}_{t+1}^{(ref)} = \alpha \mathbf{x}_t^{(ref)} + (1-\alpha)\hat{\mathbf{x}}_t \tag{4.7}$$

where $\hat{\mathbf{x}}_t$ denotes the vector resulting from the estimation of the state vector $\hat{\mathbf{b}}_t$ and α denotes a forgetting factor. This forgetting factor has been determined from experiments and fixed to 0.99 for all the tests, proving to be robust in all the video sequences tested.

The advantage of this kind of normalization is that the resulting vector presents always the same characteristics, giving us a facial textures that encodes the characteristics of one person, but with all the facial features, such as nose and eyes position, always in the same place. This will show to be useful when working with multiple people in order to obtain a general model.

4.3 Canonical Correlation Analysis

In a video sequence depicting a moving face, the tracking consists in estimating the state vector $\hat{\mathbf{b}}_t$ at frame t that best matches the measurements in the current image $\mathbf{Y}_t$. To perform this, we consider two approaches, to learn a relation between a set of perturbed state parameters and the associated image patches. One is based on Canonical Correlation Analysis, and the other one on kernel CCA. In this section we will briefly introduce CCA and KCCA, as well as the way we use them for face tracking.

4.3.1 Canonical Correlation Analysis

Canonical correlation analysis is a way of identifying and quantifying the linear relationship between two data sets of random variables. CCA can be seen as the problem of finding pairs of directions for two sets of variables, one for $\mathbf{A}_1$ representing m examples of the d-dimensional vector $\mathbf{\Delta x}_i$ and the other for $\mathbf{A}_2$ representing m examples of the p-dimensional vector $\mathbf{\Delta b}_i$, such that the correlation between the projections of the variables onto these direction vectors are mutually maximized. Let us notice that for this training, i denotes the index of the database examples, with $i = 1 \ldots m$. In our case the $\mathbf{\Delta x}_i$ vectors will represent the residual between the face vectors with respect to a reference $\mathbf{\Delta x}_i = \mathbf{x}_i - \mathbf{x}^{(ref)}$, and $\mathbf{\Delta b}_i$ the corresponding perturbation of the state vector parameters $\mathbf{\Delta b}_i = \mathbf{b}_i - \mathbf{b}_j$, $i \neq j$. The canonical correlation coefficients can be calculated directly from the two data sets. An important property of canonical correlations is that they are

invariant with respect to affine transformations of the variables. The maximum number of correlations that can be found is equal to the minimum of the data sets' row dimension $min(d, p)$. If we map our data to the directions $\mathbf{w}_1$ and $\mathbf{w}_2$ we obtain two new vectors defined as:

$$\mathbf{z}_1 = \mathbf{A}_1^T \mathbf{w}_1 \quad \text{and} \quad \mathbf{z}_2 = \mathbf{A}_2^T \mathbf{w}_2 \tag{4.8}$$

These vectors are called the *scores* or the *canonical variates* [9; 26; 87] and we are interested in maximizing the correlation between them, which is defined as:

$$\rho = \frac{\mathbf{z}_2^T \mathbf{z}_1}{\sqrt{\mathbf{z}_2^T \mathbf{z}_2}\sqrt{\mathbf{z}_1^T \mathbf{z}_1}} \tag{4.9}$$

We must point out that ρ is not affected if we scale the canonical variates, which means that we can maximize (4.9) subject to the constraints $\|\mathbf{z}_1\| = \|\mathbf{z}_2\| = 1$ and formulate our problem in a Lagrangian form. This can be expressed as the following optimization problems:

$$\begin{cases} \min \|\mathbf{z}_1 - \mathbf{z}_2\|^2 \\ \|\mathbf{z}_1\| = \|\mathbf{z}_2\| = 1 \end{cases} \Leftrightarrow \begin{cases} \max \mathbf{z}_2^T \mathbf{z}_1 \\ \|\mathbf{z}_1\| = \|\mathbf{z}_2\| = 1 \end{cases} \tag{4.10}$$

which are equivalent. The detailed optimization can be found in the appendix A, in this section we will present only the most important equations.

Our problem consists in finding the vectors $\mathbf{w}_1$ and $\mathbf{w}_2$ that maximize (4.10). In order to do that, we can define the product between the data matrices as:

$$\Sigma_{11} = \mathbf{A}_1 \mathbf{A}_1^T, \Sigma_{22} = \mathbf{A}_2 \mathbf{A}_2^T \text{ and } \Sigma_{21} = \mathbf{A}_2 \mathbf{A}_1^T$$

The matrices Σ_{11}, Σ_{22}, and Σ_{21} have a statistical interpretation if we consider the column vectors of $\mathbf{A}_1$ and $\mathbf{A}_2$ matrices as outcomes of an independent random vector, then $\Sigma_{11}/(m-1)$, $\Sigma_{22}/(m-1)$, and $\Sigma_{21}/(m-1)$ may be considered as the unbiased estimator of the covariance matrices.

Assuming Σ_{11} and Σ_{22} invertible, we obtain the relation that exists between the two direction vectors $\mathbf{w}_1$ and $\mathbf{w}_2$:

$$\Sigma_{11}^{-1}\Sigma_{12}\mathbf{w}_2 = \rho \mathbf{w}_1 \quad \text{and} \quad \Sigma_{22}^{-1}\Sigma_{21}\mathbf{w}_1 = \rho \mathbf{w}_2 \tag{4.11}$$

which take us to the equations:

$$\Sigma_{11}^{-1}\Sigma_{12}\Sigma_{22}^{-1}\Sigma_{21}\mathbf{w}_1 = \rho^2 \mathbf{w}_1 \tag{4.12}$$

$$\Sigma_{22}^{-1}\Sigma_{21}\Sigma_{11}^{-1}\Sigma_{12}\mathbf{w}_2 = \rho^2 \mathbf{w}_2 \tag{4.13}$$

Using these two equations we can calculate the direction vectors $\mathbf{w}_1$ and $\mathbf{w}_2$ that correspond respectively to the eigenvalues of the matrices $\Sigma_{11}^{-1}\Sigma_{12}\Sigma_{22}^{-1}\Sigma_{21}$ and $\Sigma_{22}^{-1}\Sigma_{21}\Sigma_{11}^{-1}\Sigma_{12}$. The number of non zero eigenvalues to these equations are limited to the smallest dimensionality of $\mathbf{A}_1$ and $\mathbf{A}_2$ ($min(n, p)$) [9] and the corresponding eigenvectors $\mathbf{w}_1$ and $\mathbf{w}_2$ are the *canonical correlation basis vectors*. These equations can be also formulated as one single eigenvalue equation:

$$\mathbf{B}^{-1}\mathbf{A}\mathbf{w} = \rho\mathbf{w} \tag{4.14}$$

where $\mathbf{w} = \begin{pmatrix} \mathbf{w}_1 \\ \mathbf{w}_2 \end{pmatrix}$, and

$$\mathbf{A} = \frac{1}{m-1}\begin{pmatrix} 0 & \mathbf{A}_1^T\mathbf{A}_2 \\ \mathbf{A}_2^T\mathbf{A}_1 & 0 \end{pmatrix}, \quad \mathbf{B} = \frac{1}{m-1}\begin{pmatrix} \mathbf{A}_1^T\mathbf{A}_1 & 0 \\ 0 & \mathbf{A}_2^T\mathbf{A}_2 \end{pmatrix} \tag{4.15}$$

are the estimates of the covariance matrices.

As is explained in [60], ρ corresponds also to the solution of the *Rayleigh quotient*, which will be useful at a later stage. It can be defined in terms of the estimates of the covariance matrices defined in equation (4.15) as:

$$r = \frac{\mathbf{w}^T\mathbf{A}\mathbf{w}}{\mathbf{w}^T\mathbf{B}\mathbf{w}} \tag{4.16}$$

Solution of the Canonical Correlation Analysis equations from data matrices As we have the data matrices $\mathbf{A}_1$ and $\mathbf{A}_2$ we can use the numerically more robust method proposed in [87] to reduce the number of matrix operations. In this approach, we perform firstly the singular value decomposition of the data matrices $\mathbf{A}_1 = \mathbf{U}_1\mathbf{D}_1\mathbf{V}_1^T$ and $\mathbf{A}_2 = \mathbf{U}_2\mathbf{D}_2\mathbf{V}_2^T$. If we replace in (4.13) we obtain after some manipulations:

$$((\mathbf{V}_1^T\mathbf{V}_2)^T(\mathbf{V}_1^T\mathbf{V}_2) - \rho^2\mathbf{I})\mathbf{D}_2\mathbf{U}_2^T\mathbf{w}_2 = 0 \tag{4.17}$$

Introducing the singular value decomposition: $\mathbf{V}_1^T\mathbf{V}_2 = \mathbf{U}\mathbf{D}\mathbf{V}^T$, we can solve the last equation, and after some rearrangement we arrive at:

$$(\mathbf{D}^2 - \rho^2\mathbf{I})\mathbf{V}^T\mathbf{D}_2\mathbf{U}_2^T\mathbf{w}_2 = 0 \tag{4.18}$$

Because $\mathbf{D}^2$ is a diagonal matrix, equation (4.18) shows that $\mathbf{w}_2$ are the column vectors of $(\mathbf{V}^T\mathbf{D}_2\mathbf{U}_2^T)^{-1} = \mathbf{U}_2\mathbf{D}_2^{-1}\mathbf{V}$. With a similar procedure we can show that $\mathbf{w}_1$ are the column vectors of $(\mathbf{U}^T\mathbf{D}_1^{-1}\mathbf{U}_1^T)^{-1} = \mathbf{U}_1\mathbf{D}_1^{-1}\mathbf{U}$. We denote then $\mathbf{W}_1$ and $\mathbf{W}_2$ as the matrices containing respectively the direction vectors $\mathbf{w}_1$ and $\mathbf{w}_2$ that maximize the correlation of (4.9), and these matrices can be written as:

$$\mathbf{W}_1 = \mathbf{U}_1\mathbf{D}_1^{-1}\mathbf{U} \quad \text{and} \quad \mathbf{W}_2 = \mathbf{U}_2\mathbf{D}_2^{-1}\mathbf{V} \tag{4.19}$$

The principal advantage of this procedure, reported in [87], is to avoid the calculation of the covariance matrices, the matrix multiplications in (4.15) and the calculation of the two inverses. Instead of that, it is necessary to perform three singular value decompositions.

4.3.2 Kernel Canonical Correlation Analysis.

As explained in chapter 3, kernel methods are used when a linear relationship can not be found in the original data. They map the data into a high dimensional feature space, in order to find a linear relations between the data in that space.

$$\phi : \mathbb{R}^p \mapsto \mathbb{R}^s, s > p. \tag{4.20}$$

The use of a kernel approach depends on whether or not the *kernel-trick* can be applied. That means that the optimization criterion must be expressed in such a way that it only uses dot products of the transformed input data. If this is true we can express the dot product in the feature space in terms of kernel functions in input space, i.e., $\phi(\mathbf{x}_1)^T\phi(\mathbf{x}_2) = k(\mathbf{x}_1, \mathbf{x}_2)$. A sufficient condition for a kernel-function to correspond to a dot product in a feature space is given by *Mercers Theorem*. In [60], it is shown that CCA can be completely expressed in terms of dot products and that vector $\mathbf{w}_1$ and vector $\mathbf{w}_2$ lie in the span of the training data. With this and if as in section 4.3, we define $\mathbf{A}_1$ and $\mathbf{A}_2$ as the centered data sets of dimension $n \times m$ and of dimension $p \times m$ respectively, we can write vectors $\mathbf{f}_1, \mathbf{f}_2 \in \mathbb{R}^m$,so that $\mathbf{w}_1 = \mathbf{A}_1^T\mathbf{f}_1$ and $\mathbf{w}_2 = \mathbf{A}_2^T\mathbf{f}_2$. Thus, we can reformulate equation (4.16) as:

$$r = \frac{\begin{pmatrix} \mathbf{f}_1^T & \mathbf{f}_2^T \end{pmatrix} \begin{pmatrix} 0 & \mathbf{A}_1^T\mathbf{A}_1\mathbf{A}_2^T\mathbf{A}_2 \\ \mathbf{A}_2^T\mathbf{A}_2\mathbf{A}_1^T\mathbf{A}_1 & 0 \end{pmatrix} \begin{pmatrix} \mathbf{f}_1 \\ \mathbf{f}_2 \end{pmatrix}}{\begin{pmatrix} \mathbf{f}_1^T & \mathbf{f}_2^T \end{pmatrix} \begin{pmatrix} \mathbf{A}_1^T\mathbf{A}_1\mathbf{A}_1^T\mathbf{A}_1 & 0 \\ 0 & \mathbf{A}_2^T\mathbf{A}_2\mathbf{A}_2^T\mathbf{A}_2 \end{pmatrix} \begin{pmatrix} \mathbf{f}_1 \\ \mathbf{f}_2 \end{pmatrix}} \tag{4.21}$$

From equation (4.21) we can see that the *Rayleigh quotient* can be expressed in terms of dot products, so we can apply the *kernel trick* to it. What we are looking for is to apply the CCA to the vectors $\phi_1(\mathbf{A}_1) = (\phi_1(\mathbf{a}_1^{(1)})...\phi_1(\mathbf{a}_m^{(1)}))$ and $\phi_2(\mathbf{A}_2) = (\phi_2(\mathbf{a}_1^{(2)})...\phi_2(\mathbf{a}_m^{(2)}))$, which are two non linear mappings as described in (4.20).

The formulation obtained in (4.21) makes the application of the *kernel trick* straightforward. To do this we define the kernel matrices $\mathbf{K}_1, \mathbf{K}_2$ as $\mathbf{K}_{1ij} = \phi_1(\mathbf{a}_i^{(1)})\phi_1(\mathbf{a}_j^{(1)})^T$ and $\mathbf{K}_{2ij} = \phi_2(\mathbf{a}_i^{(2)})\phi_2(\mathbf{a}_j^{(2)})^T$, and substitute them in (4.21), obtaining:

$$r = \frac{\begin{pmatrix} \mathbf{f}_{\phi_1}{}^T & \mathbf{f}_{\phi_2}{}^T \end{pmatrix} \begin{pmatrix} 0 & \mathbf{K}_1\mathbf{K}_2 \\ \mathbf{K}_2\mathbf{K}_1 & 0 \end{pmatrix} \begin{pmatrix} \mathbf{f}_{\phi_1} \\ \mathbf{f}_{\phi_2} \end{pmatrix}}{\begin{pmatrix} \mathbf{f}_{\phi_1}{}^T & \mathbf{f}_{\phi_2}{}^T \end{pmatrix} \begin{pmatrix} \mathbf{K}_1^2 & 0 \\ 0 & \mathbf{K}_2^2 \end{pmatrix} \begin{pmatrix} \mathbf{f}_{\phi_1} \\ \mathbf{f}_{\phi_2} \end{pmatrix}} \tag{4.22}$$

where $\mathbf{f}_{\phi_1}$ and $\mathbf{f}_{\phi_2}$ are the coefficients of the linear expansion of the principal vectors $\mathbf{w}_{\phi_1}$ and $\mathbf{w}_{\phi_2}$ in terms of the transformed data, i.e., $\mathbf{w}_{\phi_1} = \phi_1(\mathbf{A}_1)^T\mathbf{f}_{\phi_1}$ and $\mathbf{w}_{\phi_2} = \phi_2(\mathbf{A}_2)^T\mathbf{f}_{\phi_2}$.

We can formulate this problem as one simple eigenvalue equation as in (4.14), and then solve it as the traditional CCA problem to obtain $\mathbf{f}_{\phi_1}$ and $\mathbf{f}_{\phi_2}$.

However, KCCA frequently yields to useless results because we work with a finite number of points in a high dimensional feature space. To force useful

solutions a method consists in introducing a penalizing factor depending on $\mathbf{K}$ in the norms of the associated weights (for more details see [39]). If we introduce this penalizing factor, we obtain from (4.9):

$$\rho = \frac{\mathbf{f}_{\phi_2}^T \mathbf{K}_2^T \mathbf{K}_1 \mathbf{f}_{\phi_1}}{\sqrt{\mathbf{f}_{\phi_2}^T \mathbf{K}_2^2 \mathbf{f}_{\phi_2} + \kappa \mathbf{f}_{\phi_2}^T \mathbf{K}_2 \mathbf{f}_{\phi_2}} \sqrt{\mathbf{f}_{\phi_1}^T \mathbf{K}_1^2 \mathbf{f}_{\phi_1} + \kappa \mathbf{f}_{\phi_1}^T \mathbf{K}_1 \mathbf{f}_{\phi_1}}} \tag{4.23}$$

where κ is an scalar that was determined empirically based on the approach presented in [60]. After a development similar as the previously done for the CCA, we arrive to equations equivalent to (4.13) and (4.12):

$$(\mathbf{K}_1 + \kappa \mathbf{I})^{-1} \mathbf{K}_2 (\mathbf{K}_2 + \kappa \mathbf{I})^{-1} \mathbf{K}_1 \mathbf{f}_{\phi_1} = \rho^2 \mathbf{f}_{\phi_1} \tag{4.24}$$

$$(\mathbf{K}_2 + \kappa \mathbf{I})^{-1} \mathbf{K}_1 (\mathbf{K}_1 + \kappa \mathbf{I})^{-1} \mathbf{K}_2 \mathbf{f}_{\phi_2} = \rho^2 \mathbf{f}_{\phi_2} \tag{4.25}$$

We can then obtain the vectors $\mathbf{f}_{\phi_1}$ and $\mathbf{f}_{\phi_2}$ as the eigenvectors of these equations.

4.4 Pose estimation and tracking.

Once we have introduced the CCA and the KCCA, we can proceed to describe our algorithm to estimate the 3D pose. It consists of three steps: Initialization, Training process, and Tracking process. It is important to say that in this case, the vector $\boldsymbol{\tau_a}$ is supposed to be constant: the face is seen as a rigid object, with a fixed expression.

The facial 3D pose state vector $\mathbf{b}$ is then given, as stated in (4.6), by:

$$\mathbf{b} = [\theta_x, \theta_y, \theta_z, t_x, t_y, t_z] \tag{4.26}$$

where the θ elements stand for the rotations and the t elements stand for the translations.

4.4.1 Initialization.

Initializing consists in placing and fitting manually in the first frame the *Candide* model onto the person's face, obtaining the initial state vector $\mathbf{b}_0$, and the reference vector $\mathbf{x}_0^{(ref)} = \mathcal{W}(\mathbf{g}(\mathbf{b}_0), \mathbf{Y}_0)$, as described in section 4.2, and showed in figure 4.3. This vector will be kept during all the tracking process and will be updated accordingly to equation (4.7). This reference is also used to generate synthesized views of the face for the training process, as next explained.

4.4.2 Training process based on the first image.

The training process consists in finding a relationship between the variations in the state vector $\Delta \mathbf{b}_t$ and the corresponding variations in the input vector with respect to the reference $\Delta \mathbf{x}_t = \mathbf{x}_t - \mathbf{x}_t^{(ref)}$. We assume that:

$$\Delta \mathbf{b}_t = \mathbf{b}_t - \mathbf{b}_{t-1} = \mathbf{G} \Delta \mathbf{x}_t \tag{4.27}$$

where $\mathbf{G}$ explains the linear relationship between the variations in the state vector and the corresponding variations in the input vector with respect to the reference. The learning of $\mathbf{G}$ has been obtained through CCA and KCCA approaches. For that, we consider m different variation vectors $\Delta \mathbf{b}_i^{training}$ from a non-regular grid chosen empirically around the vector state obtained at initialization. A representation of this grid is depicted in Figure 4.5, where we can see the points for a two dimensional vector. This non regular grid is dense close to the state vector $\mathbf{b}_0$, corresponding to the origin of the grid, and as we get away from it, it becomes sparse. Then, from equation (4.27) we can write the following expression:

$$\mathbf{b}_i^{training} = \mathbf{b}_0 + \Delta \mathbf{b}_i^{training}; i = 1 \dots m \tag{4.28}$$

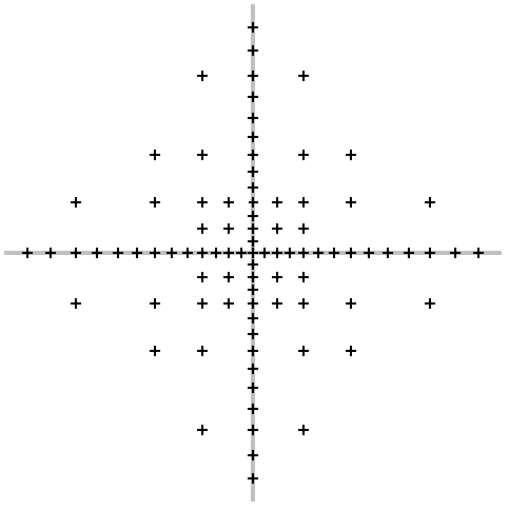


Figure 4.5: 2D representation of the sampled *Candide* parameters.

From these training state vectors we acquire m observation vectors $\mathbf{x}_i^{training} = \mathcal{W}(\mathbf{g}(\mathbf{b}_i^{training}), \mathbf{Y}_0)$, obtained from patches similar to those depicted in figure 4.6. It is with these two sets of data that we create the matrix $\mathbf{A}_1 = \left\{ \mathbf{a}_1^{(1)}, \dots, \mathbf{a}_m^{(1)} \right\}$, of dimension $d \times m$, whose columns contain the difference between the training observation vectors and the reference vector, $\mathbf{a}_i^{(1)} = \mathbf{x}_i^{training} - \mathbf{x}_0^{(ref)}$, and the matrix $\mathbf{A}_2 = \left\{ \mathbf{a}_1^{(2)}, \dots, \mathbf{a}_m^{(2)} \right\}$, of dimension $p \times m$ with $p = 6$, whose columns contain the variation in the state vector $\mathbf{a}_i^{(2)} = \Delta \mathbf{b}_i^{training}$. For this particular case these variations correspond to the three rotations and three translations of the 3D pose.

Figure 4.6: Examples of training images. Top: examples of rotations. Bottom: examples of translations. In the top left corner of each image we can see the corresponding expression free-patch used to obtain the observation vector $\mathbf{x}_t$.

Training the CCA As it is stated in equation (4.27), we want to find a relationship between the variations in the state vector $\Delta\mathbf{b}_t$ and the corresponding variations in the input vector with respect to the reference $\Delta\mathbf{x}_t = \mathbf{x}_t - \mathbf{x}_t^{(ref)}$. Based on the optimization used to obtain the CCA in equation (4.10), and using the matrices $\mathbf{A}_1$ and $\mathbf{A}_2$ containing the training data, we can assume that:

$$\mathbf{w}_1^T\mathbf{A}_1 = \mathbf{w}_2^T\mathbf{A}_2 \tag{4.29}$$

We know that the optimization in equation (4.10), if developed, can be written as $\|\mathbf{A}_1^T\mathbf{w}_1 - \mathbf{A}_2^T\mathbf{w}_2\|^2 = 2(1-\rho)$. In our case, we assumed $\rho \approx 1$ which means that there is a high correlation between the two projected sets [38].

As the training examples are chosen to describe all the possible variations of the model parameters, we can consider that the learned CCA direction vectors are valid not only for the training data, but for also for $\Delta\mathbf{b}_t$ and $\mathbf{x}_t$, so if we substitute them in (4.29) we come to:

$$\mathbf{w}_2^T\Delta\mathbf{b}_t = \mathbf{w}_1^T(\mathbf{x}_t - \mathbf{x}_t^{(ref)}) \tag{4.30}$$

This is true for all the *canonical variates*, so $\mathbf{W}_2^T\Delta\mathbf{b}_t = \mathbf{W}_1^T(\mathbf{x}_t - \mathbf{x}_t^{(ref)})$ is also true, and matrices $\mathbf{W}_1$ and $\mathbf{W}_2$ contain the full set of *canonical correlation direction vectors*. We can replace equations (4.19) in the last equation, and after some mathematical manipulation we arrive to:

$$\Delta\mathbf{b}_t = \mathbf{U}_2\mathbf{D}_2\mathbf{V}\mathbf{U}^T\mathbf{D}_1^{-1}\mathbf{U}_1^T(\mathbf{x}_t - \mathbf{x}_t^{(ref)}) \tag{4.31}$$

where the matrices correspond to those explained in Section 4.3.1. From equation (4.31), we can conclude that the training process consists in computing:

$$\mathbf{G} = \mathbf{U}_2\mathbf{D}_2\mathbf{V}\mathbf{U}^T\mathbf{D}_1^{-1}\mathbf{U}_1^T \tag{4.32}$$

that will be used during the tracking process for estimating the variation vector.

It is important to notice that the matrix $\mathbf{G}$ is strongly dependent of the training points selected. This means that this matrix will only valid for a certain rotation and translation interval and it will be also dependent of the correctness of the initialization. From experiments we have seen that indeed there exists a linear relationship that is well explained by means of the CCA approach. However, we wanted to see if a non linear approach could improve the results already obtained. This is why we have decided to use a KCCA approach, that is presented next, based on the approach given by [60].

Training the KCCA For the training process, we need to obtain vectors $\mathbf{f}_\phi$ and $\mathbf{g}_\theta$ as described in the last section, using the same data matrices as the ones described for the CCA approach. It is important to point out that in the case of the variation vectors, we did not use any kernel. Once this is done, the starting point is equation (4.29) that can be developed according to the KCCA as:

$$\mathbf{K}_t\mathbf{f}_{\phi_1} \approx \mathbf{A}_2\mathbf{A}_2^T\mathbf{f}_{\phi_2} \tag{4.33}$$

With a similar development, we can assume that the tracking data satisfies also this equation and after some mathematical manipulation we obtain:

$$\Delta\mathbf{b}_t = \mathbf{K}_t\mathbf{f}_{\phi_1}(\mathbf{A}_2^T\mathbf{f}_{\phi_2})^{-1} \tag{4.34}$$

Here we can see that the $\mathbf{K}_t$ corresponds to the kernel matrix at time t obtained between the training vectors $\mathbf{x}_i^{training} = \mathcal{W}(\mathbf{g}(\mathbf{b}_i^{training}), \mathbf{Y}_0),\ i = 1..m$ and the current stabilized face image $\mathbf{x}_t = \mathcal{W}(\mathbf{g}(\mathbf{b}_{t-1}), \mathbf{Y}_t)$. However, for the algorithm implementation, we perform a linear regression between the result of the product of the matrix kernel $\mathbf{K}_t\mathbf{f}_{\phi_1}$ and the variation vector $\mathbf{A}_2$, so that we have the update equation:

$$\Delta\mathbf{b}_t = \mathbf{K}_t\mathbf{f}_{\phi_1}\mathbf{G} \tag{4.35}$$

being the matrix $\mathbf{G}$ obtained from the training kernel matrix $\mathbf{K}$. The kernel used in our work was the Gaussian Radial Basis Function:

$$\mathbf{K}(\mathbf{x}_i, \mathbf{x}_j) = \exp(-\frac{\|\mathbf{x}_i - \mathbf{x}_j\|^2}{2\sigma^2}) \tag{4.36}$$

In our experiments, the two parameters σ (eq. (4.36)) and κ (equation (4.23)) are set respectively to 0.009 and 0.003. They were obtained empirically from simulations using training sequences, and based on the method used in [60] to estimate a starting point.

4.4.3 Tracking process.

We can then formulate our tracking process as the estimation of the state variation $\Delta \mathbf{b}_t$ from the current expression-free patch vector $\mathbf{x}_t = \mathcal{W}(\mathbf{g}(\hat{\mathbf{b}_t}), \mathbf{Y}_t)$, obtained from the current frame $\mathbf{Y}_t$ by means of the state at the preceding time $\mathbf{b}_{t-1}$, and compared to a reference $\mathbf{x}_t^{(ref)}$ that is updated at each time once the pose is correctly estimated.

This is done by using the matrix $\mathbf{G}$ obtained during the training process (Eq. (4.32)) for the linear case, or using the matrix $\mathbf{G}$ obtained during the training process, and matrix $\mathbf{K}_t$, obtained at each frame arrival. So, we can write our update equation as:

$$\hat{\mathbf{b}}_t = \mathbf{b}_{t-1} + \mathbf{G}(\mathbf{x}_t - \mathbf{x}_t^{(ref)}) \tag{4.37}$$

for the CCA, and for the KCCA:

$$\hat{\mathbf{b}}_t = \mathbf{b}_{t-1} + \mathbf{K}_t\mathbf{f}_\phi\mathbf{G} \tag{4.38}$$

This $\hat{\mathbf{b}_t}$ is the estimate of the state vector and we use it to obtain a new observation vector $\hat{\mathbf{x}}_t = \mathcal{W}(\mathbf{g}(\hat{\mathbf{b}_t}), \mathbf{Y}_t)$. Then we compare this observation vector $\mathbf{x}_t$ with the reference $\mathbf{x}_t^{(ref)}$ to obtain a measure of the error:

$$e(\hat{\mathbf{b}_t}) = \sum_{i=1}^{d} \left(\hat{\mathbf{x}}_{t,i} - \mathbf{x}_t^{(ref)}\right)^2. \tag{4.39}$$

Before presenting the algorithm, we need to remember equation (4.7):

$$\mathbf{x}_{t+1}^{(ref)} = \alpha\mathbf{x}_t^{(ref)} + (1-\alpha)\hat{\mathbf{x}}_t.$$

and then we can present the algorithm as:

1. Initialize $e(\hat{\mathbf{b}}_{t-1}) = 1$, and a counter $i = 0$.
2. Estimate $\hat{\mathbf{b}_t}$ with the equation (4.37) for the CCA or (4.38) for the KCCA.
3. Estimate $\hat{\mathbf{x}}_t = \mathcal{W}(\mathbf{g}(\hat{\mathbf{b}_t}), \mathbf{Y}_t)$.
4. Estimate $e(\hat{\mathbf{b}_t})$ with equation (4.39).
5. Increase i by one.
6. If $e(\hat{\mathbf{b}}_{t-1}) - e(\hat{\mathbf{b}_t}) > 0$ and $i < 5$, then $e(\hat{\mathbf{b}}_{t-1}) = e(\hat{\mathbf{b}_t})$ and $\hat{\mathbf{b}}_{t-1} = \hat{\mathbf{b}_t}$ and return to 2
7. Else update the reference accordingly to equation (4.7).

As we can see we continue this loop until there is not an improvement of the error, or a certain number of iterations are performed, as other algorithms do, like in [5]. In our case we have found from experiments that 5 iterations represent a good choice, as will be showed below.

We have fixed the forgetting factor in the update equation (4.7) to $\alpha = 0.99$ and it gives good experimental results for all the video sequences used to test the algorithm.

It can be seen that the proposed algorithm is easy to implement and from a practical point of view, it is intended to perform very fast, at least in the linear version of the algorithm, as will be corroborated with the results presented in the following section.

4.5 Implementation and results

The algorithm has been implemented on a PC with a 3.0 GHz Intel Pentium IV processor and a NVIDIA Quadro NVS 285 graphic card. Our non optimized code uses OpenGL for texture mapping and OpenCV [1] for video capture.

Based on the algorithm described in section 4.4, we have implemented, for comparison purposes, the two algorithms to estimate six head pose parameters. The first version uses the CCA, and the second version uses the KCCA. The size of the stabilized face image is of of 96×72 pixels, thus, the resulting observation vector is of size $d = 6912$.

For training, we use $m = 317$ training state vectors with the corresponding appearance variations for the pose, obtained synthetically by means of 3D *Candide* model, the first frame and the OpenGL library. These vectors correspond to variations of $\pm 20^\circ$ for the rotations, $\pm 10.5\%$ of the face width for translations. We chose these points empirically, from a symmetric grid centered on the initial state vector. The sampling is dense close to the origin and coarse as it moves away from it (see Figure 4.5). Due to the high dimensionality of our state vectors, we did not use all the combinations between the chosen points. In this implementation, we consider the head as a rigid object only. The facial animation parameters were not implemented to be tracked.

To test the tracker we have used the 45 video sequences used in [48] [2], the annotated talking face video made available from the *FGnet Working Group*[3], as well as some video sequences made with a Winnov analog video camera XC77B/320.

Video sequences provided in [48] are 200 frames long, with a resolution of 320×240, 30 *fps*, taken under uniform illumination. There are 5 subjects performing free head motion including translations and both in-plane and out-of-plane rotations, in 9 different video sequences for each person. Ground truth has been collected via a "Flock of Birds" 3D magnetic tracker. Figure 4.7 shows the estimated pose obtained with the CCA algorithm compared with the ground data.

[1] $http://www.intel.com/technology/computing/opencv/index.htm$

[2] $http://www.cs.bu.edu/groups/ivc/HeadTracking/$

[3] $http://www-prima.inrialpes.fr/FGnet/data/01-TalkingFace/talking\_face.html$

Temporal shifts can be explained because the center of the coordinate systems used in [48] and ours are slightly different. In our case, the three axes cross close to the nose, due to the *Candide* model specification, and in the ground truth data, the 3D magnetic tracker is attached on the subject's head. In other words, for our system the origin is the nose while for the other it is the top of the head. This induced some small differences because what in one system represents a rotation, in the other system represents a rotation and a translation. We check experimentally (on all the provided video sequences) the stability and precision of the tracker and do not observe divergences of the tracker.

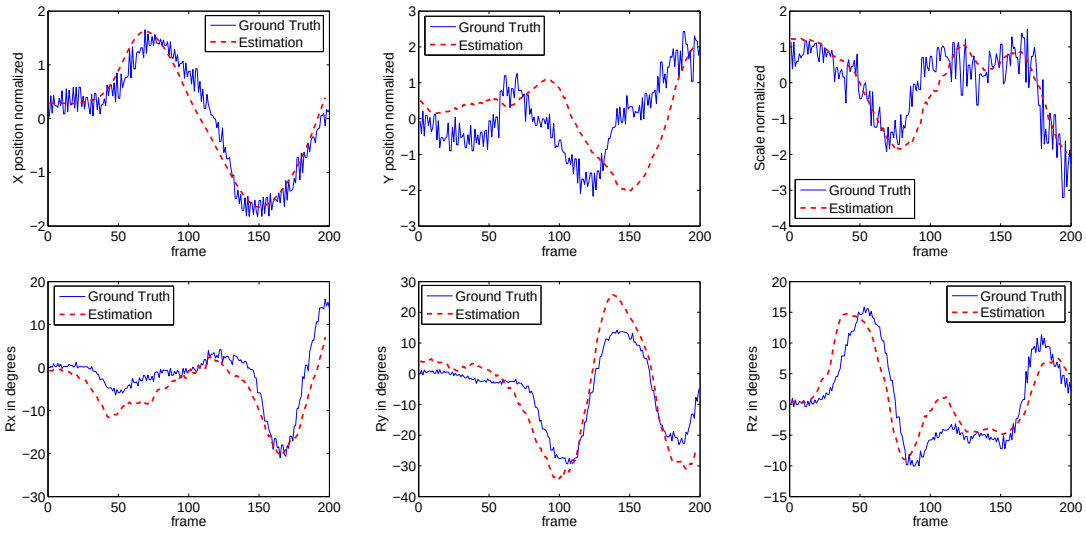


Figure 4.7: Ground truth compared with the CCA algorithm's results.

Figure 4.8 depicts some video frames taken from two of the 45 tracked videos from LaCascia [48], of 200 frames and a webcam video that corresponds to a long video sequence of 734 frames. The CCA and the KCCA algorithm performance did not present any problem when tracking the head in the LaCascia's video sequences, especially because there are not facial gestures involved. In the case of the webcam video, we succeeded in tracking correctly rotations in the y plane going as far as $\pm 35^{\circ}$ (nodding-no head movements). However, when trying to go further, the algorithm could not estimate the correct variation of the angle and got lost. However, we observed from simulations that the effectiveness of this kind of tracker depends on the mask initialization, i.e., the 3D pose of the mask must be correctly initialized in the first frame. Otherwise, the tracker can get lost because the profile of the model affects directly the texture extraction and consequently the state vector predictor.

Another test has been performed using a part of the talking face video. The talking face video sequence consists of 5000 frames, with a resolution of 720×576, taken from a video of a person engaged in conversation. This corresponds to about 200 seconds of recording. The sequence was taken as part of an experiment designed to model the behavior of the face in natural conversation. For practical

Figure 4.8: Results from tracking three video sequences using the CCA algorithm in the first two rows, and the KCCA algorithm in the last one. For the KCCA-based (bottom raw) tracking results correspond from left to right to: frames 305, 332 and 691.

reasons (to display varying parameter values on readable graphs) we used 1720 frames of the video sequence, where the ground truth consists of characteristic 2D facial points annotated semi-automatically. From 68 annotated points per frame, we select 52 points that are closer to the corresponding *Candide* model points, as can be seen in Figure 4.9. These points are not exactly the same as the ones given in the ground truth, so there exists an initial distance between the points.

In order to evaluate the behavior of our algorithm we calculated for each point the standard deviation of the distances between the ground truth and the estimated coordinates divided by the face width. Figure 4.10 depicts the standard deviation over the whole video sequence for each point using the two implementations of our algorithm. We can see that the points with the greater standard deviation correspond to those on the contour of the face. The precision of these points is strongly related to the correctness of the estimated pose parameters.

In Figure 4.11 we can see the result of tracking the talking face over 1720 frames. The importance of this figure is that we can see the evolution of the error during the video. We have seen that the peaks appearing in this figure cor-

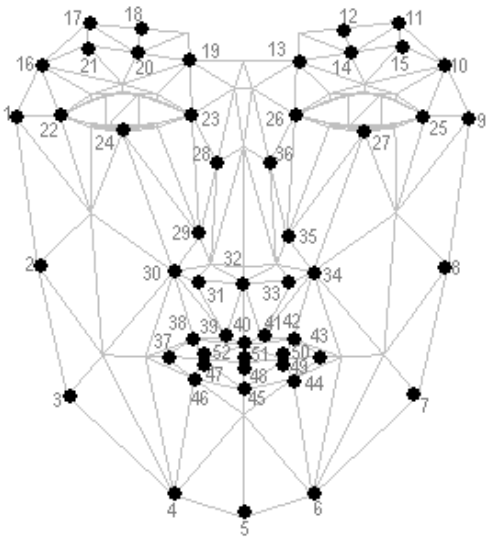


Figure 4.9: *Candide* model with the corresponding talking face ground truth's points used for evaluation.

respond to the moments when there is a facial gesture or an important rotation. However, as it can be seen in the frames 378 and 991, we can consider that these peaks do not represent a significant error between the state vector estimated and the real head pose. If we look at frame 712, where a minimum error is located, we can appreciate that the 3D pose of the person is close to that used as reference.

In figure 4.12, we show the influence of the α parameter in the long term evolution. The fact of keeping the reference vector untouch, leads to an important error accumulation that leads to tracking errors and in the long term could lead to divergences of the pose estimation. However if we update this reference with a low factor we see that it drives also to the degradation of the pose estimation in the long term. We have seen that using a value of $\alpha = 0.99$ was a good compromise in performance and stability for this implementation. This parameter was tested for all the video sequences and it probed to be robust independently of the video sequence used.

The time required per frame processing depends on the video size, as can be seen in the table 4.1. In that table we show also the comparison between the CCA and the KCCA implementation. The presented results include the time required for write/read operations. For the CCA algorithm, the mean value of the 720×576 pixels talking face video was of only 50 ms. The rest of the time was expended principally in the read/write process, and it was dependent on the choice of the video codecs used for both, input and output video. We can also see from this table that the KCCA is more expensive from a computing time point of view.

This time however depends in the number of iterations that the algorithm performs at each step. The maximum number of iterations is fixed to 5. In figure 4.13 we can see the behavior of our algorithm when we vary the number of iterations.

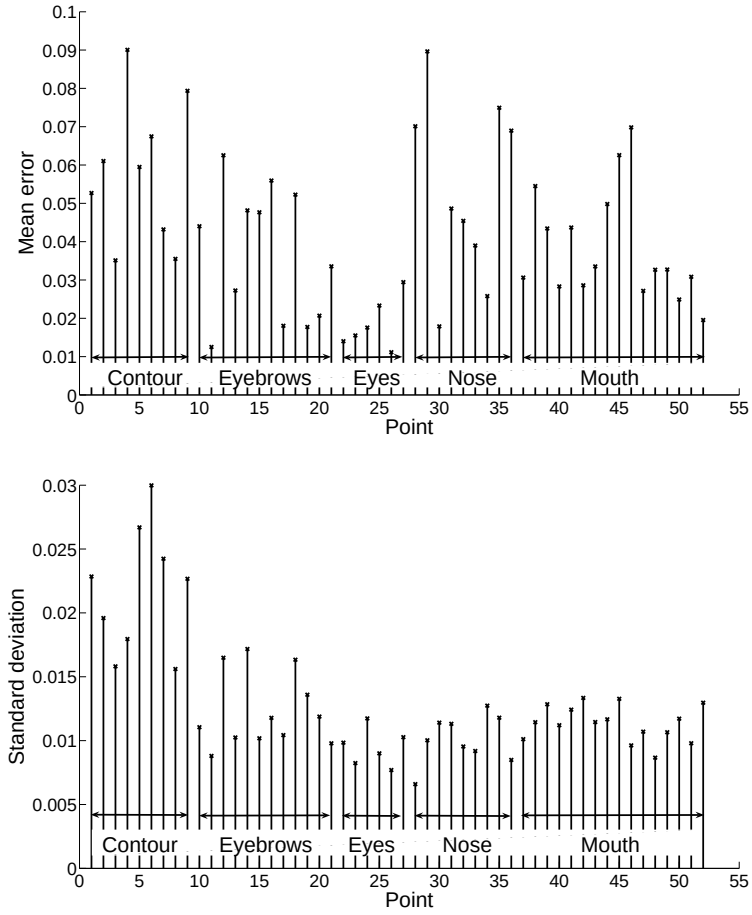


Figure 4.10: Mean and standard deviation of the error for each point provided for the talking face video sequence.

Table 4.1: Comparison of time performance per frame.

Video's size [pixels]	time per frame [s]
CCA 640×480	0.15
CCA 720×576	0.18
KCCA 320×240	2.49

We can see that the more iterations we perform the better the result we obtain, but we have to trade off between correctness and computing time, and we have found that 5 iterations is a good trade off. This number of iterations was founded to be robust because there is a little difference between two consecutive frames, however, if we want to use this algorithm in a real time application where some frames are skipped, the number of iterations should be augmented in order to allow the algorithm to reach the convergence.

We have observed from simulations that this kind of tracker is very dependent of the mask initialization, i.e., for some variations in the mask adaptation we can

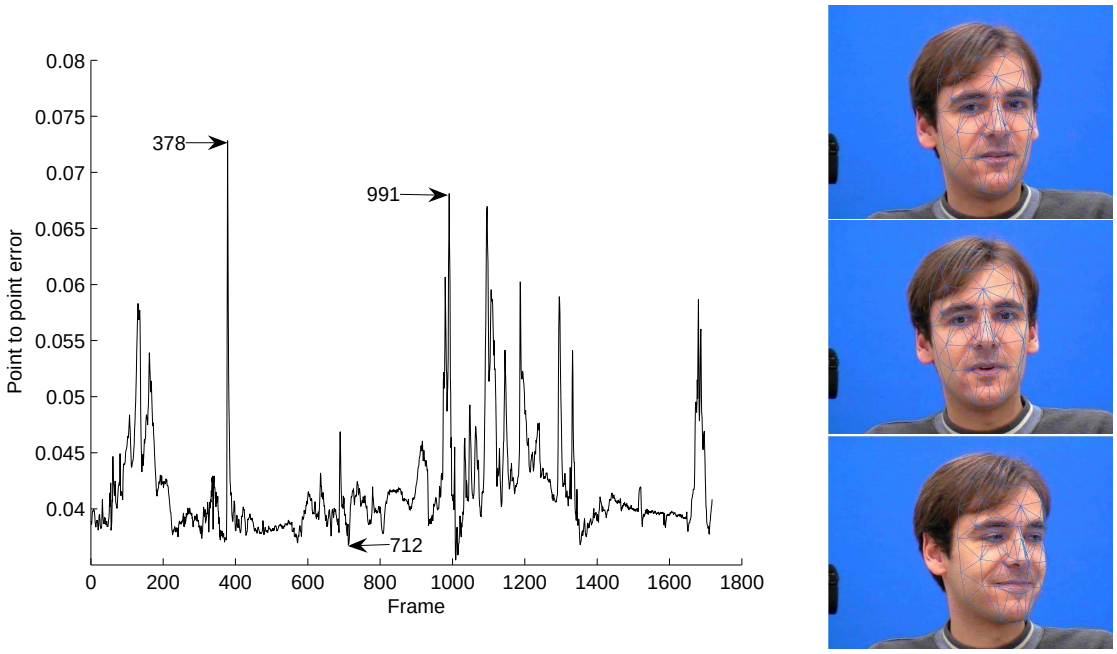


Figure 4.11: Mean point to point error, and three frames of the sequence corresponding to frame 378, to frame 712, and to frame 991 respectively, using the CCA algorithm.

see sometimes the tracker that does not work as expected, especially in low resolution video sequences. The principal problems arrive when there are rotations in θ_x and θ_y, because the profile of the model affects directly the texture extraction and consequently the state vector predictor. To make the algorithm more robust we had introduced a depth's scale factor, and this gave more robustness to the initialization and rotation.

Figure 4.14 displays the six pose parameter estimates against time t, for the face sequence in the last row of figure 4.8, using the CCA and KCCA-based trackers. These plots reveal few differences between the two trackers. We observe that the results obtained with the KCCA-based tracker are slightly more precise and robust for head rotations close to 40 degrees around the y axis. However, this amelioration implies a more complex algorithm that requires more time between frame processing. For the cases where there is not a strong head rotation, both algorithms present the same results.

4.6 Conclusions.

We have seen that the algorithm proposed in our work estimates correctly the 3D pose of faces using the CCA and the KCCA version that were implemented. This pose estimation is correctly conducted in long video sequences of 5000 frames, as the full talking face sequence, but in order to make the graphics more readable, we present the results obtained with the 1700 frames version of the talking face

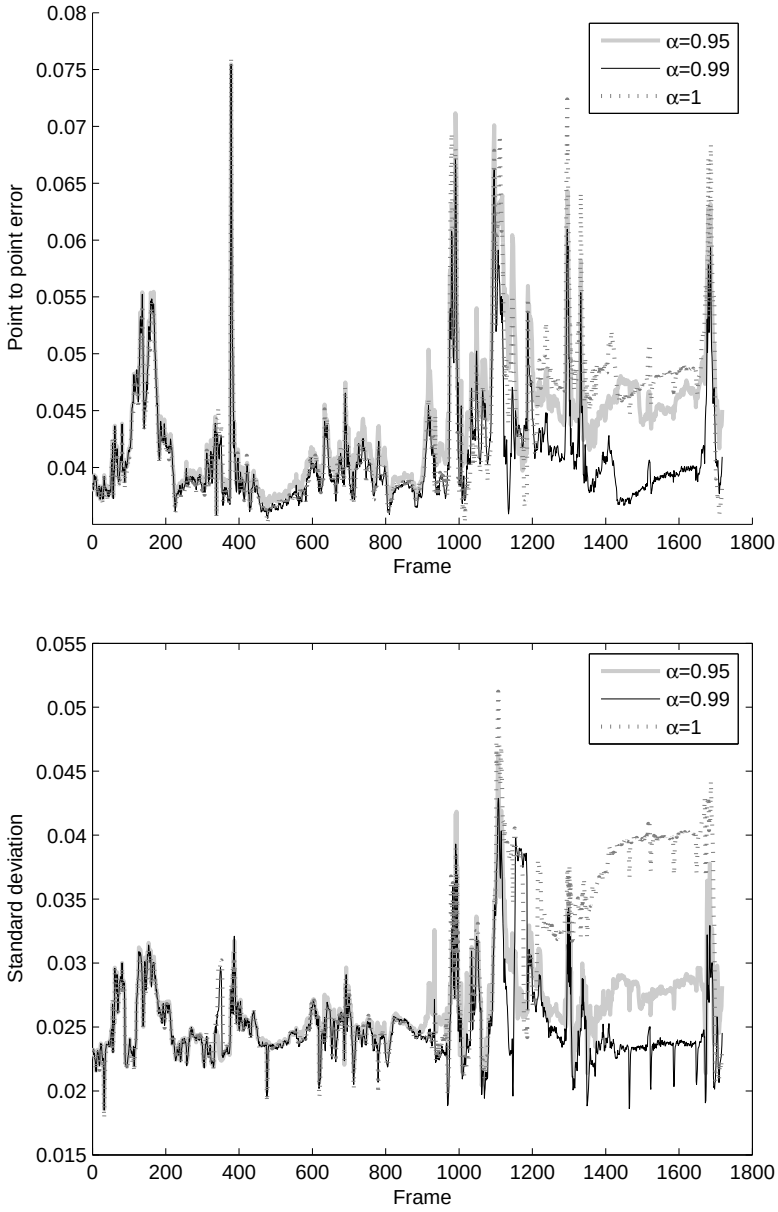


Figure 4.12: Influence of α values in (4.7) over the point to point error and the standard deviation of a long video sequence.

sequence. This confirms the stability in time of this approach, even if the matrix obtained during the training is not updated. This presents the advantage that the resulting implementation is simple, fast and reliable, and supports some perturbations that are not included during training, such as facial gesture. The results obtained in small video sequences with more important 3D pose variations but without facial gesture also confirmed the robustness of this algorithm in the limits of the training process. Indeed, we have seen that the algorithm can track

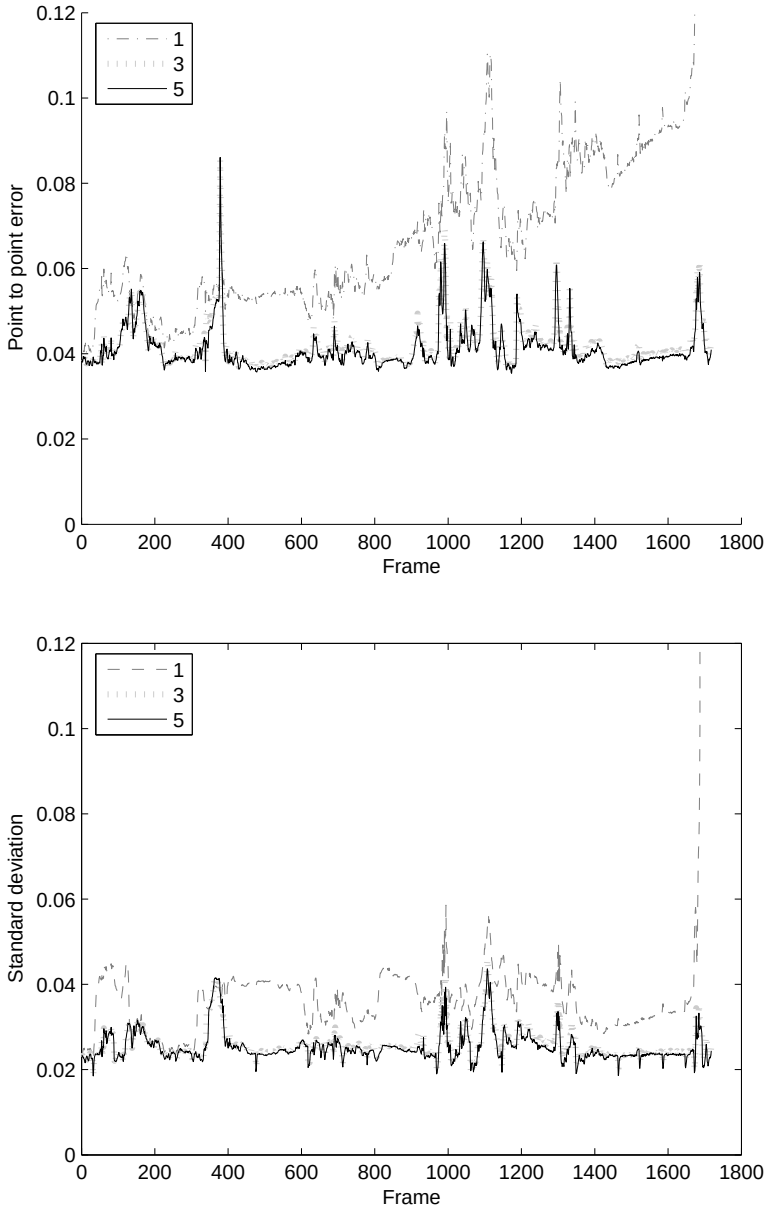


Figure 4.13: Influence of the maximum number of iterations over the point to point error and the standard deviation in a long video sequence.

rotations in R_y corresponding to $\pm 30°$.

However, we observed from simulations that the effectiveness of this kind of tracker is dependent, first on the training set used, and secondly on the mask initialization. The process of choosing the training set is laborious and was obtained empirically. What we have seen of this set is that we have to do a trade off between the span of the perturbations, because if we choose a little span, we will not be able to track video sequences containing fast movements, or sequences where

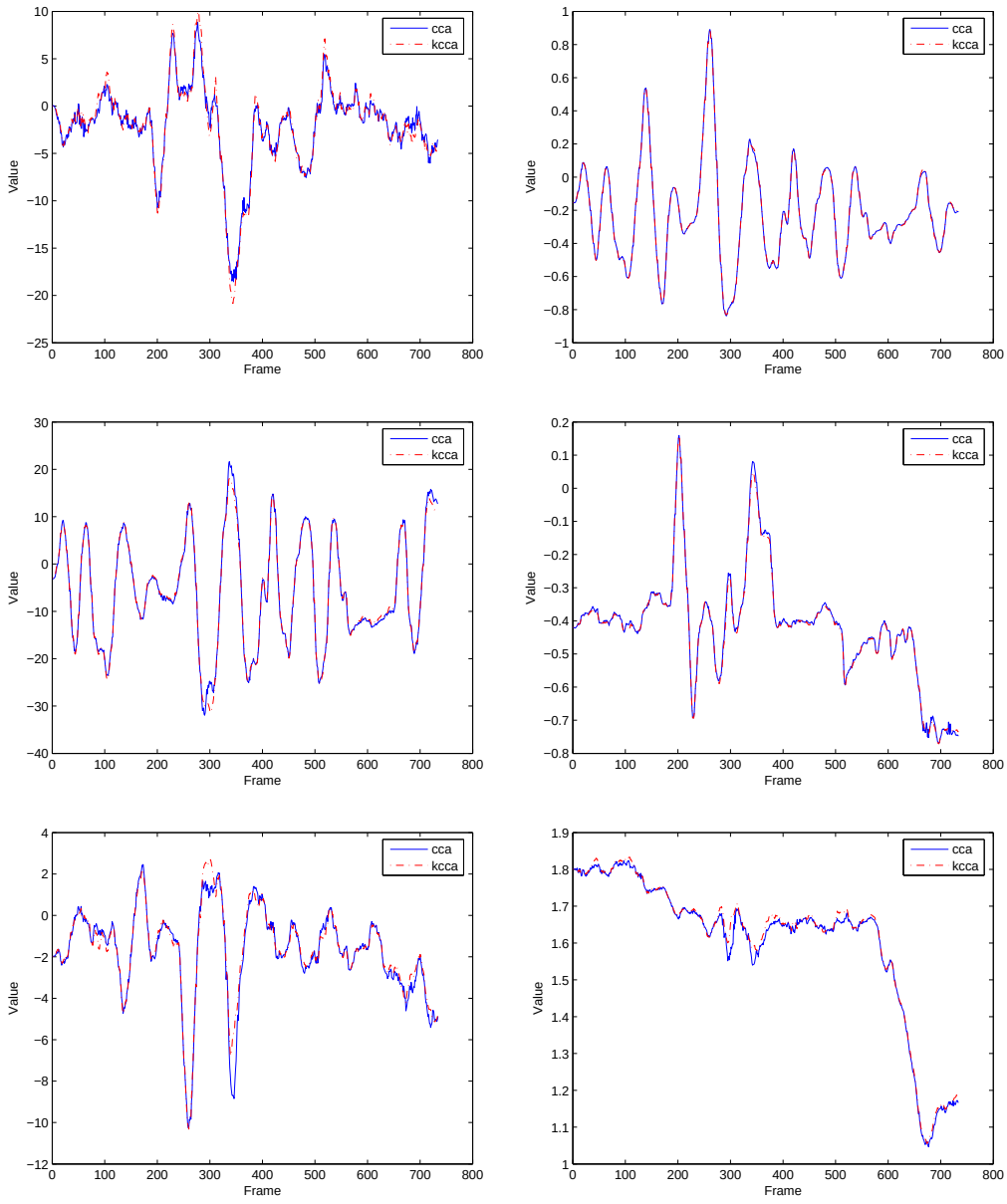


Figure 4.14: Left column: graphics showing rotations in plane x, y and z. Right column: Translations in x, y and z axes.

some frames are dropped to achieve a tracking in webcam input sequences. In the other hand, if the span is too large, the resulting algorithm will not be very robust neither for the large perturbation, nor for the small perturbations, because of the synthetic faces, that contains more distortions when the perturbations are larger. In our case we can cite as an example the rotation corresponding to the vertical axis, Ry, which was trained with perturbations going as far as 20 degrees.

For the case of the 3D mask initialization, the pose and the facial features must be as realistic as possible at the first frame, otherwise, the tracker can get lost because the model affects directly the texture extraction and consequently

the state vector predictor, especially for the out of plane rotations.

Another important choice that has to be done is the size of the stabilized face images (Figure 4.4). If we chose a patch too small, we will have a very fast training process, and a fast tracker, but we will lose detail that can give robustness to our estimation matrix. In the other hand, if we use a too big image, the training will be slower as well as the tracking, and we will also be constrained in the vector points that we can use for training purposes.

The results obtained by means of the CCA and the KCCA did not present a significant difference. However, if we consider the computation time required for the KCCA algorithm, which was more than 16 times slower than the CCA algorithm, we can conclude that for the type of data we use, it is better to use the linear approach.

In the following chapter we will show how we introduced the gesture tracking, based on the CCA approach, principally for tracking the mouth, eyes, and eyebrows.

Chapter 5

Extension to Facial Gesture Estimation in Video Sequences.

In the last chapter we have presented an algorithm that estimates a head's 3D position based on a geometric model. Although this tracker appears to be robust with respect to facial gesture, there are some applications that will require to estimate the facial gesture for more complex analysis of the face. This will also give more robustness to the tracker, because it can lose track if there is a considerable amount of facial gesture.

Some of the computer vision applications where the head pose and facial gesture estimation are crucial tasks can be video surveillance, human-computer interaction, biometrics, vehicle automation, etc. It poses a challenging problem because of the variability of facial appearance within a video sequence. This variability gets increased when we consider not only the changes in head pose (particularly out-of-plane head rotations), lighting, occlusions, but also the changes in facial expression, or a combination of all of them.

We shall show experimentally on different video sequences that, indeed, our CCA approach is well suited not only to obtain a simple and effective facial pose estimation but also the facial animation. We decided to keep the linear version of the CCA, given that the non linear method did not present a big difference in robustness, but was a lot more expensive from a computational point of view.

In this chapter we present an extension to our model-based approach that incorporates CCA for monocular 3D face pose and facial animation estimation. This approach fuses the use of a parameterized 3D geometric face model with the CCA in order to correctly track the facial gesture corresponding to the lip, eyebrow and eye movements and the 3D head pose encoded in 17 parameters.

5.1 Joint 3D pose tracking and facial gesture estimation

We have seen in the last chapter that CCA is a powerful tool well suited to regression task. It was demonstrated in [76] that using the CCA to estimate few

parameters resulted in noisy parameter estimation, because the number of *canonical variates* used in CCA is fixed by the minimum dimension of the two data sets. Having this in mind, we use the same algorithm as the described before, but in this case what we want is to estimate the complete facial 3D pose and animation state vector $\mathbf{b}$ given in (4.6), by:

$$\mathbf{b} = \left[\theta_x, \theta_y, \theta_z, t_x, t_y, t_z, {\tau_\mathbf{a}}^T\right]$$

where $\boldsymbol{\tau_a}$ is a 11 dimensional vector, containing some of the *Candide* model's animation parameters that control eyebrows', eyes' and lips' movements, that will be listed below when describing the implementation. In that case, the state vector has 17 components, $\mathbf{b} \in \mathbb{R}^{17}$.

To accomplish this, we have modified the training part, that in this case will include the animation parameters corresponding to the facial animation of the eyebrows, the eyes and the mouth. We added the resulting state vector perturbations $\Delta\mathbf{b}_i^{training}$ and the corresponding input image residuals $\mathbf{x}_i^{training} - \mathbf{x}_0^{(ref)}$ to the corresponding matrices to perform the explained CCA.

Even if this method showed to work efficiently, the fact of using a larger matrix, with all the operations relative to the algorithm, proved to become very expensive for the training process with the new dimension of the vector state. We have also seen that the 3D pose estimation was estimated less precisely than in the case where no facial gesture was estimated. In order to avoid this penalty, given that the objective of this tracker was to be light, simple, and robust, we decided to estimate separately the 3D pose and the facial gesture, as we will explain in the following section.

5.2 Independent 3D pose and facial gesture estimation.

Given that in chapter 4 we proved that our tracker performed well when we estimate the 3D pose in presence of facial animation, we decided to prove that estimating some facial gesture just after the estimation of the pose could be as robust as the joint estimation of them. We do that because the facial gesture was not very precise, especially for the eyeballs' rotation. So, to cope with this, we proceed to train two trackers, one for the pose and the facial gesture corresponding to the mouth's region, and other corresponding to the facial gesture of the eyes' region, as depicted in figure 5.1, where we can see the difference between the joint and independent pose and facial gesture estimation based in two trackers.

As expected, the results obtained showed to be more accurate for the eyes' region, so we decided to test if going to a three model tracker could further improve the performance of the facial gesture estimation. For this, we proposed to use a third tracker to estimate the facial gesture corresponding to the mouth's region. The resulting algorithm uses three different stabilized face images (SFI), as showed in figure 5.2. The assumption of independence of the upper part of the

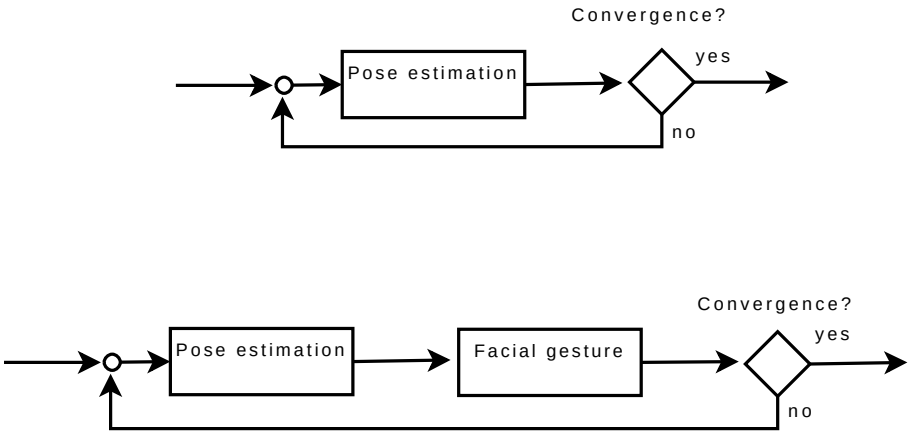


Figure 5.1: Diagrams showing the joint and independent pose and facial gesture estimation.

face with respect to the lower part proved to be a good choice, since we could use a stabilized face image for each region with more resolution that the one used for the pose estimation. This lead us to a more precise estimation of the state vector as we will show in the section 5.6.

SFI_1 SFI_2 SFI_3

Figure 5.2: Stabilized face images (SFI) used for tracking the pose: SFI_1, the eyebrows and the eyes: SFI_2, the mouth: SFI_3, respectively.

Finally, we wanted to see if we could use an approach based on local feature of the face, in order to reduce the size of the input vector used, as is explained in the following section.

5.3 Local approach

Local features, as explained in section 3.2, have been widely used to locate and track objects. The principal advantage that is reported from this kind of local approach, as in [51], where a classification of feature points is used, or in [24] where the Constrained Local Model is described, with respect to global approaches, is

that they are more robust to scale, viewpoint, illumination changes, clutter background, partial occlusions or the fact that the pose of the target object may be very different from the ones in the training set. One important point to remark is the fact that for normalization, as each point is normalized independently of each other, local methods are supposed to be more robust to important illumination variations.

In our local approach, we used some points of the *Candide* model corresponding to important face features. These features can be seen in figure 5.3, where the local stabilized face image is compared with the global stabilized face image used in the previous proposition. As we can see from the figure, in the local image we did not synthesize the profile views.

The change of this approach with respect to the previous is basically the construction of the vector $\mathbf{x}_t = \mathcal{W}(\mathbf{g}(\mathbf{b}_t), \mathbf{y}_t)$. It consists, as explained in 4.2, in taking the texture obtained by placing the 3D *Candide* model and drawing it with all the rotations and translations fixed to a predefined value, in this case, zero, except for the scale parameter in a frontal view, and all the expression parameters $\tau_{\mathbf{a}}$ set to zero. Then, around the 96 selected vertices of the *Candide* model we extract independently normalized greyscale image patches of size 6×6 pixels. This size was obtained experimentally, using the smaller window size that gives the best results. Then, we concatenate all these patches to get the observation vector $\mathbf{x}_t$ of size 3456.

(a) (b)

Figure 5.3: (a) Local stabilized face image compared to the (b) global stabilized face image used in the previous versions.

The remaining of the algorithm is the same as that explained for the jointly estimation of the pose and facial gesture in section 5.1.

5.4 Use of the mean vector.

To perform the processing related to the correlation , as the CCA, it is necessary to use centered data. In practice two possibilities exist. Either, we known the mean value, or we have to estimate it from the data. In our case, we were faced to study the two choices. The first was to utilize the first image, that we used to create the synthetic training images, to center the data matrix, assuming that this image corresponds to the true mean of the database. This assumption is based

on the fact that it was the only image where no modifications were introduced. So we use this image as a reference vector $\mathbf{x}_t^{ref}$ obtained at the first frame and updated accordingly to (4.7). The second one consisted in estimating the mean vector of the synthetic training database. This is why we have decided to estimate the mean of the training database created from a single person, and compare the results obtained with this mean estimate with respect to the results obtained with the reference vector. The mean computation is then written as:

$$\overline{\mathbf{x}}_0 = \frac{1}{N}\sum_{i=1}^{N} \mathbf{x}_i^{training} \tag{5.1}$$

and it is updated in a similar way as the reference vector:

$$\overline{\mathbf{x}}_{t+1} = \alpha\overline{\mathbf{x}}_t + (1-\alpha)\hat{\mathbf{x}}_t \tag{5.2}$$

where α is the forgetting factor. The results obtained with this approach are discussed below.

5.5 Implementation.

We have used the same equipment as explained in 4.5, and for all the implementations used in this chapter, we have retained the following eleven animation parameters for facial gesture tracking, that are also depicted in figure 5.4:

(1) upper lip raiser
(2) jaw drop
(3) mouth stretch
(4) lip corner depressor
(5) left eyebrow lowerer
(6) right eyebrow lowerer
(7) left outer eyebrow raiser
(8) right outer eyebrow raiser
(9) eyes closed
(10) left eyeball's yaw
(11) right eyeball's yaw

Based on the algorithm described in this chapter, we have implemented, for comparison purposes, four versions of the tracker combining different stabilized face images and a local approach.

- ⋆ The first version of the algorithm uses a stabilized face image (`SFI_1`, in Figure 5.2) to estimate simultaneously the 6 head pose parameters and the 11 facial animation parameters.

- ⋆ The second version uses a stabilized face image (`SFI_1`) to estimate simultaneously the head pose and the lower face animation parameters (parameters (1) to (4)) and then, starting from the previously estimated state parameters, we use a stabilized face image (`SFI_2`, in Figure 5.2) to estimate the upper face animation parameters (5) to (11).

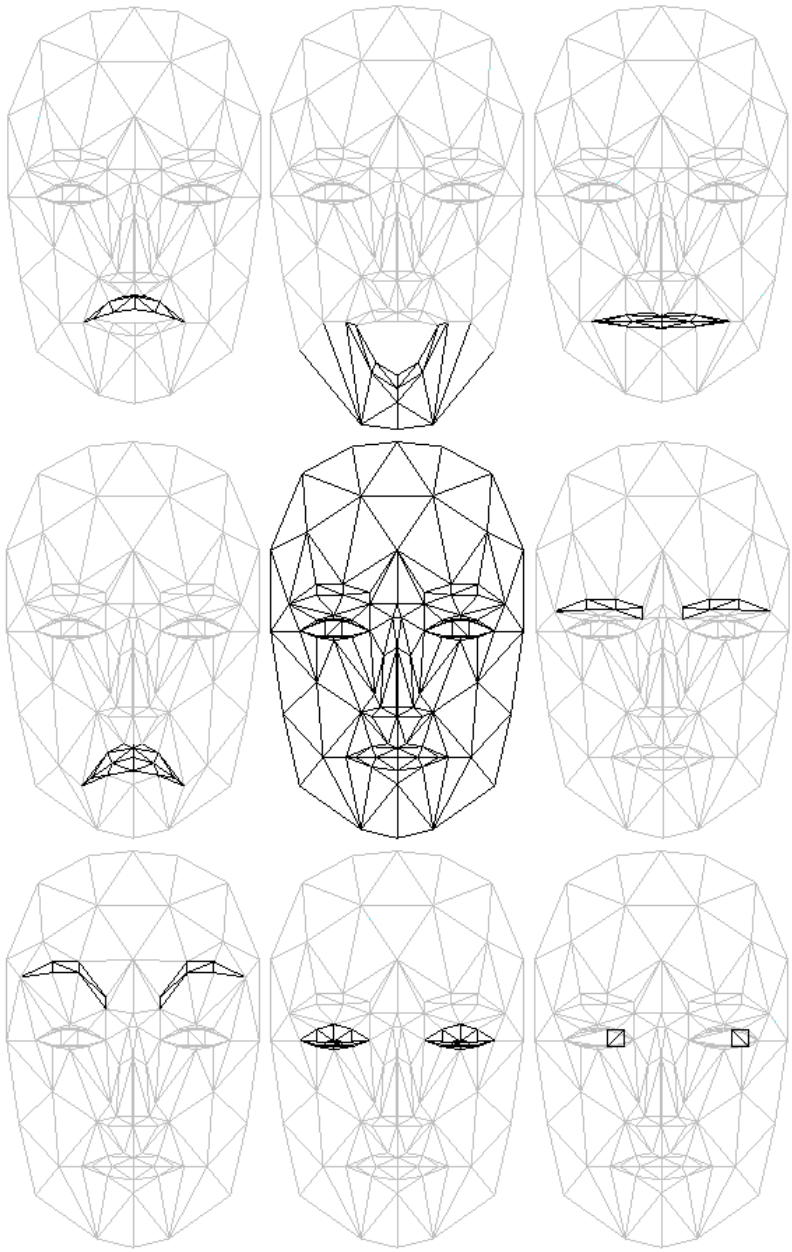


Figure 5.4: Animation units used for tracking purposes. From left to right, Top row: (1) upper lip raiser (2) jaw drop, (3) mouth stretch. Middle row:(4) lip corner depressor, standard shape of the *Candide* model, (5-6) left and right eyebrow lowerer. Bottom row:(7-8) left and right outer eyebrow raiser, (9) eyes closed,(10-11) left and right eyeball's yaw.

⋆ The third version of the tracker uses three stabilized face images sequentially: one to track the head pose (`SFI_1`), one to track the lower face animation parameters (`SFI_3`, in Figure 5.2), and a last one (`SFI_2`) to track the upper face animation parameters. `SFI_1`, `SFI_2` and `SFI_3` are respectively

composed of 96×72, 86×28, and 88×42 pixels.

- ⋆ Finally the local approach uses 96 selected vertices to extract patches of size 6×6 pixels, as shown in figure 5.3.

For training, we use 317 training state vectors with the corresponding appearance variations for the pose, 240 for the upper face region and 200 for the mouth region. The same points are used in the three implemented versions. This means, for the one model algorithm we used 757 training state vectors. For the two model algorithm we used 517 training state vectors for the mouth's facial gesture and pose estimation, and 240 training state vectors for the eye's facial gesture. Finally we used 748 training state vectors for the local approach.

These vectors correspond to variations of $\pm 20^{\circ}$ for the rotations, $\pm 10.5\%$ of the face width for translations, and animation parameter's values corresponding to valid facial expressions. We chose these points empirically, from a symmetric grid centered on the initial state vector as explained in 4.4. The sampling is dense close to the origin and coarse as it moves away from it (see Figure 4.5). Due to the high dimensionality of our state vectors, even after the separation into three models, we did not use all the combinations between the chosen points. It is important to say that we consider the lower and the upper face animation parameters as mutually independent.

Finally, to limit divergences when important out of plane rotations are present, we test the estimated space vector $\mathbf{b}_t$, and if a parameter is bigger than a threshold fixed for each parameter, we fix it to that limit and do not let it go farther. This threshold was important for the rotation parameters, especially $\mathbf{R_y}$ which was set to $\pm 60^{\circ}$.

5.6 Experimental results

For validation purposes, we use the video sequences described in section 4.5. We have structured this part in order to present the comparison of the first three algorithms that use one, two and three models respectively. Then we will compare the best approach with the local model. Afterward, we will present the use of the mean instead of the reference vector. Then, we will consider the use of the components of the most common color representations, to model the face, and finally we will talk about the CCA's coefficients.

5.6.1 3D pose tracking.

To test the 3D pose estimation we used the 45 video sequences provided by La-Cascia [48] as well as the talking face. The fact of estimating the facial gesture did not improve significantly the pose estimation. This can be seen in figure 5.5, where we compare the best tracker's results with the tracker that only estimates the pose, and we can see that the difference can be considered almost null. As a

consequence we have decided to implement separately the facial animation and the pose estimation, as it is explained below.

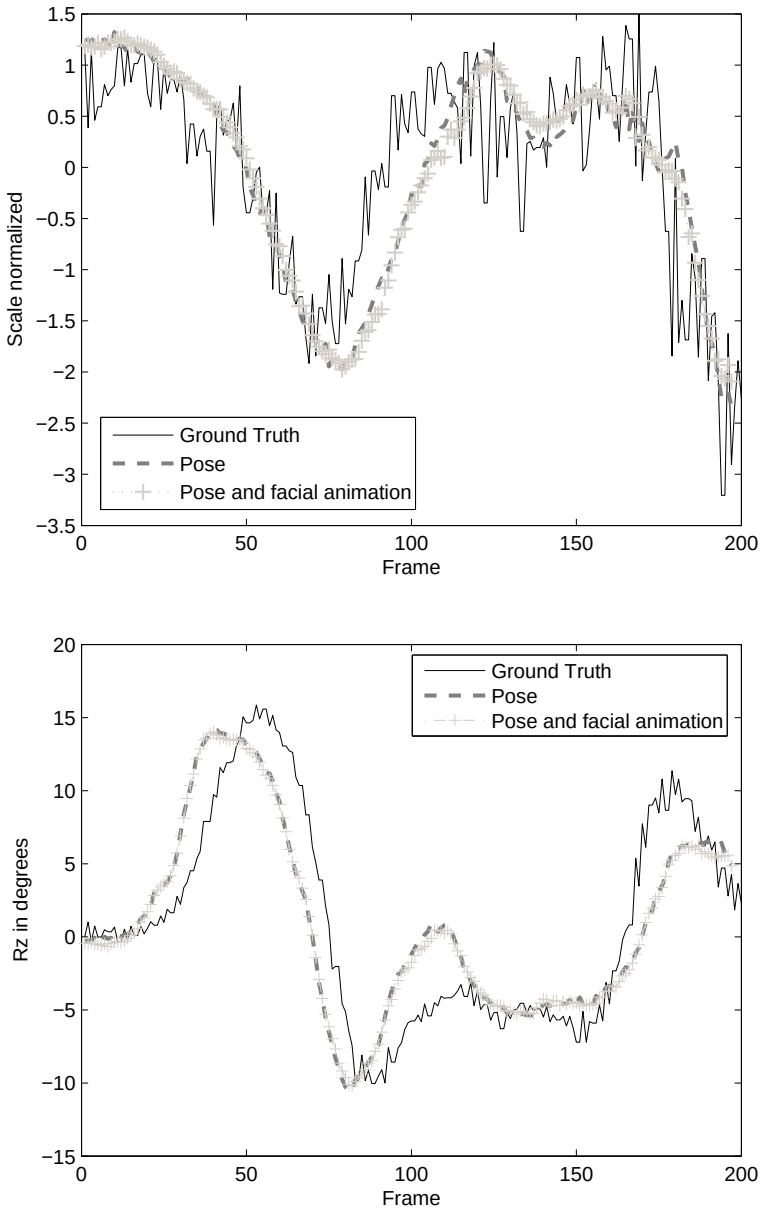


Figure 5.5: Comparison of the pose estimation when no facial gesture is estimated and when we estimate it with the three model version and both of them compared to the ground truth.

5.6.2 Simultaneous pose's and facial animation's tracking.

To test the performances of the three versions of the tracker we employ the talking face video sequence. We use the same characteristic points as those described in 4.5. In order to evaluate the behavior of our algorithm we calculate for each point the standard deviation of the distances between the ground truth and the estimated coordinates divided by the face width. Figure 5.6 depicts the standard deviation over the whole video sequence for each point using the three implementations of our algorithm. We can see that the points with the greater standard deviation correspond to those on the contour of the face. The precision of these points is strongly related to the accuracy of the estimated pose parameters. The best performance, as expected, is obtained with the third version of our algorithm based on three stabilized face images to estimate first the pose, then the lower face animation parameter and finally the upper face animation parameters. When using a single model, the tracker presents some imprecisions, not only in the facial animation parameters, but also in the pose estimation, making its performances worst than in the case when no facial animation was estimated. This is due principally to the fact that the region representing the eyes and the mouth has not an enough amount of pixels to correctly describe the movements of local features, making the pose and the facial animation parameters sensitive to each other. The second version of the algorithm (based on the two stabilized face images `SFI_1` and `SFI_2`) improves the estimation of the upper face animation parameters. Using three instead of two models brings an improvement only for the points corresponding to the mouth, but no further improvements were obtained for the pose estimation. Based on these results, we retain the third version of the tracker to explore its robustness.

In figure 5.7 we compare the evolution in time of the mean standard deviation of the 52 points used. From the image we can see that the fact of estimating the facial gesture with the three model gives a more accurate tracking.

The α parameter affects the way we update the reference stabilized face image in equation (4.7). From experiments we find that $\alpha = 0.99$ is a good choice. It is important to say that when there is no update, i.e. $\alpha = 1$, the tracker diverged in long video sequences like the talking face. We see that the mean standard deviation of the 52 facial points stays approximately constant with some peaks. These peaks correspond to important facial movements. In the case of frame 993 the rotation around the y axis corresponds to $36.62°$. In frame 1107, the rotations around on the x, y and z axes are respectively $-13.3°$, $18.9°$ and $-10.5°$. We observe on the whole video sequence that even if peak values are large, the tracker still performs correctly. Figure 5.9 shows sample frames extracted from the whole talking face video sequence and from different video sequences part of the data set in [48] and from a webcam. We can appreciate the robustness of the tracker even in the case of cluttered backgrounds.

Experiments were conducted to evaluate the sensitivity of the facial animation tracker in the case of erroneous 3D pose estimations. To do that we assumed that the 3D pose estimation was correctly performed. Then, as the facial anima-

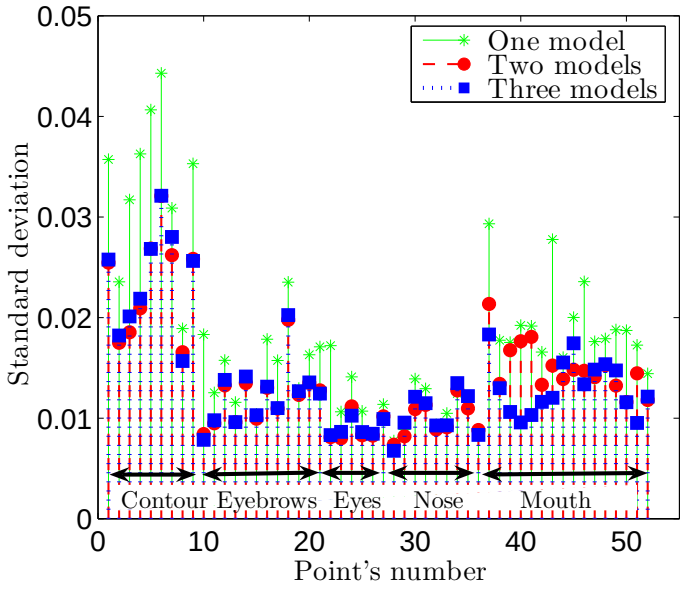


Figure 5.6: Standard deviation of the 52 facial points w.r.t. the face width, using the three versions of our algorithm, to track both the pose and facial animation parameters. It is obtained from the 5000 frames of the talking face video sequence.

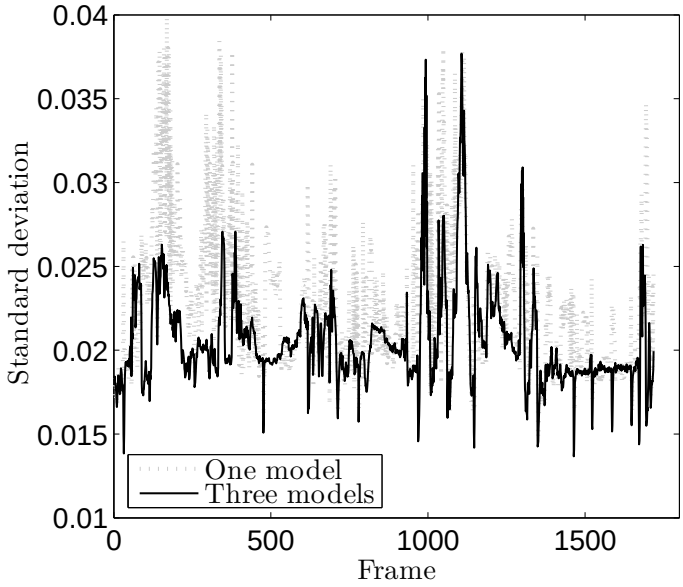


Figure 5.7: Evolution in time of the mean standard deviation of the 52 facial points for each video frame comparing the one model and three model approaches.

tion is estimated after the 3D pose, we added some Gaussian noise to the six pose parameters before estimating the facial animation, with a noise's variance into

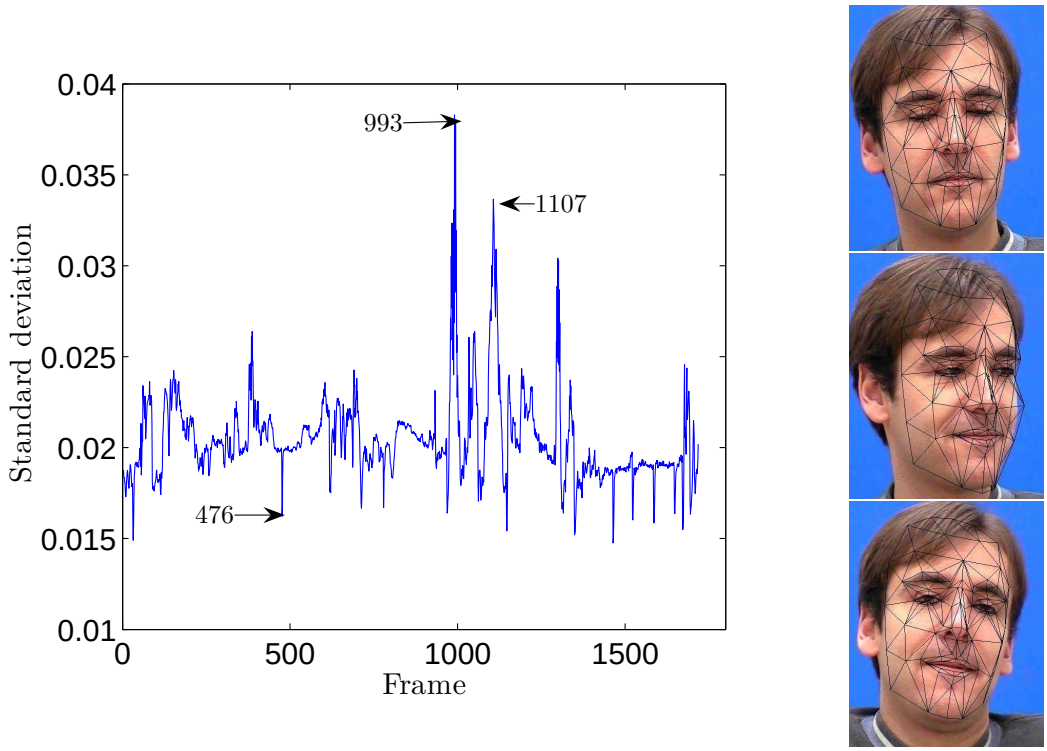


Figure 5.8: Standard deviation of all the points w.r.t. the face width for each video frame, and three frames of the sequence corresponding, to frame 476, 993, and 1107 respectively, using the CCA algorithm.

the following intervals: $\pm 10\%$ of the estimated head width added to the three translation parameters, and $\pm 3^\circ$ added to the three rotation parameters. Figure 5.10 shows the stability of the "eyebrow lowerer" animation parameter estimation even if the six pose parameters have been previously altered, as well as the perturbed rotation along the z axis.

Features like eyebrows and eye closure tend to change rapidly during emotional expressions. We show in Figure 5.11 the time evolution of the "eye closed" animation parameter, and observe on the graph that 18 eye blinks and one long period with closed eyes around frame 100 are correctly detected. This is confirmed when looking at the talking subject in natural conversation in the talking face video sequence going from frame 140 to frame 1850.

5.6.3 Local approach.

For validation purposes, we use the talking face video. As in the previous evaluations, we used the 52 points previously described to evaluate the performances of this approach. In order to evaluate the behavior of our algorithms we calculated for each point the standard deviation of the distances between the ground truth and the estimated coordinates.

We see in figure 5.12 that the mean standard deviation of the 52 facial points

Figure 5.9: Frames from different video sequences showing the pose and gesture tracking. From left to right: Talking face video sequence, two LaCascia's video sequences and two video sequences from a webcam.

stays around a constant value for both trackers with some peaks. These peaks correspond to important facial movements. In case of frame 992 the rotation around the y axis corresponds to $36.62°$. In frame 1102, the rotations around on the x, y and z axes are respectively $-13.3°$, $18.9°$ and $-10.5°$. The tracker using the local model, presents slight oscillations when visually compared to the global model tracker, that can also be see from results depicted in this figure. This is because the local model tracker is more sensitive to out-of-plane rotations and facial gestures.

Figure 5.12 also depicts the standard deviation over the whole video sequence for each point. The points with the greater standard deviation correspond to those on the outer contour of the face. The precision of these points is strongly related to the correctness of the estimated pose parameters. In this figure we can

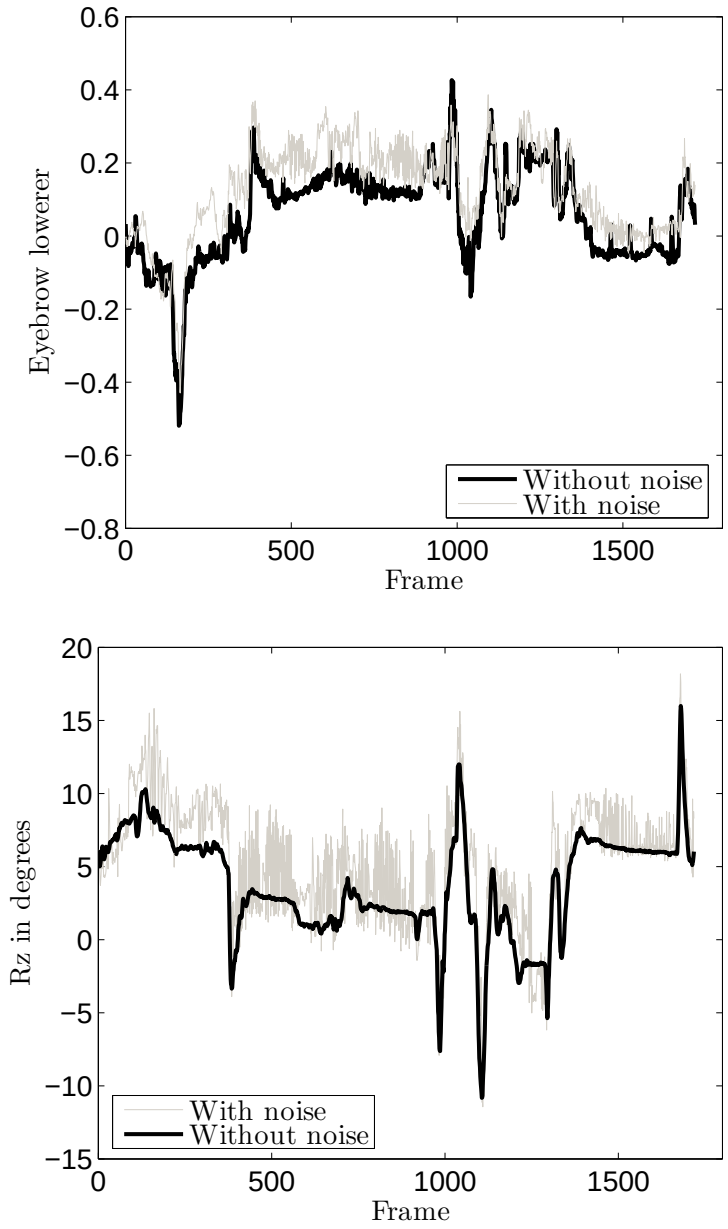


Figure 5.10: Top: Estimated eyebrow lowerer animation parameter with and without perturbations applied on the previously estimated pose parameters. Bottom: rotation in the z-plane with and without noise.

see that except for some points, the behavior of the global model tracker outperforms that of the local model one. This can be explained from the fact that the global model uses more information about the face than the local model, especially if we consider that we synthesize two profile views of the face for the global model. This makes the global model more robust to rotations, as can be seen in the frames shown in figure 5.13. The average time for pose and facial ani-

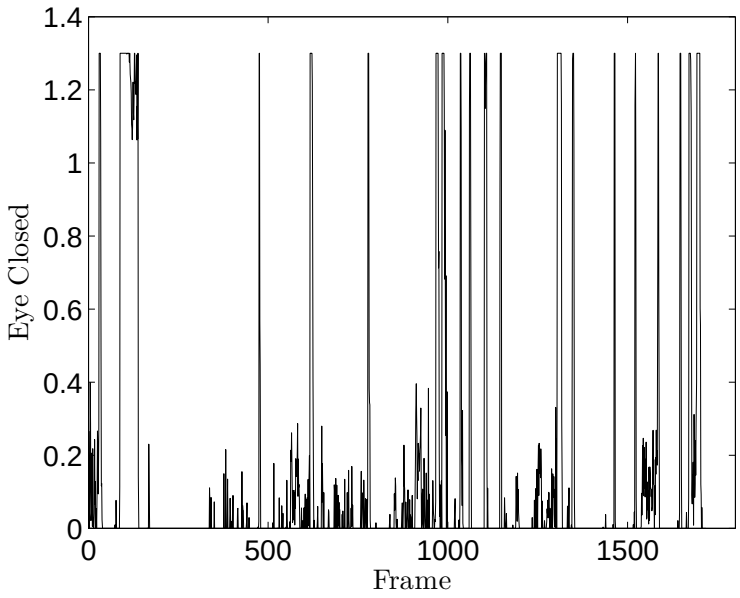


Figure 5.11: Temporal evolution of the "eye closed" animation parameter.

mation tracking is about 26 ms per frame for the local model tracker and about 46 ms per frame for the global model tracker if we exclude the time for video read, decompression and write/display operations. The average time for training is 29.1 and 33.2 seconds for the local and global model tracker respectively.

Finally, to test the robustness of the trackers to illumination changes we used the challenging 967 frame long video sequence given at that time by the Polytechnic University of Madrid [11][1]. Sample results with both trackers are shown in figure 5.13. We observe the same behaviors as before: the global appearance based tracker is more robust to significant pose variations, and the local appearance based tracker works more accurately for the facial features if the pose is correctly estimated.

5.6.4 Use of the mean vector.

Till now, we have used the reference vector obtained at the beginning of the tracking, and we updated it at each frame with the new estimated state vector. However, we wanted to compare the behavior of the algorithm when using the mean of the synthetic data with respect to the use of the reference vector to center the data. In order to do that we estimate the mean of the synthetic data during the training, and we center the data with respect to this mean. We obtain then the relation existing between this mean vector and the perturbation parameters.

We tested the behavior of the facial pose and face animation over almost the 5000 frames of the talking face video sequence. We perform this additional test to

[1] http://www.dia.fi.upm.es/∼pcr/downloads.html

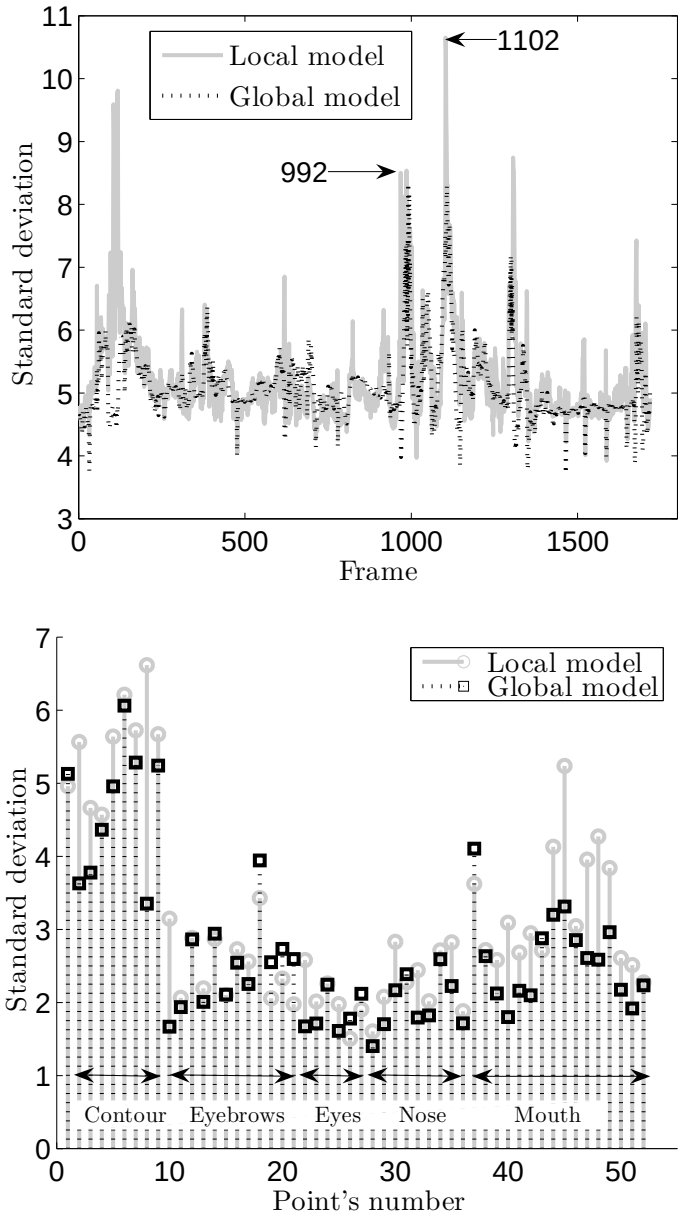


Figure 5.12: Top: Mean standard deviation evolution in time. Bottom: Standard deviation of each point.

see the stability of the pose and facial animation estimations when there was an update of the mean, and when wee keep it untouched. The results are shown in figure 5.14

We can appreciate that unlike the case when we use a reference vector, in this case the update of the mean does not improve the performance of the tracker, but it introduces perturbations and can make it diverge. This implied that if we have an application where the video conditions can be assured to be almost con-

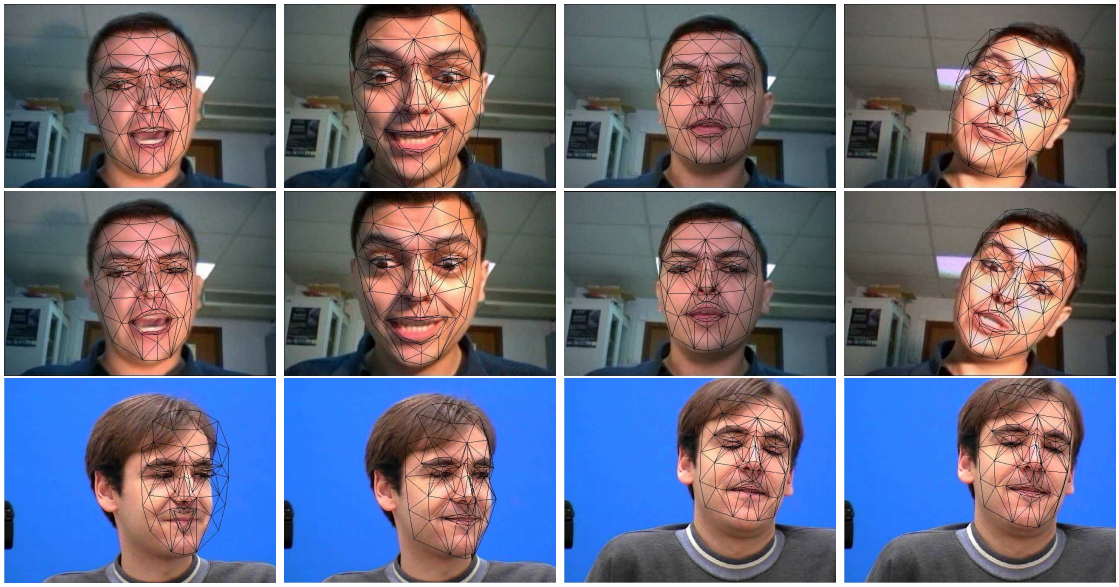


Figure 5.13: Top row: frames obtained with the local model. Middle row: frames obtained with the global model, Bottom row: frames from the talking face obtained with the local and global model alternatively.

stant or where we have a database with the examples of variations that can affect our tracker; we don't need to update the mean vector. This will increment the robustness of our tracker in case of some track lost, because the model will be kept untouched, and if a recovery algorithm is implemented, it could perform the tracking with the same parameters estimated during training.

We compared the behavior between the mean and the reference based algorithm. The results can be seen in figure 5.15, where we can see that the use of the mean performs slightly better than the reference based algorithm, but not enough to discard the reference based algorithm.

Finally, we have tested our algorithm in video sequences where the camera was moving as well as the person to be tracked. Due to the fact that we only analyse the arriving video frame in the last known position, the fact of adding the camera movement did not present a problem for our tracker.

5.6.5 Results using different color spaces

As explained in 3.2.1, there exist transformations of the color space RGB different as the grayscale transformation, as the Lab, HSV, YCrCb. In this part we present the results obtained when we use only one component in any of these different color spaces. We perform this test to verify that the grayscale conversion was well adapted to our approach. To evaluate the performance we used the 5000 frames of the talking face video sequence, given that we have the ground truth to verify the real performance in a long video sequence.

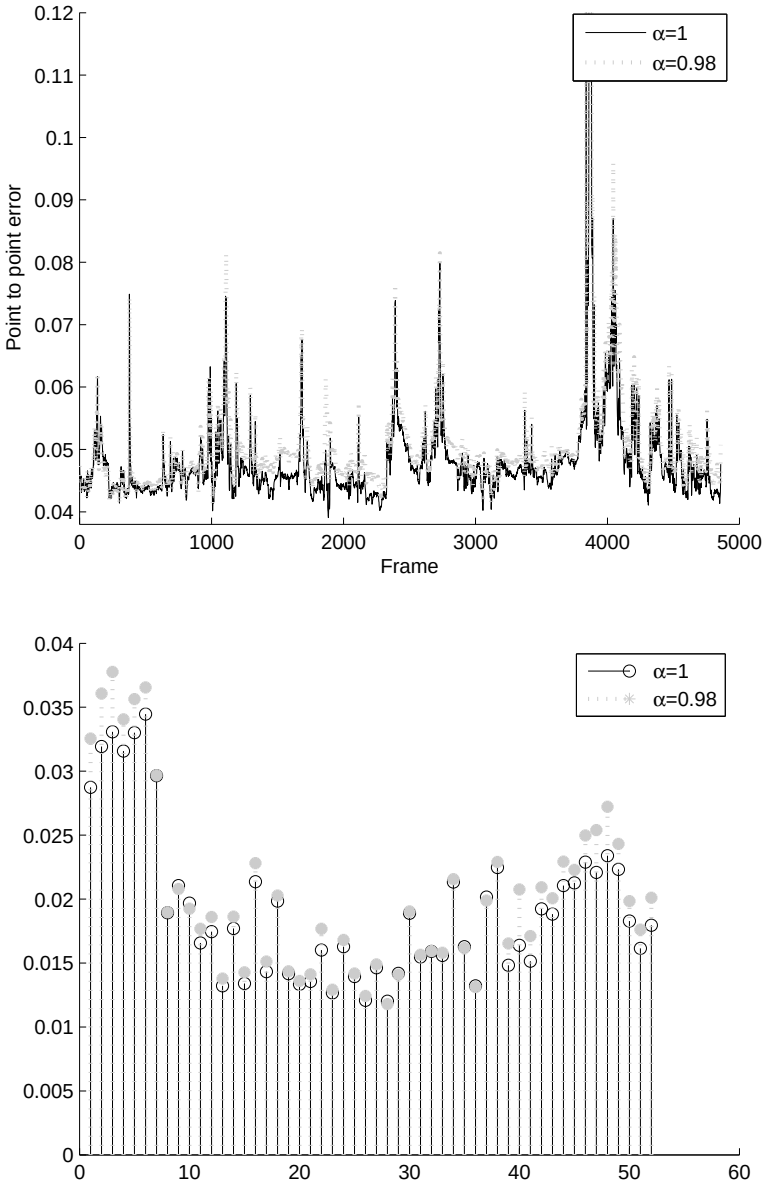


Figure 5.14: Top:Point to point mean error evolution in time when the α parameter is modified. Bottom: Mean point to point error.

YCrCb components. The first transformation that we have tested was from the RGB space to the YCrCb space. YCrCb is a transformation of the RGB space where the Y component stands for the luma, and it contains the most of the image information, that corresponds to a grayscale image. The Cr and Cb stand for the red and blue chroma components. In table 5.1 we present the mean point to point error for different face regions normalized with respect to the face width and in percentage.

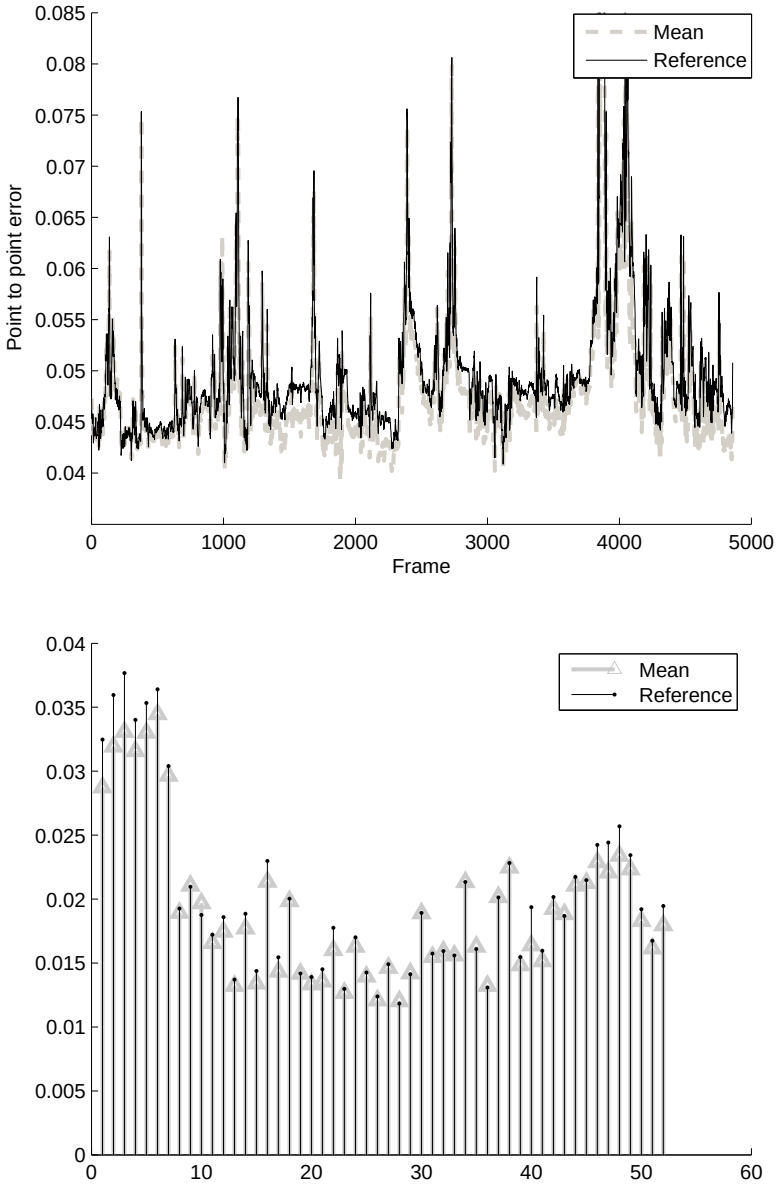


Figure 5.15: Comparison between the mean and the reference vector approaches. Top:Point to point mean error evolution in time. Bottom: Mean point to point error.

From these results we can conclude that using the other components of this space transformation does not improve the results obtained with the grayscale transformation. We can see that the results for the two chroma components have an inferior performance. This is because the most of the image information is contained in the luma component Y. The Y component is equivalent to the grayscale transformation.

	Y	Cr	Cb
General	5.03	5.18	5.18
Contour	6.34	6.56	6.86
Eyebrow	4.29	3.94	4.16
Eyes	2.31	3.44	2.83
Nose	6.02	7.25	6.68
Mouth	5.30	4.83	5.06

Table 5.1: Experimental using the YCrCb different color channels. The Y component is equivalent to the grayscale transformation.

RGB channels. In this part we have used a single channel of the RGB image to perform the tracking instead of the grayscale transformation.

	R	G	B
General	5.32	5.03	4.98
Contour	7.26	6.56	6.14
Eyebrow	4.49	4.38	3.96
Eyes	2.53	2.31	2.45
Nose	6.20	6.03	6.34
Mouth	5.42	5.10	5.28

Table 5.2: Experimental using the three different color channels.

It can bee seen in the table 5.2 that the use of each channel independently improves the results obtained, particularly in the case of the green channel for the mouth, or the blue channel in general. It is important to say that in the RGB representation of an image there is a high correlation between the different components, what makes the images stored in this format to have a big size.

HSV components. We have also used the HSV transformation. HSV (hue, saturation, value) is another space created to be perceptually uniform. We have observed that the H component leads to higher dispersions than the grayscale representation. In the other hand our algorithm does not work properly with the S component. We have found no explanation for this behavior.

However, the results for the V component are better to those obtained with the grayscale transformation, as can be seen in the table 5.3, particularly for the contour tracking.

Lab components. Finally we used the Lab transformation. Lab is a space created to be perceptually uniform. From results displayed in table 5.4, we can see

	H	S	V
General	6.83	105.4	4.98
Contour	9.64	102.5	6.10
Eyebrow	6.47	111.1	4.07
Eyes	3.07	107.3	2.35
Nose	6.90	104.4	6.22
Mouth	6.89	102.6	5.33

Table 5.3: Experimental using the three different HSV space's components.

that the a component presents a performance that is a little better that the one presented by the grayscale transformation for the mouth points. However, for the b component the algorithm does not work whenever there is a high head rotation in the video sequence.

	L	a	b
General	5.04	5.58	77.6
Contour	6.39	7.46	67.7
Eyebrow	4.35	5.64	128.8
Eyes	2.34	3.38	112.4
Nose	6.03	6.24	72.2
Mouth	5.25	4.92	34.8

Table 5.4: Experimental using the three different Lab space's components.

Mixture of color transformations. As we have seen, from tables 5.3 and 5.1, the grayscale transformation and the V component of the HSV transformation presented the best performances for the pose tracking, specially if we see the contour points. For the eyes region also these two transformations performed better. However, for the mouth region the a component of the Lab transformation and the Green component of the RGB representation presented the best performances, as can bee seen in tables 5.4 and 5.2. From these results we decided to test the fact of using one space color component for the mouse region and other for the eyes and the 3D pose.

After using the four possible combinations the best performance is obtained with that using the V component of the HSV transformation for the pose tracking, and the a component of the Lab transformation for the mouth facial gesture tracking.

In table 5.5 we can see the comparison of the grayscale tracker against the proposed algorithm. We can see that the performance for the mouth tracking is better, and as there are more points for the mouth than for the other parts of

the face, this reduces the global error. However, for the contour tracking the performance of the grayscale tracker is better.

	Grayscale	Combination
General	5.03	4.69
Contour	6.34	6.47
Eyebrow	4.29	4
Eyes	2.31	2.47
Nose	6.02	6.48
Mouth	5.30	4.03

Table 5.5: Experimental using the YCrCb different color channels.

This results can also be seen in figure 5.16, where the evolution in time for the error and the standard deviation are depicted. We can see that the general behavior of this algorithm outperforms slightly the performance of the grayscale algorithm.

In figure 5.17 we can see the mean error and the standard deviation for each point. We can see that the points corresponding to the mouth region present a better performance, but some of the nose points and the contour points present a worst behavior.

If we see the tracking results in the video sequence, the differences are slight and both algorithms can be considered to work properly.

5.6.6 The CCA coefficients obtained

As we have explained, the CCA is a very powerful tool that obtains the relationship between two data sets. It is able to obtain a set of vectors that will relate, the variation of an input vector, in our case the stabilized face images, and the parameters that produce this kind of variations. In figure 5.18, we show the coefficient for the pose and the facial gesture estimation. We can see from them that some features, as the nose, the mouth or the eyes, are those that will determine more strongly the movement that has been detected, in the case of translation and rotation estimation. For the eye's region, we can clearly see the parameters that correspond to the eyebrows movements and the eye's movements, and finally for the mouth's region we can see the different parameters modifying the mouth's shape.

It is interesting to see that the parameters corresponds to the face model, not as in the case in [59] where the eigenfaces, obtained with a PCA approach, are not related to the human face.

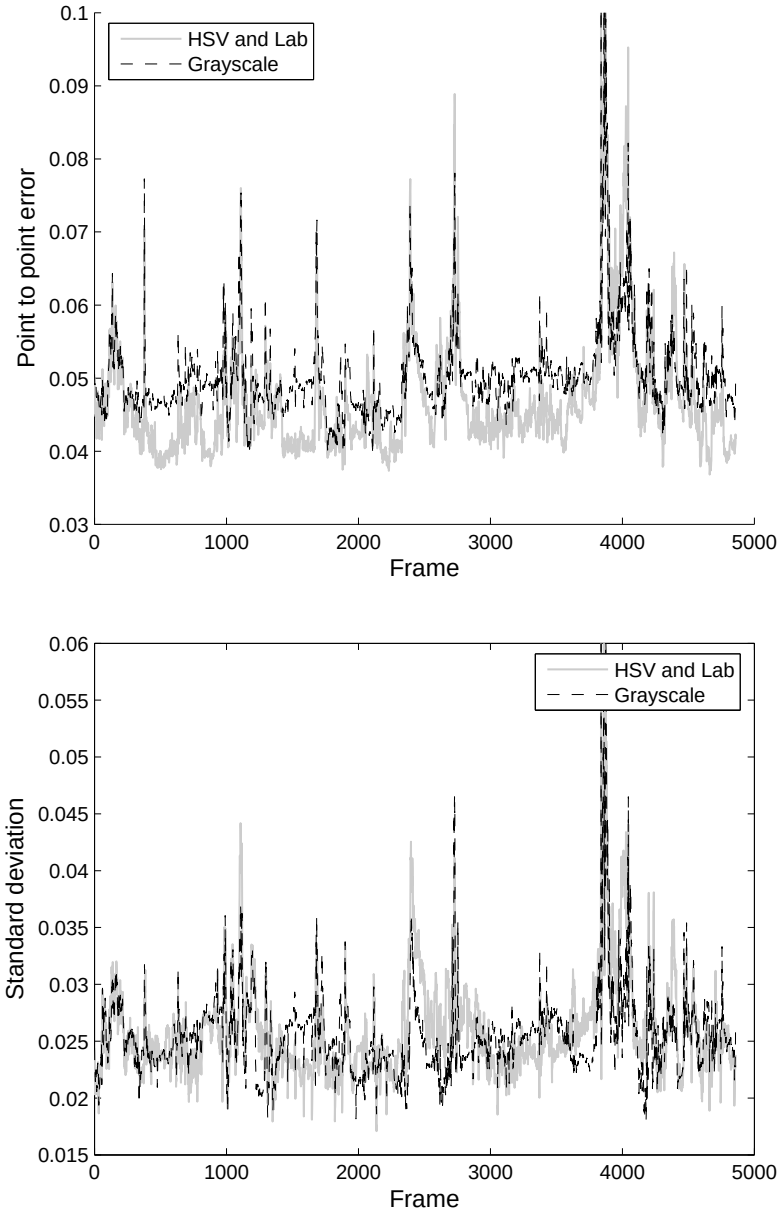


Figure 5.16: Top: Evolution in time of the mean error for the algorithm using the grayscale for the pose estimation and the L component of the Lab color space for the mouth and eye parameters. Bottom: corresponding standard deviation.

5.7 Conclusion

We have presented a method that is capable of tracking both 3D pose and facial animation parameters from individuals in monocular video sequences. The tracking approach is simple from the training and tracking points of view, robust even to slight illumination changes and precise when the out-of-plane face rota-

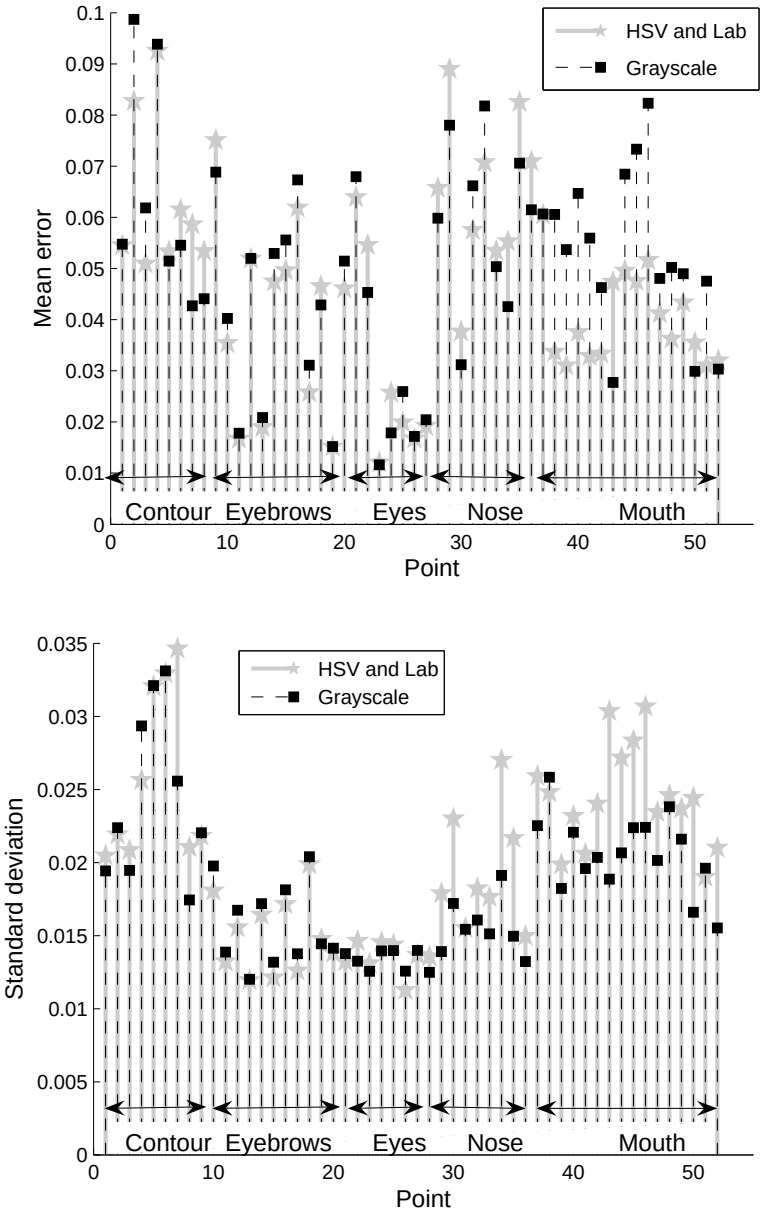


Figure 5.17: Top: Mean error for each point of the algorithm using the grayscale for the pose estimation and the L component of the Lab color space for the mouth and eye parameters. Bottom: corresponding standard deviation.

tion angles stay in the interval $\pm 30°$, as it has been proved with the simulation results.

We have presented several tests to verify the robustness and behaviour of our algorithm under different situations to track both 3D pose and facial animation parameters of individuals in monocular video sequences. The first thing we have seen was that estimating separately the pose and the facial gesture did not worsen

the global tracking result. This presents two principal advantages: Firstly we have three smaller training sets to estimate the CCA coefficients instead of only one huge set, which represents a gain of computation time during the training phase, and secondly, we have different resolutions to analyze the local movements corresponding to facial gesture. The advantage of this approach is that the tracking results for the gesture animation can be improved greatly because we can use a bigger resolution to determine the correctness of the tracking for the mouth and eyes parameters.

From the comparison with respect to the local model we can conclude that using a local model presents the advantage of being faster and has good performances, nevertheless, this approach is more sensitive to strong out-of-plane rotations and important facial gesture. Another advantage of using the local approach is that it also presents a robust behaviour when faced to important illumination changes, especially because each local feature is normalized independently of the others features. However, we can conclude that the global approach represents a better solution for real world conditions, where important rotations can appear and facial gestures are expected. To make more robust the local approach, we should implement an approach similar to those explained before in 5.3, particularly in [53; 54] where the local features are treated independently one from each other, and then a geometric model is used to find the best configuration of the model that contains the most of the feature points detected, making this approach robust to outliers.

The use of the mean or of the reference vector in the algorithm will depend on whether or not we want a model that can be updated. The principal advantage of the mean-based algorithm is that we make all the computations once and our model can be stored, to track the same person under the same conditions without changing or re-estimating these parameters. However, if we want to use our algorithm in an application that must be initialized each time because the external factors change widely, we will need to train each time our algorithm and depending on the extrinsic characteristics of the environment we will chose to use the reference base or the mean base algorithm. If we know that the video conditions will not make our algorithm to diverge, then both algorithms are well suited. However, if we know that our algorithm can diverges or have some bad estimation, and then it would be better to use the mean based algorithm, because these bad estimations will undermine the performance of the reference based algorithm because of the update step. Another advantage of using the mean based vector, as will be explained in chapter 6, is that we can build a general model based on several training images that are not related with the first frame of a video sequence.

When analyzing the results from different color space's components, we can conclude that the fact of only applying a color space transformation to the RGB image and working with one of the component of the new space, does not gives a better result than the one obtained if we work in the grayscale space, or not for all the parameters. We have seen that using a mixture of the V component of the HSV transformation for the pose and eyes region, and a component of the

Lab transformation for the mouth's region resulted in a slight amelioration of the tracker performances, but due to the complexity that it adds to the algorithm, we can consider that using this mixture of spaces is not worth.

Finally, it is important to say that there exists another kind of processing to use the three components of an RGB image in order to highlight some specific characteristics of the video images, as the belonging of a pixel to a skin region or a lip region as in [34] and [56], or some feature detector, as contours or interest points. However, as the principal objective of our algorithm is to be kept light and fast, we keep the grayscale transformation without adding an additional step to the image process.

In this chapter we have then introduced an algorithm which is able to follow the face's 3D pose and facial gesture using as a starting point the first frame that is initialized by hand. From this initial frame we create three models with the aid of the CCA, which probed to cope with long video sequences.

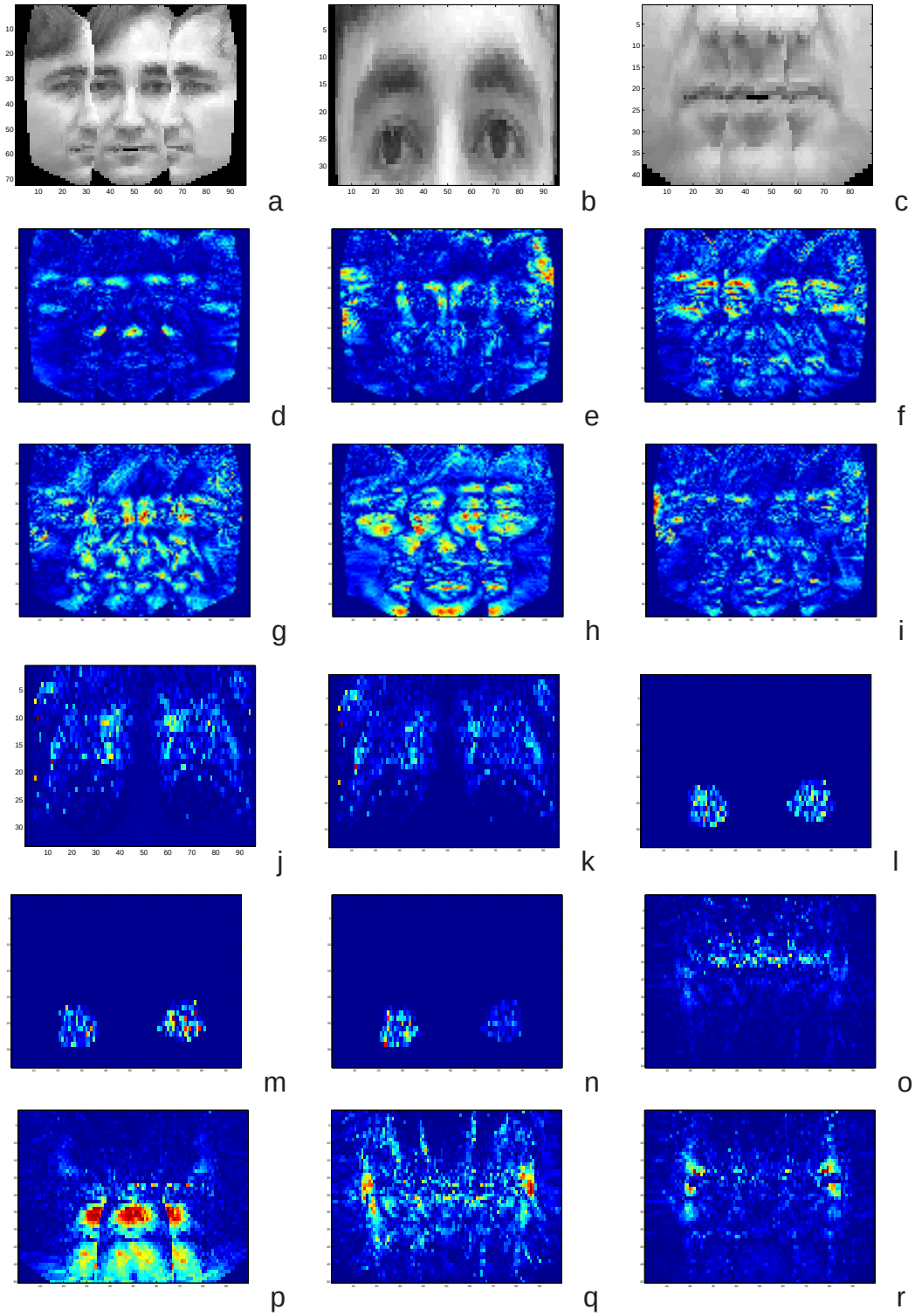


Figure 5.18: Stabilized face image corresponding to the (a) 3D pose (b) Eye's region facial gesture and (c) Mouth's region facial gesture. (d-f) Coefficients for the rotation, (g-i) Coefficients for the translation. (j-k) Coefficient for the eyebrows' parameters, (l-n) Coefficient the eyes' parameters, (o-r) Coefficients for mouth's parameter.

Chapter 6

3D pose and shape estimation in images of unknown faces.

While there are multiple effective approaches for tracking, they all require an initial pose estimate, as stated in chapter 4, which remains a difficult task to provide automatically, fast and reliably. The processes of initialization and tracking are complementary, since all trackers require initialization and, even those with better performance, will lose track as a result, for example of severe occlusions or severe illumination changes. To cope with this problem, several methods to detect objects exist, like point detection, background subtraction, segmentation, supervised learning, etc. In our case we are interested in supervised learning applied to object detection.

Supervised learning consists in having a set of learning images containing different views of the selected object features to be learned, some object class that are manually given to each learning image, and a mechanism to learn the relation existing between these object features and the object class. Basically it consists in creating a function that maps inputs to desired outputs [100]. This mapping can be of two forms. If the output parameters can take continuous values we will talk about a regression. In the other hand, if we have an output in the form of a class label, then we will talk about a classification.

Some examples of methods using the classification approach to detect objects in images can be found in [15; 54; 67; 86]. In all these approaches, the features obtained are classified as belonging or not to a certain geometric model. In the case of [15], they also use the *Candide* model. The use of a model gives a robustness with respect to outliers. In [54] a feature detection algorithm is used to track rigid objects in video sequences, using the classification of the feature points as specific points of the interest object, taking each video frame as an independent image, and [67] is an extension to track non-rigid objects.

Maybe one of the most popular approaches, presented in chapter 3, is the one described in [86], where the Adaboost algorithm is used to detect faces, but can be extended to any object. This work is used in [75] to train a contour tracker, or in [77] to track faces in indoor environments combining a color approach to track the whole human body. In [62] the Adaboost is trained to detect hockey players

and the tracking is performed by means of a particle filter to keep track of the multiple objects.

From these works we have seen the advantage of the classification methods to estimate an initial position. However, to tune this initial estimation to a 3D geometric model, a regression method appears to be more adequate. This is why in this chapter we will propose an algorithm to estimate the 3D pose and the face's shape in images based in a first classification approach and then in a regression approach. This algorithm uses the Adaboost algorithm [86] to detect a face in frontal view. Then based on this first location, we adapt the 3D *Candide* model to obtain the correct 3D pose of the face. Then we proceed to adapt the shape of the face based on the shape units of the *Candide* model. The algorithm used is based on the CCA algorithm that we have already explained, but the principal difference is the training of our algorithm, which is explained in the following section. The structure of this chapter is the following. First, we will present the principal changes of the training process, showing a different normalization used to create an expression and shape free patch. Then we will describe our algorithm for images. Finally we will present the implementation, the results and the conclusions.

6.1 Training process based on multiple images.

As explained in chapter 4, the training process consists in creating a matrix that explains the perturbations of the state vector, 3D parameters for the pose and shape parameters in this case, in terms of the residual of the difference between an image vector and a reference or mean vector. In order to obtain this matrix we need to create two databases. The first containing the faces with multiple variations in pose and the second containing the pose variations applied to each face. We need to highlight that we have proceeded as for the facial gesture tracking algorithm, i.e. we have separated the face's shape from the 3D pose estimation as three separated models. First we estimate the pose, and then we estimate the shape parameters for the eyes-eyebrow region and then the parameters for the mouth region.

The main difference in this algorithm with respect to the previous one is that instead of using the first frame of a video sequence we use a database of several people [80], and we create in a similar way as before, multiple images containing variations of the pose around the original 3D pose. This is because our algorithm is intended to work on images, or as an initialization/recovery process in video sequences.

To cope with this challenging problem the first thing we have to do is to make a normalization of the faces. In our case we performed a geometrical normalization as described in [3] and later in [31]. The reason for this geometrical normalization is that face features of different people are not placed in the same place. This means that the shape of people's faces varies from one person to another. For instance, if we see the eyes of several people in a database, we will discover that

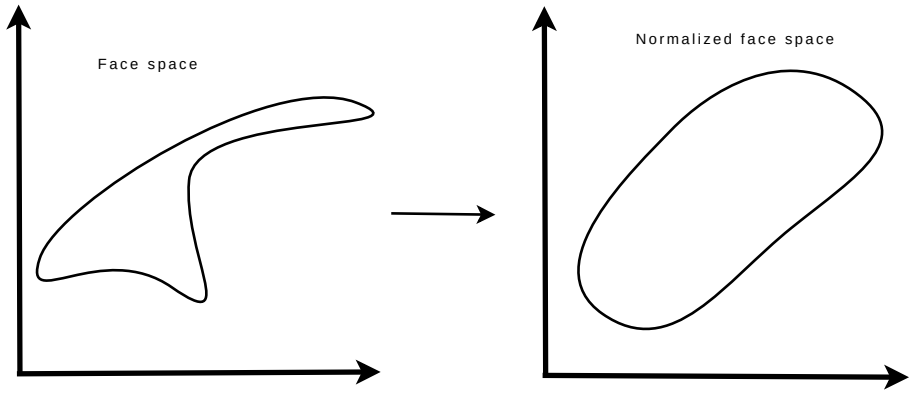


Figure 6.1: Representation of the geometrical normalization of the face space.

the eyes have different sizes, different separation between them, different alignment, and different vertical position with respect to the face. In order to cope with this variability, we can use the geometric model to produce a normalized face. To proceed like this we use a transformation of the texture obtained at each frame. It consists of taking the texture obtained by placing the 3D *Candide* model, as described before, and then, drawing it as if it was at a distant point (as in the frontal view of figure 4.4) with all the rotations fixed to a predefined value, in this case, zero for a frontal view. In this case, we will not include the two synthesized profile views. Then we will set all the expression parameters $\tau_{\mathbf{a}}$ and shape parameters $\tau_{\mathbf{s}}$ to zero, in order to find all the face features at the same place in all the resulting face patches. Then we load a fixed size window containing this draw and transform it to a grayscale image, obtaining finally an expression-and-shape-free patch of size 58×72 pixels. In figure 6.2 we show the difference between two expression-free patches and two expression-and-shape-free patches. We can see that some distortions are introduced due to this normalization of the shape, but without this normalization the parameters obtained would not be general enough.

Figure 6.2: Expression-free and Expression-and-Shape-free patches.

From this figure we can see that in general the distortions correspond to the vertical displacement of the eyes, eyebrows and mouth. The advantage of doing this transformation is that independently of all the different faces that we have used from the database, we can find all the features at the same locations, i.e., the eyebrows, the nose, the eyes and the mouth are always at the same position,

which is a requirement to have a general model able to detect people that are not in the database.

Once we have performed the normalization, we proceed with the construction of the synthesized images in order to create two databases for each model, containing the possible perturbations and the synthesized images, similarly as described in the previous chapters, but there is a difference in the grid chosen to create the synthesized faces. In this case, we have used a grid with fewer points, but also we have introduced a random component to each parameter. This was done to have different perturbations for each individual, and to have a more dense global grid. The random noise used was Gaussian with zero mean, and with a particular variance for each component, that was about the magnitude of the smallest perturbation used in the training grid for each component.

After building the six matrices, we center them, to then obtain, by means of the CCA, a $\mathbf{G}$ matrix for each model, as described in chapter 4. It is important to say that in this case we used the mean vector $\overline{\mathbf{x}}$ to center the data, as described in chapter 5, when we described the use of the mean vector instead of a reference. This mean is necessary in this case, because we are working with faces of multiple people, and in order to have a general algorithm we need to center the data with respect to this mean face. The resulting mean face vector, as well as the corresponding mean of the eyes' region and mouth's region are depicted in figure 6.3.

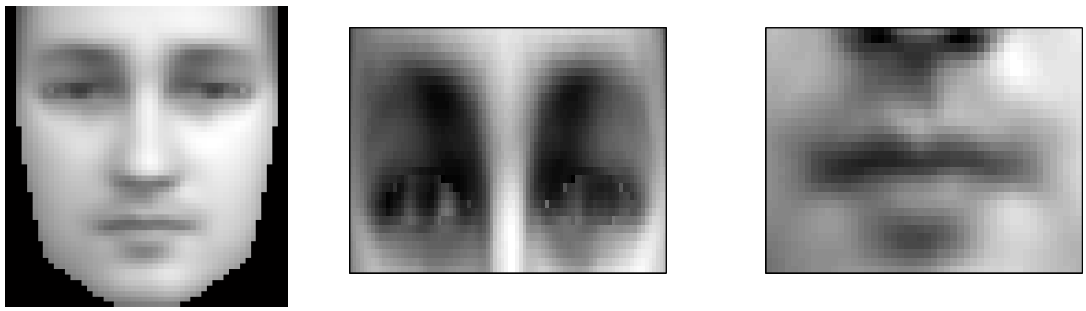


Figure 6.3: Mean face obtained from several people.

With these mean vectors and with the $\mathbf{G}$ matrices containing the CCA coefficients for each model we can then proceed to the 3D pose and shape estimation.

6.2 3D pose and shape estimation algorithm.

This algorithm, although similar in construction to the algorithm described in chapter 4, presents the difference of estimating the pose and the shape parameters of the *Candide* model, for faces that are not present in the training set of images.

In order to do that, we used first the Adaboost algorithm to detect a face in the image. This algorithm gives us a rectangular window, defined by two points with coordinates (x_1, y_1) and (x_2, y_2) respectively, where there is a face in frontal view. We proceed then to interpret this window parameters to 3D parameters by

means of a linear transformation, in order to approximate the position of the *Candide* model to be over the detected face. As the Adaboost algorithm is intended to detect frontal view faces, we fixed all the rotation parameters to zero and the parameters to be obtained from this window are those corresponding to the t_x, and t_y displacements and to the scale,t_z. This linear model is obtained by moving the *Candide* model in the 3D space of our program, and by doing a linear regression between the pixel coordinates and the 3D pose parameters, as expressed in the following equation:

$$[t_x, t_y, t_z]^T = \mathbf{A}\,[x_1, y_1, x_2, y_2]^T \tag{6.1}$$

The $\mathbf{A}$ matrix finds the relation between the pixel coordinates corresponding to the vertical axis an the scale factor t_z, and then it finds the relation between the horizontal components in the pixel domain and the t_x and t_y, taking into account the scale factor. Once we have estimated the initial position and scale of the *Candide* model, we proceed to iterate with the CCA algorithm. In this case the state vector for the pose estimation is of dimension seven, being it:

$$\mathbf{b}^{(pose)} = [\theta_x, \theta_y, \theta_z, t_x, t_y, t_z, h_x] \tag{6.2}$$

where h_x denotes the horizontal scaling of the geometric model, because we have seen that between different people the face shape varies considerably, so we kept the vertical and deep scale factors h_y and h_z untouched, and we added the horizontal factor as a parameter of the state vector.

In the case of the shape parameters, the state vector is separated in two, the part corresponding to the eyes' shape $\mathbf{b}^{(eyes)}$ containing four parameters, and the part corresponding to the mouth's shape $\mathbf{b}^{(mouth)}$ containing two parameters.

The CCA algorithm, as was described before, consists in using the matrix $\mathbf{G}$, obtained during the training, to estimate the perturbation of the state vector $\Delta\mathbf{b}_i$, where the index i stands for the iteration of the CCA algorithm, and the update equation is written as in chapter 4

$$\hat{\mathbf{b}}_i = \mathbf{b}_{i-1} + \mathbf{G}(\mathbf{x}_i - \overline{\mathbf{x}}) \tag{6.3}$$

This equation is valid for the three models, denoting $\mathbf{G}_1$, $\mathbf{G}_2$, and $\mathbf{G}_3$ the pose, the eye shape parameters and the mouth shape parameters respectively.

At each iteration the algorithm updates the state vector until it converges to the 3D pose estimation, and then it updates the shape parameters. To declare the apparent convergence we see the variation of the error at each iteration. When there is not an improvement of the error we can consider that we arrived to the desired position. To determine this improvement of the error we stored the previous and the current error, and when the difference is negative, we assume the convergence of the algorithm. However, experimentally we have seen that it was not enough to this criterion to arrive to the desired solution, so, we let our algorithm reach five times the criterion of having the current error bigger than the one estimated in the previous iteration, for the pose estimation, and two for the shape parameters estimation, to assure that the solution was the correct one. Once we

have estimated the 3D pose, we proceed to estimate the shape parameters. This is because we need to know the final 3D pose to correctly estimate the shape parameters.

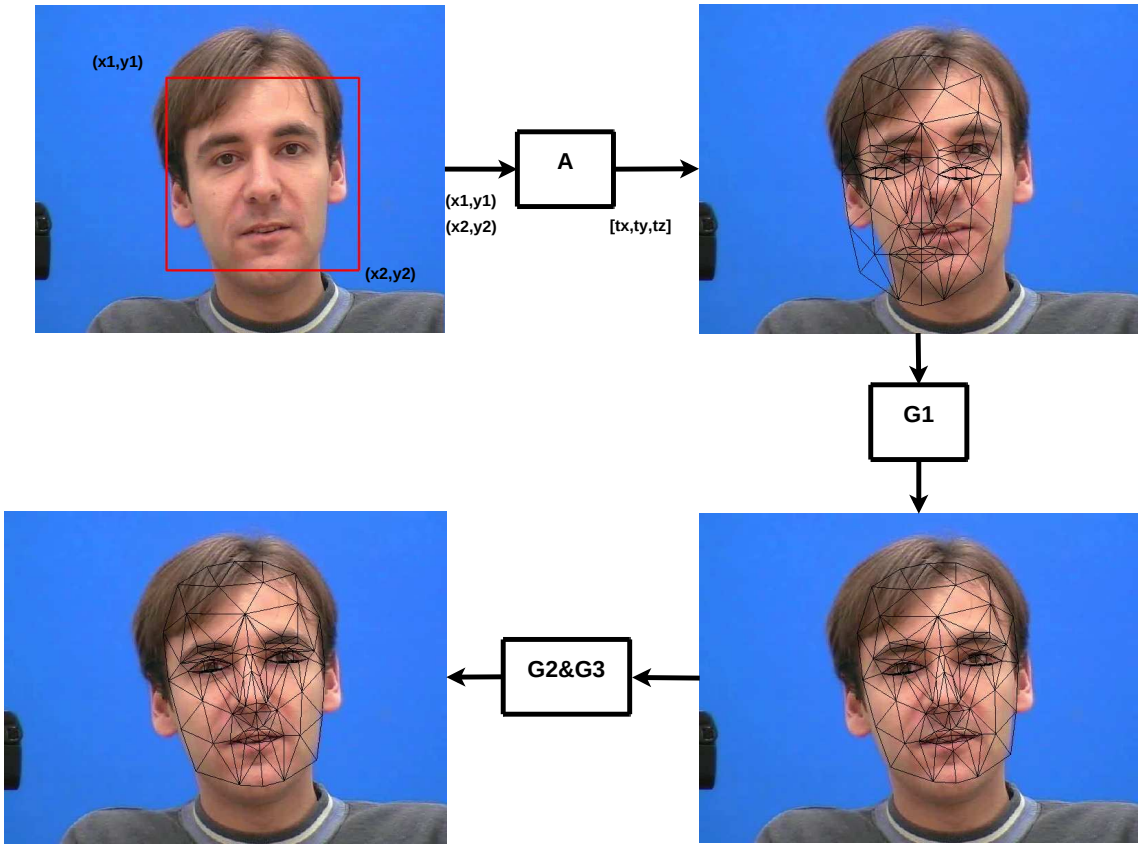


Figure 6.4: Diagram of the detection algorithm. Top left: Window detected with the Adaboost algorithm. Top right: The window coordinates are transformed to *Candide* model translation parameters by means of the $\mathbf{A}$ matrix. Bottom right: The pose is estimated with the $\mathbf{G}_1$ matrix. Bottom left: The shape parameters are estimated with the matrices $\mathbf{G}_2$ and $\mathbf{G}_3$.

The process of estimating the pose and the shape parameters is explained graphically in figure 6.4, where we show the window resulting from the Adaboost algorithm, then the resulting pose estimated by the linear transformation, the convergence of the *Candide* model to the correct 3D pose of the face, and finally the estimation of the shape parameters.

6.3 Implementation.

We have used the same equipment as explained in section 4.5. The pose parameters that were estimated consisted, as previously described, of the six 3D parameters containing the rotations and translation that will be applied to the *Candide*

model, and of the horizontal scaling of the geometric face model. The shape parameters estimated for the eyes and the mouth were:

(1a) eyebrows' vertical position
(2a) eyes' vertical position
(3a) eyes' separation distance
(4a) eyes' vertical difference
(1b) mouth vertical position
(2b) mouth width

In this case, for training purposes we use the database containing 37 different people described in [80]. In figure 6.5 we show some of the 37 different photos used to create the database. For each person we have adapted the *Candide* model manually, so we could then create synthesized images that were close to the real face.

Figure 6.5: Example of faces used for training.

We used only the frontal neutral faces, where no expression and no light variations were taken into account. For each person we have 319 different points for the pose training, 134 for the eyes region and 111 for the mouth region. These points have been chosen from a random non-symmetric grid around the origin, but in this case, as the Adaboost algorithm is intended for frontal view faces, we limited the span of the training points described in previous chapters. In this case the maximum rotation, for the y-axis was of $10°$, and of $5°$ for the other two parameters. We have tested two versions of our algorithm. The first one without estimating the shape parameter, and the second estimating them.

The Adaboost algorithm used is distributed with the OpenCV library [1], and it is based on the work of [55], that is an extension of the work presented in [86]. To train the classifier 5000 positive frontal face patterns and 3000 negative patterns have been used.

It is important to say that in order to make the matrix $\mathbf{A}$ in equation (6.1) independent to different input images sizes, we have performed a normalization of the input image in order to have the values of the detected face window in the interval $[0, 1]$ for the vertical and horizontal component. In this way, the linear model obtained to transform the Adaboost coordinates to OpenGL coordinates is independent of the size of the input image. This linear model was obtained, as previously described, by looking for the relation that exists between the pixel coordinates of the *Candide* model's center and the variations of the OpenGL pa-

[1] $http://www.intel.com/technology/computing/opencv/index.htm$

rameters. Then some little adjustments have been performed because the reference point of the *Candide* model it is not exactly at the center of it.

Due to the computing time of the algorithm of more than 48 hours, we have stored in a file the mean vectors after they were calculated off-line, as well as the three $\mathbf{G}$ matrices containing the corresponding CCA coefficients, to be used for the pose and shape estimation of the first frame in video sequences or for still images.

Another important factor for the implementation is the number of iterations performed before assuming convergence or well, until a certain number of iterations is performed. This number, in our case, was fixed to 30. This is done to prevent divergence problems when no face is detected due to an occlusion or to a false face detection of the Adaboost algorithm. We have seen that for the talking face video sequence the mean number of iterations was 18, and for the LaCascia video sequences it was between 11 and 15 depending on the person.

To probe the robustness and behavior of this algorithm we have used the same video sequences described before, but we consider each frame independently of the precedent, i.e. we consider each video frame as a standing alone image and we set to zero all the parameters when a new frame is loaded. Then we apply our method to detect the 3D pose of the face and the shape parameters as it is shown in the following section.

6.4 Results.

The first test we have performed, uses the video sequences of LaCascia, [48], where the ground truth is provided. From these sequences we have seen that the algorithm is robust to estimate faces when close to the frontal view. However, when there is a rotation it can not follow it beyond the $6°$ used for the training. This can be easily explained if we compare the input patch used for training and pose estimation, where we use only the frontal view and not the two synthesized views as in the tracking implementation. Moreover, the training uses less points, especially in the regions far from the origin, making this tracker less robust to strong face rotations. In this case we have done a trade off between the amount of different people used and the robustness to rotations and movements. As our intention was to implement an algorithm able to initialize and to recover in the presence of a frontal view, we have decided that it would be more important to have more people variance than robustness to rotation. It is also important to say that the Adaboost algorithm that gives the first estimation of the face pose is also constructed to localize frontal view faces, so it was incoherent to try to go beyond the Adaboost capabilities. This is why when there is not an estimation given by the Adaboost algorithm, we can not obtain results with our algorithm. For this special case we put the *Candide* model in the center of the frame. This can be seen in figure 6.6, where the dark regions correspond to the frames where no face was detected by the Adaboost algorithm. When the face is close to the center the algorithm estimates correctly the pose, but when the face is far enough of the

face, the algorithm diverges.

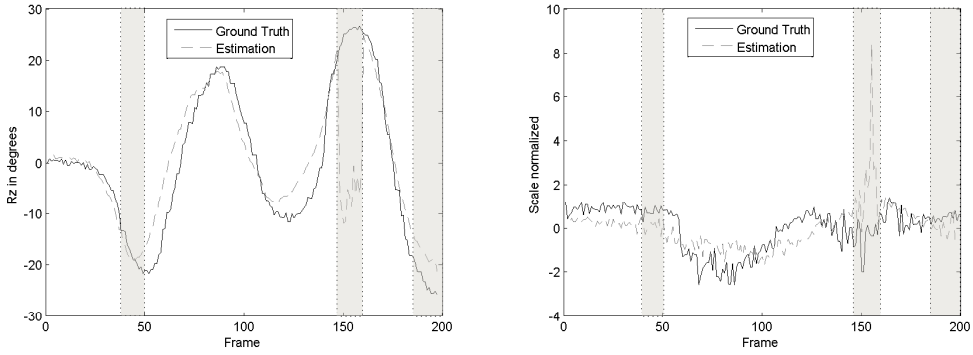


Figure 6.6: Results from a LaCascia's video. The dark regions show the frames where the Adaboost algorithm did not found a face.

In figure 6.7 we can see some of the parameters estimated from the LaCascia video sequences compared to the ground truth provided. We can see that some parameters are very robustly estimated, as the horizontal translation, while some others, as the rotations in general, are more sensible to the correct estimation of the initial window as well as the characteristics of the person appearance. If we look carefully at the rotation parameters, we can see that the rotation parameter R_y can not follow movements beyond the $\pm 5°$, corresponding to the values used for training. When this happens, the rotation parameter R_z is wrongly estimated, even if the ground true values are into the training dataset. Let us recall the differences existing between the ground truth and the data obtained with the *Candide* model, as it is explained in chapter 4.

From these experiments we have seen that the fact of adding the random part to the training grid improved the performance of the pose estimation, i.e., the fact of using different training points for each person introduces a more robust pose estimation as we have more possible perturbations in the database. The fact of having different data for two different parameters was already addressed in [11], where the authors create images containing variations of the facial gesture, and for only one facial gesture they introduced the variation of illumination. They have proved that this approach produced almost the same results that creating a huge database containing all the possible combinations of facial gesture and illumination variation. In our case we use different faces for the training and for each face we use different training points.

To see the correctness of the pose and shape estimation we have used the full talking face video sequence, consisting of 5000 frames. We have used this video sequence to first estimate the 3D pose only and then the pose and the shape parameters. These experiments have also been used to determine the number of iterations explained for the convergence criterion. It can be seen in figure 6.8 that the performances of the pose and shape estimation were better when the number of times that we could have a negative difference between the current error and the error in the precedent iteration was bigger. From experiments we

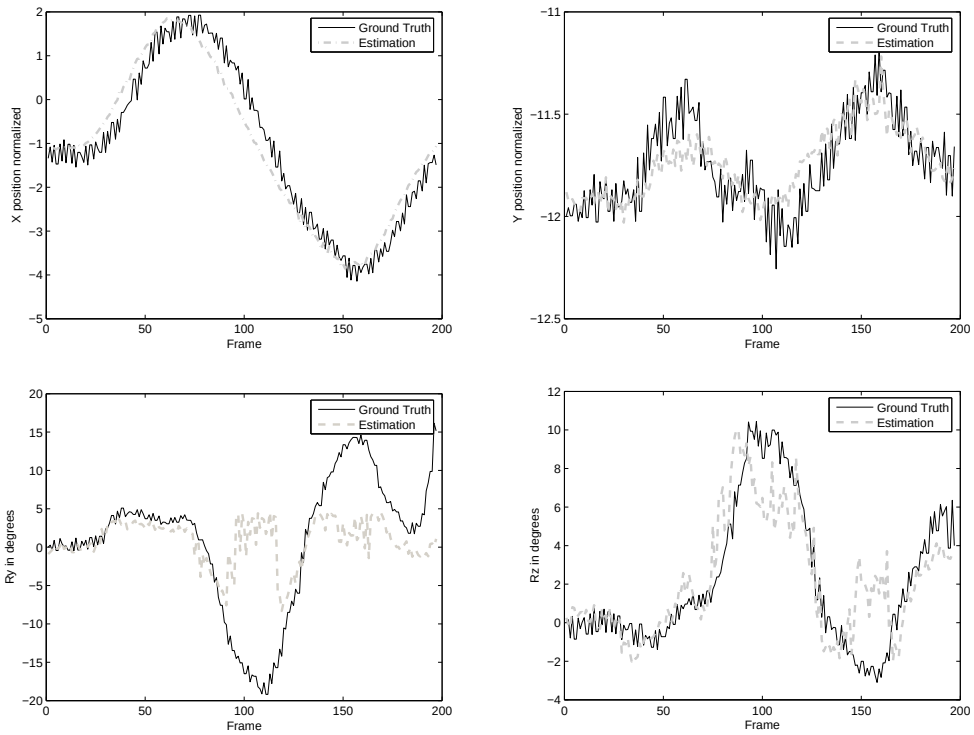


Figure 6.7: Results obtained from some LaCascia videos showing the ground truth against the estimated pose of each frame.

have fixed this number to be five for the pose estimation and two for the shape parameters. We can see in the figure that the pose estimation is more accurate and robust in the case where the criterion was fixed to five than when it was fixed to three. In the case of fixing the parameter to three, we can see from the figure that sometimes we arrived to the correct solution, and sometimes the algorithm declared convergence before arriving to the correct solution.

From figure 6.9 we can see that estimating also the shape reduced the mean error for some points of the eyes. In the case of the contour points and the nose points the error was almost the same. In the case of the mouth and the eyebrows, the error was slightly bigger, but this is due to the fact that in the video sequence the person is engaged in a conversation, producing facial gesture that is not intended to be estimated in this approach.

When we analyze the evolution in time of the mean point to point error, we can see that the behavior is almost the same for both cases, the pose and the shape estimation, or the pose-only estimation. This is depicted in figure 6.10. We can see however that there are some peaks that correspond to important facial gesture. This facial gesture produced some error estimation, especially for the mouth's vertical position estimation, mainly because the training phase for this parameter considers only the case where the mouth is closed.

Finally we wanted to compare the performances of this approach in a tracking context. To do that, we have compared the results of the tracking algorithm that

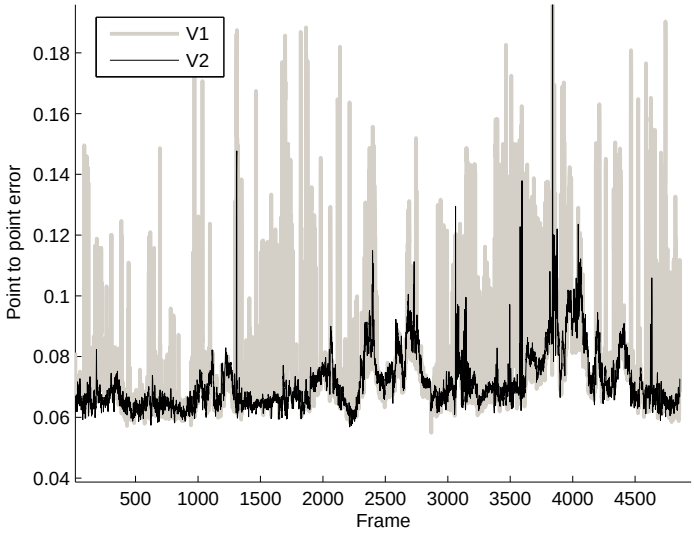


Figure 6.8: Results obtained from the talking face video. V1 stands for the iteration criterion fixed to three, and V2 for the iteration criterion fixed to five.

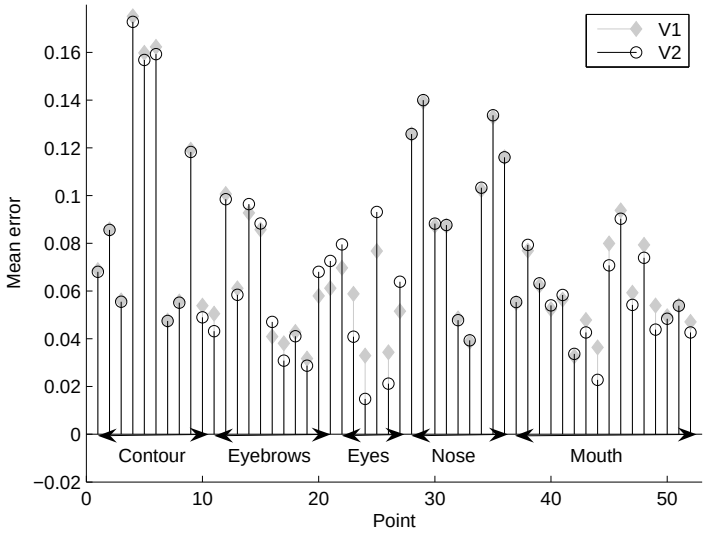


Figure 6.9: Comparison of the mean error for each point, between the pose-only and the pose-and-shape estimation at each frame. V1 stands for the pose and shape estimation and V2 stands for the only pose estimation.

uses the first image for the training process with the results of the pose and shape estimator presented in this chapter.

The results are shown in figure 6.11. We can see from this figure that the global mean error is more important than in the case of the tracking performed with the model obtained from the first frame that uses the two synthesized profile views.

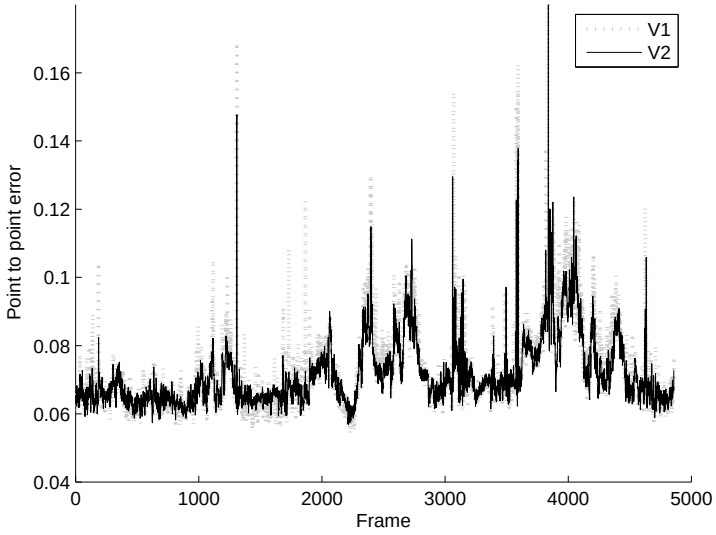


Figure 6.10: Comparison of the mean point to point error between the pose-only and the pose-and-shape estimation at each frame. V1 stands for the pose and shape estimation and V2 stands for the only pose estimation.

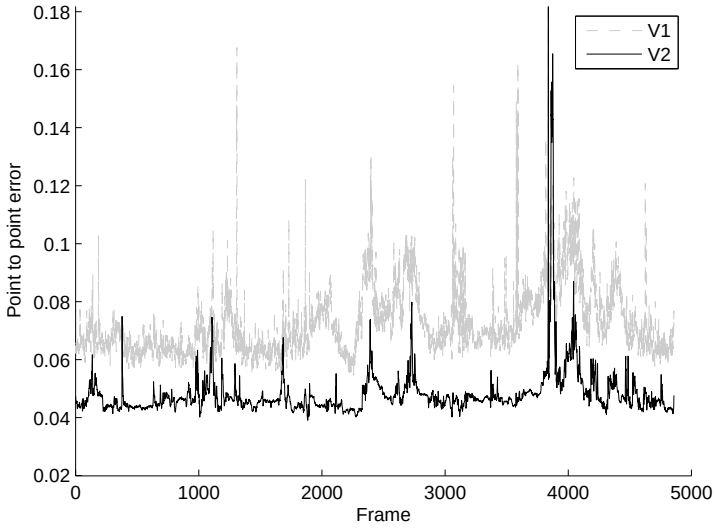


Figure 6.11: Comparison between the normal tracking V1, and the pose and shape estimation at each frame V2.

This is the result of using a general model that estimates only the pose and the shape and does not takes into account the variations of the facial gesture. We also see that there are some parts of the graphic where the pose estimation is not optimal, giving as a result some peaks in the graphic. This peaks corresponds to wrong estimations produced by extreme face rotations, or to some errors due to facial gesture that produced a wrong estimation of the pose. The off-set that

we can appreciate between the two curves is also due to the initialization that we perform in the tracking contexts, where we initialize carefully all the shape and facial animation parameters, and also some points of the *Candide* model. For the case of tracking unknown faces we use the standard model without this fine initialization. However, we can see that this offset remains practically constant for all the images of the video sequences, what confirms the robustness and correctness of the pose and shape estimation.

6.5 Conclusions.

We have presented an algorithm able to automatically estimate the 3D pose and shape of faces in images, where the face is close to a frontal view. The principal characteristics of this approach are that it is simple, fast and robust. This algorithm uses two databases containing synthesized images created from several people and another with the pose and shape perturbations applied to each face. From them we calculate the coefficients that give the relation between the residual of a mean face vector and the current observed patch. These coefficients are estimated once and stored in a file with the mean face vector. This approach uses the Adaboost algorithm present in the OpenCV library.

We have seen from experiments that using different random points for each people improved the performance of the training, improving the tracker robustness. For most of the tested video sequences the algorithm presented a good performance. However, it is important to say that we have seen that for some tested video sequences containing a particular person, the algorithm presented some difficulties to estimate correctly the pose and the shape, arriving to suboptimal solutions. This was the case for a person who was very different from the persons used for the training. To overcome this, an extended database should be created, containing more subjects from different origins, because the database used contains Caucasian people only.

This algorithm can be used as a robust method to automatically initialize a tracker that uses the *Candide* model, or to recover after lose of tracking due to severe occlusions. In the case of the recovery after track losing, we have two options. The first is to use the algorithm described here, as an initialization problem, this means, use the CCA coefficients estimated from multiple persons and the Adaboost algorithm. The second is to only use the Adaboost algorithm, and the linear model that converts the pixel window to OpenGL 3D position parameters, as explained in this chapter and keep the CCA coefficients obtained from the first image training, explained in chapters 4 and 5. This means that we will keep the CCA coefficients obtained for a particular person only when the person disappears in the current frame as a result of an occlusion, in a region of the frame where he can not get out of the scene. However, if the person is close to the borders or to a region where the person can disappear, for instance a door, the first option seems to be a better choice.

As it has been said at the beginning of this chapter, object detection algorithm

can also be used to track objects. We can also use the approach presented in this chapter as a tracking algorithm, estimating the pose at each frame independently of the previous seen results. However, using this algorithm for tracking purposes presents some disadvantages with respect to the algorithm presented in previous chapters. The most important is that it was not implemented to estimate the facial gesture, which leads to bad shape estimation. Moreover, we do not take into account the previous results, which makes this algorithm to need more iterations to arrive to convergence, 18 for the talking face for this approach while only 5 for the tracking algorithm. Additionally, the span of this algorithm is narrower, because we use a face vector without the synthesized profile views, and because we do not add the same perturbations as in the case of the tracker of chapters 4 and 5. This limits the range of this algorithm as a tracker to be very close to a frontal view. As we have seen, when we go further in rotations as for example in the R_y rotation (see figure 6.7), the algorithm may go in the opposite sense of the real variation.

We can then conclude that this algorithm is a simple and robust 3D pose and facial shape estimator, whose principal application is to initialize our tracker and it can also be used to recover when tracking is lost.

Chapter 7

Estimation with an Incremental Training Algorithm.

The appearance of a target object may change drastically due to intrinsic and extrinsic factors. Then, we have to adapt the appearance model on-line, while tracking, to reflect these changes. This is true for most of real life tracking situations, where extrinsic factors as changes in illuminations or occlusions due to the configuration of the video scene are present. This problem has been an important issue in recent years[10; 25; 31; 57; 72], especially because of the computing power of modern computers that let the implementation of real time application in the computer vision domain.

In order to cope with these possible variations, people had performed incremental algorithms, that learn on-line how to track the object. We can consider two approaches to these algorithms. In one hand the algorithms that at each time learn the new model based on the last observation, as in [25; 31], where a gradient matrix variations of a face appearance with respect to 3D pose changes is recalculated at each iteration, but the appearance is only updated. In the other hand, we can cite the algorithms that uses an update of the tracking coefficients keeping the knowledge already obtained previously, as described in [10; 57; 72].

In [10], the author proposes a method to update the SVD of a matrix from arriving data. The keypoint of this update consists in obtaining a decomposition of the new data such that there is an orthogonal matrix in this decomposition. To do that the author proposes a QR decomposition. Once we obtain this matrix, we use its properties to update the SVD matrices. However this approach did not take into account the fact that the data could be not centered. In [57], the authors take into account the fact that the data can be non centered, and introduce a SVD update that also considers the mean update applied to a visual tracker.

In [72] the authors use an incremental principal component analysis algorithm that also takes into account the variation of the mean. For this, the authors project the new data matrix over the eigenspace already obtained, and they obtain a residual from this projection. This residual is then used to update the eigenspace giving a faster algorithm. Indeed, creating a new matrix containing the eigenvectors and the new data, and then performing the orthogonalization of it is slower

than using only the reduced size new data residual, and keeping the previous eigenvectors, which are already orthogonal. These two last approaches are used in visual trackers and use particle filter to estimate the motion parameters, instead of the gradient method that often gets stuck in a local minima.

In this chapter we will focus in these approaches that used the prior knowledge and the current observation to update the tracking criteria, by means of an incremental update. In order to present this approach, this chapter is structured as follows. First we will present the incremental update of the CCA coefficients. For that, we will introduce an incremental SVD algorithm and the way we apply it for the CCA. Then we will propose an approximation used to make this algorithm faster. Finally we will present the results obtained.

7.1 Incremental update of the CCA coefficients.

As we have said, the fact of updating the coefficients of a tracker algorithm has been recently used as a method to make more robust the tracking process with respect to important changes of the object appearance. These changes are produced by extrinsic factors, such as illumination changes, occlusions, and for the case of face tracking and biometry in very long time periods, aging and changes in the person appearance.

The principal idea behind the incremental CCA consists in adding the faces that have been estimated during the tracking process. This means that when a new frame arrives, we will store the patch obtained from the last known position in a new matrix $\mathbf{A}_1^{(new)}$ of dimension $d \times n$, as well as the perturbation estimated from this patch $\mathbf{A}_2^{(new)}$ of dimension $p \times n$, and when we arrive to a certain number of new observations we introduce this set to the original database. This can be formulated as:

$$\mathbf{A}_1^{(total)} = \left[\begin{array}{c} \mathbf{A}_1^{(old)} \mathbf{A}_1^{(new)} \end{array} \right] \tag{7.1}$$

$$\mathbf{A}_2^{(total)} = \left[\begin{array}{c} \mathbf{A}_2^{(old)} \mathbf{A}_2^{(new)} \end{array} \right] \tag{7.2}$$

where $\mathbf{A}_1^{(old)}$ is the original data matrix of dimension $d \times m$ containing the training faces and $\mathbf{A}_2^{(old)}$ is the original data matrix of dimension $p \times m$ containing the perturbations. The new matrices become then $\mathbf{A}_1^{(total)}$ of dimension $d \times (m+n)$ and $\mathbf{A}_2^{(total)}$ of dimension $p \times (m+n)$.

There are two approaches that we have used to study the advantages of this approach. The first one consists in directly using the resulting matrices from equations (7.1) and (7.2), that we call direct approach, and the second approach, that consists in using a sequential singular value decomposition, in order to update more efficiently the CCA coeffients. These methods are next explained .

7.1.1 Direct update of the CCA coefficients.

The departing point is the CCA algorithm described in chapter 4, where we create the training matrices $\mathbf{A}_1^{(old)}$ and $\mathbf{A}_2^{(old)}$. With these matrices we estimate the CCA coefficients. To update these CCA coefficients, we store the matrices $\mathbf{A}_1^{(old)}$ and $\mathbf{A}_2^{(old)}$ and every n frames, we introduced the new data stored in matrices $\mathbf{A}_1^{(new)}$ and $\mathbf{A}_2^{(new)}$ that we introduce into the matrices $\mathbf{A}_1^{(total)}$ and $\mathbf{A}_2^{(total)}$ accordingly to equations (7.1) and (7.1). With these new matrices we estimate again the CCA coefficients, but every time we make this update the dimension of the matrices is increased of n elements.

This approach, evidently, is not the optimal from a computing time of view. Every time we update the data matrices the computing time grows exponentially as the matrices get bigger. The amount of memory used for storage grows also, but linearly as well as the computing time to estimate the mean vector. However, the interest of developing and implementing this approach is that it is simple, and it gives us the certitude that it works and that it is worth to ameliorate it by means of an algorithm that takes into account the previous SVD. In order to do this amelioration we have studied how to perform the SVD sequentially, as it is explained in the following section.

7.1.2 Sequential Singular Value Decomposition.

For any rectangular matrix $\mathbf{A}$ the singular value decomposition is defined by:

$$\mathbf{A} = \mathbf{U_A}\mathbf{D_A}\mathbf{V_A}^T \tag{7.3}$$

where $\mathbf{U_A}$ and $\mathbf{V_A}$ are two unitary matrices and $\mathbf{D_A}$ is a "diagonal" matrix. This decomposition is a powerful and widely used technique used for many matrix computations, particularly when we have to use properties related to the matrix rank. However, this decomposition is very time consuming, because all the data should be processed at the same time, and when the size of matrix $\mathbf{A}$ become important, it could be difficult to perform (SVD algorithm perform $O(dm^2+d^2m+m^3)$ operations for a $d \times m$ matrix). This is why, we are interested in the use of an incremental way of performing this decomposition keeping it robust.

Based on the algorithms presented in [10; 57; 72], we will present the incremental SVD algorithm that we have retained. The starting point is then an existing SVD of a $d \times m$ data matrix $\mathbf{A} = \{\mathbf{I_1}, ..., \mathbf{I_m}\}$ whose columns $\mathbf{I_i}$ contain the training data, and whose singular value decomposition is expressed as in equation (7.3). When a new matrix $\mathbf{B}$ of dimension $d \times n$ arrives, we want to correctly estimate the SVD of the concatenation of both matrices $[\mathbf{A}\ \mathbf{B}] \stackrel{SVD}{=} \mathbf{U_C}\mathbf{D_C}\mathbf{V_C}^T$. We will call this concatenated matrix $\mathbf{C}$.

Let $\mathbf{L}$ be the projection of $\mathbf{B}$ onto the orthogonal basis $\mathbf{U_A}$ described by

$$\mathbf{L} = \mathbf{U_A}^T\mathbf{B} \tag{7.4}$$

And let $\mathbf{H}$ be the component of $\mathbf{B}$ orthogonal to the subspace spanned by $\mathbf{U_A}$:

$$\mathbf{H} = (\mathbf{I} - \mathbf{U_A}{\mathbf{U_A}}^T)\mathbf{B} = \mathbf{B} - \mathbf{U_A}\mathbf{L} \tag{7.5}$$

Then we need to perform a decomposition of the matrix $\mathbf{H}$ such that an orthogonal matrix is obtained from this decomposition. Commonly the QR decomposition is used in the literature because it is faster. However, in our case we performed the SVD decomposition, because of implementation facility, as we used the OpenCV library where only the SVD is implemented. We could have used another library containing the QR decomposition, but as we have already mixed the OpenGL and the OpenCV library, this could led us to a even more complex code. The decomposition is then:

$$\mathbf{H} = \mathbf{U}_H\mathbf{D}_H\mathbf{V}_H^T \tag{7.6}$$

We can then write the following identity

$$\begin{aligned}\begin{bmatrix} \mathbf{U_A} & \mathbf{U}_H \end{bmatrix}\begin{bmatrix} \mathbf{D_A} & \mathbf{L} \\ \mathbf{0} & \mathbf{U}_H^T\mathbf{H} \end{bmatrix}\begin{bmatrix} \mathbf{V_A} & \mathbf{0} \\ \mathbf{0} & \mathbf{I} \end{bmatrix}^T &= \begin{bmatrix} \mathbf{U_A} & \mathbf{H}\mathbf{V}_H\mathbf{D}_H^{-1} \end{bmatrix}\begin{bmatrix} \mathbf{D_A} & \mathbf{L} \\ \mathbf{0} & \mathbf{U}_H^T\mathbf{H} \end{bmatrix}\begin{bmatrix} \mathbf{V_A} & \mathbf{0} \\ \mathbf{0} & \mathbf{I} \end{bmatrix}^T \\ &= \begin{bmatrix} \mathbf{U_A}\mathbf{D_A} & \mathbf{U_A}{\mathbf{U_A}}^T\mathbf{B}+(\mathbf{I}-\mathbf{U_A}{\mathbf{U_A}}^T)\mathbf{B} \end{bmatrix}\begin{bmatrix} \mathbf{V_A} & \mathbf{0} \\ \mathbf{0} & \mathbf{I} \end{bmatrix}^T \\ &= [\mathbf{A}\ \ \mathbf{B}]\end{aligned} \tag{7.7}$$

where we will call $\mathbf{R} = \begin{bmatrix} \mathbf{D_A} & \mathbf{L} \\ \mathbf{0} & \mathbf{U}_H^T\mathbf{H} \end{bmatrix}$, the square matrix that we need to decompose as $\mathbf{R} = \mathbf{U}_R\mathbf{D}_R\mathbf{V}_R^T$. From this decomposition and from equation (7.7) we can then write the SVD of $[\mathbf{A}\ \ \mathbf{B}]$ as

$$[\mathbf{A}\ \ \mathbf{B}] \ = \ (\begin{bmatrix} \mathbf{U_A} & \mathbf{U}_H \end{bmatrix}\mathbf{U}_R)\mathbf{D}_R(\mathbf{V}_R^T\begin{bmatrix} \mathbf{V_A} & \mathbf{0} \\ \mathbf{0} & \mathbf{I} \end{bmatrix}^T) \tag{7.8}$$

In order to take into account the varying mean, and based on the algorithm presented in [57; 72], it is necessary to add some steps to this algorithm to introduce this variation to the SVD matrices. In [57; 72] the proofs are presented. We must then introduce the mean vector for each which are estimated as $\bar{\mathbf{I}}_\mathbf{A} = \frac{1}{m}\sum_{i=1}^{m}\mathbf{I}_i$, $\bar{\mathbf{I}}_\mathbf{B} = \frac{1}{n}\sum_{i=m+1}^{m+n}\mathbf{I}_i$ and $\bar{\mathbf{I}}_\mathbf{C} = \frac{m}{m+n}\bar{\mathbf{I}}_\mathbf{A} + \frac{n}{m+n}\bar{\mathbf{I}}_\mathbf{B}$. The centred matrices are then denoted $\hat{\mathbf{A}} = \mathbf{A} - \bar{\mathbf{I}}_\mathbf{A}\mathbf{1}_{1\times m}$, $\hat{\mathbf{B}} = \mathbf{B} - \bar{\mathbf{I}}_\mathbf{B}\mathbf{1}_{1\times n}$ and $\hat{\mathbf{C}} = \mathbf{C} - \bar{\mathbf{I}}_\mathbf{C}\mathbf{1}_{1\times m+n}$, being $\mathbf{1}_{a\times b}$ a matrix of dimension $a \times b$ containing only ones. Finally, it is necessary to add the vector $\sqrt{\frac{mn}{m+n}}(\bar{\mathbf{I}}_\mathbf{B} - \bar{\mathbf{I}}_\mathbf{A})$ to the $\hat{\mathbf{B}}$ matrix, in order to take into account the mean variation. We can then resume the complete algorithm as described below.

We start with the matrix decomposition of $\hat{\mathbf{A}} = \mathbf{U_A}\mathbf{D_A}{\mathbf{V_A}}^T$ and then we know $\bar{\mathbf{I}}_\mathbf{A}$ and m. Then, when the new matrix $\mathbf{B}$ arrives we proceed to:

1. Compute the mean vectors $\bar{\mathbf{I}}_\mathbf{B}$ and $\bar{\mathbf{I}}_\mathbf{C}$.
2. Form the matrix $\hat{\mathbf{B}} = \left[\mathbf{I}_{m+1} - \bar{\mathbf{I}}_\mathbf{B} \ldots \mathbf{I}_{m+n} - \bar{\mathbf{I}}_\mathbf{B}\ \ \sqrt{\frac{mn}{m+n}}(\bar{\mathbf{I}}_\mathbf{B} - \bar{\mathbf{I}}_\mathbf{A})\right]$

3. Compute the matrix$\mathbf{H} = \mathbf{I} - \mathbf{U_A}{\mathbf{U_A}}^T\hat{\mathbf{B}}$ and its SVD decomposition $\mathbf{H} = \mathbf{U}_H\mathbf{D}_H\mathbf{V}_H^T$

4. Form the matrix $\mathbf{R} = \begin{bmatrix} \mathbf{D_A} & \mathbf{L} \\ \mathbf{0} & \mathbf{U}_H^T\mathbf{H} \end{bmatrix}$

5. Compute the SVD decomposition of $\mathbf{R} = \mathbf{U}_R\mathbf{D}_R\mathbf{V}_R^T$.

6. Finally $\mathbf{U_C} = (\begin{bmatrix} \mathbf{U_A} & \mathbf{U}_H \end{bmatrix}\mathbf{U}_R)$, $\mathbf{D_C} = \mathbf{D}_R$ and $\mathbf{V_C} = (\begin{bmatrix} \mathbf{V_A} & \mathbf{0} \\ \mathbf{0} & \mathbf{I} \end{bmatrix}\mathbf{V}_R)$

This algorithm can be also extended to the case when a forgetting factor is introduced, to take into account that the model evolves in time and that what was learned at the beginning can be incoherent with the data that is obtained after some time. To introduce this forgetting factor we have firstly to modify the mean update. It can be rewritten as:

$$\bar{\mathbf{I}}_\mathbf{C} = \frac{fm}{fm+n}\bar{\mathbf{I}}_\mathbf{A} + \frac{n}{fm+n}\bar{\mathbf{I}}_\mathbf{B} \tag{7.9}$$

where f denotes the forgetting factor. Finally, to modify the influence of the old vectors, we need to introduce f in the $\mathbf{R}$ matrix to reduce the effect of the old vectors with respect to the new vectors. That is written as:

$$\mathbf{R} = \begin{bmatrix} f\mathbf{D_A} & \mathbf{L} \\ \mathbf{0} & \mathbf{U}_H^T\mathbf{H} \end{bmatrix} \tag{7.10}$$

We will apply now incremental SVD to the CCA.

7.1.3 Application of the incremental SVD to the CCA.

In order to develop the CCA, we recall from section 4.3.1 the equations $\mathbf{A}_1 = \mathbf{U}_1\mathbf{D}_1\mathbf{V}_1^T$, $\mathbf{A}_2 = \mathbf{U}_2\mathbf{D}_2\mathbf{V}_2^T$, the matrix $\mathbf{V}_1^T\mathbf{V}_2 = \mathbf{U}\mathbf{D}\mathbf{V}^T$, that in this case needs to be stored, and the equation (4.32), $\mathbf{G} = \mathbf{U}_2\mathbf{D}_2\mathbf{V}\mathbf{U}^T\mathbf{D}_1^{-1}\mathbf{U}_1^T$.

These equations represents all the matrices involved in the CCA, that we want to update using the incremental SVD described before. Then, according to the algorithm described above, we proceed with the incremental CCA.

1. Every n new images we form two new matrices $\mathbf{B}_1$ and $\mathbf{B}_2$ containing respectively the new images $\mathbf{I_i}$ and the corresponding perturbations $\Delta\mathbf{b}_i$.

2. We compute the mean vectors $\bar{\mathbf{I}}_{\mathbf{B_1}}$, $\bar{\mathbf{I}}_{\mathbf{A_1}}^{\mathbf{(new)}}$, $\bar{\mathbf{\Delta b}}_{\mathbf{B_2}}$, and $\bar{\mathbf{\Delta b}}_{\mathbf{A_2}}^{\mathbf{(new)}}$.

3. Then we form the matrix $\hat{\mathbf{B}}_1 = \left[\mathbf{I}_{m+1} - \bar{\mathbf{I}}_{\mathbf{B_1}} \ldots \mathbf{I}_{m+n} - \bar{\mathbf{I}}_{\mathbf{B_1}} \ \sqrt{\frac{mn}{m+n}}(\bar{\mathbf{I}}_{\mathbf{B_1}} - \bar{\mathbf{I}}_{\mathbf{A_1}}^{\mathbf{(old)}})\right]$ and $\hat{\mathbf{B}}_2 = \left[\Delta\mathbf{b}_{m+1} - \bar{\Delta\mathbf{b}}_{B_2} \ldots \Delta\mathbf{b}_{m+n} - \bar{\Delta\mathbf{b}}_{B_2} \ \sqrt{\frac{mn}{m+n}}(\bar{\mathbf{\Delta b}}_{\mathbf{B_2}} - \bar{\mathbf{\Delta b}}_{\mathbf{A_2}}^{\mathbf{(old)}})\right]$

4. Compute the matrices $\mathbf{H}_1$ and $\mathbf{H}_2$, as defined in equation (7.5), and their SVD decompositions $\mathbf{H}_1 = \mathbf{U}_{H_1}\mathbf{D}_{H_1}\mathbf{V}_{H_1}^T$ and $\mathbf{H}_2 = \mathbf{U}_{H_2}\mathbf{D}_{H_2}\mathbf{V}_{H_2}^T$.

5. Form the matrices $\mathbf{R}_1 = \begin{bmatrix} \mathbf{D_{A_1}} & \mathbf{L_1} \\ \mathbf{0} & \mathbf{U}_{H_1}^T\mathbf{H_1} \end{bmatrix}$ and $\mathbf{R}_2 = \begin{bmatrix} \mathbf{D_{A_2}} & \mathbf{L_2} \\ \mathbf{0} & \mathbf{U}_{H_2}^T\mathbf{H_2} \end{bmatrix}$
6. Compute the SVD decompositions of $\mathbf{R_1} = \mathbf{U}_{R_1}\mathbf{D}_{R_1}\mathbf{V}_{R_1}^T$ and $\mathbf{R_2} = \mathbf{U}_{R_2}\mathbf{D}_{R_2}\mathbf{V}_{R_2}^T$.
7. Update the matrices as: $\mathbf{U_1^{(new)}} = (\begin{bmatrix} \mathbf{U_1} & \mathbf{U}_{H_1} \end{bmatrix}\mathbf{U}_{R_1})$, $\mathbf{D_1^{(new)}} = \mathbf{D}_{R_1}$, $\mathbf{U_2^{(new)}} = (\begin{bmatrix} \mathbf{U_2} & \mathbf{U}_{H_2} \end{bmatrix}\mathbf{U}_{R_2})$ and $\mathbf{D_2^{(new)}} = \mathbf{D}_{R_2}$.
8. Compute the SVD of $\mathbf{V_{R_1}^T}\begin{bmatrix} \mathbf{V_{A_1}}^T\mathbf{V_{A_2}} & \mathbf{0} \\ \mathbf{0} & \mathbf{I} \end{bmatrix}\mathbf{V}_{R_2} = \mathbf{U^{(new)}}\mathbf{D^{(new)}}\mathbf{V^{(new)}}^T$
9. Finally update the matrix $\mathbf{G} = \mathbf{U}_2^{(new)}\mathbf{D}_2^{(new)}\mathbf{V}^{(new)}\mathbf{U}^{(new)T}\mathbf{D}_1^{(new)-1}\mathbf{U}_1^{(new)T}$

We can see that in this case we can store the multiplication $\mathbf{V}_{A_1}^T\mathbf{V}_{A_2}$ at each update in order to save one matrix multiplication. As it is discussed in more detail in the appendix B, we used the truncated SVD decomposition, i.e. we take into account only the most important singular values, and the dimensional analysis is presented in this appendix. This truncation is in order to limit the size of the matrices as new data arrives, keeping only the most important eigenvalues and the corresponding reduced matrices. We have chosen to keep the eigenvectors such as the ratio between the minimum and the maximum eigenvalue was of 5×10^5. This value was obtained empirically by testing a dozen video sequences.

7.2 Implementation and results.

We have used the same equipment as explained in 4.5. In order to prove the stability and robustness of the tracker we have used the talking face video sequence as well as the LaCascia video sequences. It is important to say that we have tested two versions of the algorithm, the direct approach described in 7.1.1, and the approach described in 7.1.3. Both approaches were implemented to update the six pose parameters only, keeping the facial gesture parameters unchanged. This update is performed every five frames, in a similar way as it was done in [72], as a trade-off between computational efficiency and effectiveness of modeling appearance change during fast motion.

We have first tested our algorithm with the LaCascia video sequences, but the size of these videos is not great enough to see if there is an improvement of the performance or not. This is because we updated the CCA coefficients every 5 frames and these video sequences contain only 200 frames.

Then, we have used the talking face video sequence, but in this case we have used only 2000 frames, instead of the 5000 frames. The reasons to do this are that using the 2000 frames is enough to understand the behavior of the algorithm, and the direct approach becomes unfeasible when we use a very big amount of data. For the direct approach, the computing time behaves exponentially w.r.t. the amount of data used.

The first test we have performed is, as previously explained, the implementation of the incremental CCA using the data matrices. This approach, although led us to see the interest of updating the CCA coefficients, presents the disadvantage of being very slow for the training update, especially as more data arrive. The behavior of the computing time can be seen in figure 7.1. We can see that the computing time as a function of the number of frames presents an exponential growth.

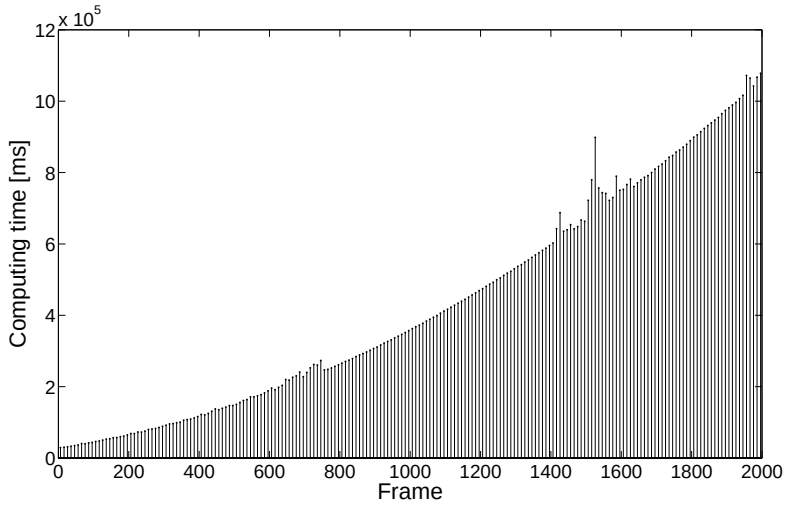


Figure 7.1: Computing time for the CCA updated recalculated every 5 frames.

By contrast, if we see the computing time for the i-CCA algorithm, we can appreciate that the updating computing time is almost constant. The figure 7.2 shows this computing time.

Finally, we have compared the mean point to point error between the CCA and the incremental CCA. Results are reported in figure 7.3. We see that the incremental property introduced to the CCA algorithm improves the robustness in long sequences. This can be seen from frame 1500, where the mean point to point error performs better than for the case where no CCA update is not performed. However, the cost to be paid for this amelioration is that the algorithm has to be used off-line, because introducing new data into the CCA coefficients takes more than 16 seconds.

However, special care should be taken when using this algorithm in video sequences with occlusions and drastic illumination changes, because our tracker is not robust in the presence of outliers. In the presence of outliers our tracker becomes unstable and diverges even in easy situations just after some updates with this erroneous data. To avoid that, we should test if the estimation obtained gave us a correct solution or not, and in the case of determining that the solution is correct, then we can use this data to update the coefficients.

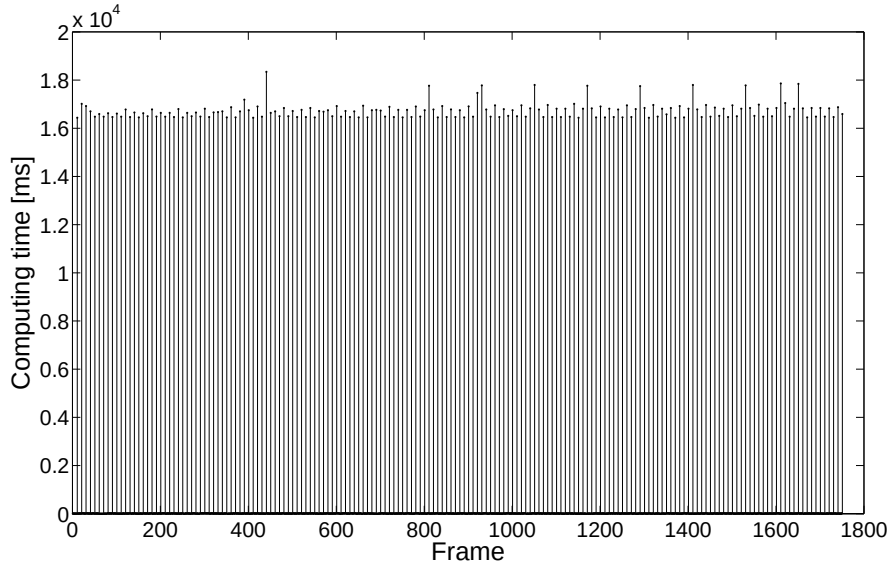


Figure 7.2: Computing time for the i-CCA updated every 5 frames.

7.3 Conclusions.

We have shown in this chapter the interest of updating the CCA coefficients. The principal advantage of this algorithm is that we can introduce the external variations to the CCA coefficients, corresponding to slow changes in illumination and pose, in order to adapt it to the changing conditions of a video sequence. We have proposed an algorithm to perform the CCA update of the parameters such that every time we update the CCA coefficients, the computing time can be considered constant. Although this method makes this algorithm slower and thus, inappropriate for real-time applications, it can be applied in two ways.

The first option is to analyze each image result in order to determine if it is worth to use this image to update the CCA coefficient or not, depending on the difference between this image and the database. If the image does not contribute significantly with respect to the learned database, then this image should be discarded for the CCA coefficients update. In this way, we will store only images that contribute with new data and only when an important change is present, so the computing time is not constant but evolves depending on the variations of the video sequence. However, special care should be taken to avoid using wrong estimations, because this can introduce errors to the coefficients, and make the algorithm diverge.

The second option consists in obtaining the CCA coefficients by means of the incremental CCA. In this way we can reduce the complexity of the algorithm explained in chapter 6, where a database containing multiple faces was used. Using the algorithm in this way, we can also learn more people sequentially and also include aging and evolution parameters for applications where the tracker is optimized to recognize a specific person. In other words, we can update the

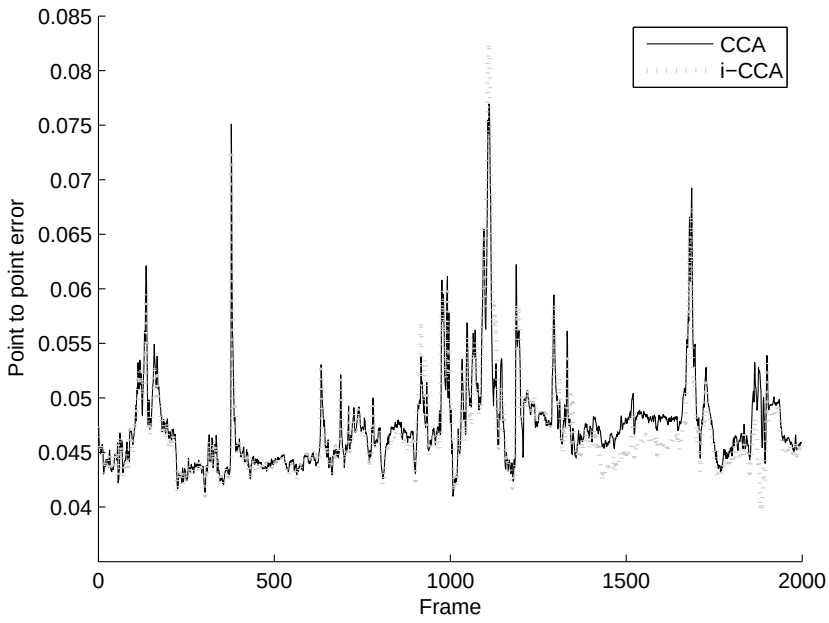


Figure 7.3: Results of the CCA incremental compared with the normal CCA.

CCA coefficients each time that we find a person for whom the CCA coefficients are not well adapted.

Chapter 8

Conclusions and Perspectives.

In this thesis we have presented a supervised learning method based on the canonical correlation analysis (CCA), to perform:

- ⋆ the tracking of rigid faces (chapter 4),
- ⋆ the tracking of the 3D pose and facial animation (chapter 5),
- ⋆ the estimation of the pose and shape of unknown faces (chapter 6),
- ⋆ and the tracking by means of an incremental algorithm (chapter 7).

In order to do that, we use the 3D parametric face model *Candide*. With this parametric model, we create a normalized expression-free face patch that corresponds to our observation vector $\mathbf{x}_t$. The parameters used to manipulate this face model are set inside a state vector $\mathbf{b}_t$. We use then these parameters to create two data sets. One data set contains synthesized images created by modifying the 3D pose parameters, and the shape and animation parameters, and the other data set contains the parameter's perturbations $\Delta\mathbf{b}_t$ used to create the synthesized faces. Then, by means of the CCA, we obtain the linear relation that exists between the variations in the state vector $\Delta\mathbf{b}_t$ and the corresponding variations in the input vector with respect to the reference $\Delta\mathbf{x}_t = \mathbf{x}_t - \mathbf{x}_t^{(ref)}$ such that this relation can be expressed as $\Delta\mathbf{b}_t = \mathbf{G}\Delta\mathbf{x}_t$.

In chapter 4, we have proposed two versions of our approach. A linear version where the matrix $\mathbf{G}$ is learned by means of the CCA and a non linear where the matrix $\mathbf{G}$ is learned by means of the kernel-CCA (KCCA). We have seen that these two approaches estimate correctly the 3D pose of faces. However, we have noticed that there is not an significant improvement when we use the KCCA, and the computing cost is too high, in terms of time and memory requirements. Therefore, we have extended the CCA algorithm to track some facial gesture parameters of the 3D face model in chapter 5.

We have tested the robustness and behavior of our algorithm under different situations to track both 3D pose and facial animation parameters of individuals in monocular video sequences. The first thing we have seen was that we can

track independently the 3D pose and the facial gesture. Based on this, we have proposed to use three different models. One to track the 3D pose, one to track the facial animation of the mouth, and another to track the facial animation of the eyes. This presents two principal advantages. Firstly we have three smaller training sets to estimate the CCA coefficients, making the training faster, and secondly, we have used different resolutions to analyze the local movements corresponding to facial gesture, which improves the facial gesture tracking.

Then we have compared the algorithm with a local approach that uses small windows around some characteristic points. This local model presents the advantage of being faster and robust enough when there are neither strong out-of-plane rotations nor important facial gesture. It also presents a robust behavior when faced to important illumination changes, specially because each local feature is normalized independently of the others features. However, we can conclude that the global approach represents a better solution for real world conditions, where important rotations can appear and facial gestures are expected.

We have then compared the behavior of the algorithm when using a reference vector, corresponding to the first image, with respect to the use of the mean vector of the training process. The results obtained showed that both approaches perform well, and the only difference was that no update step was required for the approach using the mean vector. Depending on the kind of application we can then use one approach or the other.

Afterward, we have tested the use of the components of different representation color spaces. The fact of using a transformation of the RGB space to another representation of the image can ameliorate the tracking results, especially for the mouth region. We have seen that the use of the V component of the HSV transformation for the tracking of the 3D pose and eyes facial gesture, and the a component for the tracking of the mouth's facial gesture performed slightly better than the use of the grayscale transformation. However, this increases the complexity of the tracker, and visually we can not see a big difference between the tracking results. This is why we have decided to keep the grayscale transformation to represent the appearance of the face.

It was proved with the simulation results that the proposed approach gives a fast and simple tracker, that is robust and accurate when the out-of-plane face rotation angles stay in the interval $\pm 30^\circ$ and when there are slight illumination changes. However, we observed from simulations that the effectiveness of this kind of tracker is dependent, first on the training set we use, and secondly on the initialization of the geometric model. For the training we have chosen a small dataset that was obtained empirically, and that proved to be robust for the desired purposes. For the case of the 3D mask initialization, the pose and the facial features must be as realistic as possible at the first frame, especially if we want to keep the robustness for the out of plane rotations.

From simulations, we have seen that introducing a limit for each parameter could improve the performance of the tracking. This means that instead of taking the estimation of the vector state's update as we get it, we compare it to a given limit, such that the resulting pose estimate was not an incoherent one with re-

spect to the algorithm design. What we observe, is that the resulting algorithm is more robust when track is lost, because it keeps at least a small portion of the face in track, specially for out-of-plane rotation along the y axis, and when the person returns to a more neutral position, the model comes back to the correct pose and continues the pose estimation. This takes us to conclude that including a temporal continuity constraints across frames and to deal with dynamics of the tracked object could improve the performance of this approach, as in [7; 42; 59; 61; 65; 82], specially doing more smooth the tracking, which corresponds more to the natural movements of a person. This can be obtained by means of a Kalman filter or a particle filter, for example.

Then, we have presented in chapter 6 an algorithm capable of automatically estimating the 3D pose and shape of unknown faces in images, where the face is close to a frontal view. In this case we have used the Adaboost algorithm present in the OpenCV library to obtain an initial approach of the face position. This algorithm was similar to the algorithm used for tracking, but the data used for training and the state vector changes. We have seen from experiments that using different random points for each people improved the performance of the training, giving a more robust tracker.

This algorithm can be used as a robust method to automatically initialize a tracker that uses the *Candide* model, or to recover after tracking loss due to severe occlusions. In the case of the recovery after track losing, we have two options. The first is to use the algorithm described here, as an initialization problem. The second is to use the Adaboost algorithm, and the linear model that converts the pixel window to OpenGL 3D position parameters, and use the coefficients estimated from the first image. To decide which of these approaches is better to be used will depend in whether or not the person goes out of the camera's vision region, or if it was occluded, which implies that other algorithm that takes into account the geometry of the camera's vision region should be used.

Finally we have shown the interest of updating the CCA coefficients. The principal advantage of this algorithm is that we can introduce the external variations to the CCA coefficients in an efficient way. To this purpose, we had proposed an incremental CCA algorithm that uses the incremental SVD to update the CCA coefficients.

As we have seen, human face processing, and more particularly human face tracking is a very challenging domain that still have a very big amount of research to be done. Even if there are people trackers that perform well and help people to perform some tasks, they still being far from the human visual discrimination capacity. Moreover, the ever growing computer's processing power is pushing the research community to improve and propose new computer vision methods. Among the research areas that are currently in process of exploration there are the sequential principal component analysis, multidimensional space decomposition, the use of 3D scans instead of traditional 2D images for tracking, stereo-vision based trackers,etc.

One research line that can improve our approach is the use of the 2D Principal Component Analysis and 2D Singular Value Decomposition. Traditionally, 2D objects, as images or matrices, are converted into 1D vectors and then are packed together as a large matrix, to then apply a low rank approximation method as the SVD. Although this approach is widely used, as in the case of the eigenfaces, or our algorithm, the fact of rearranging images as vectors creates matrices of very high dimensions that are ill conditioned and produces a 2D lose of information. We can cite for example the work presented in [93], where a 2D PCA approach is proposed for face reconstruction and it is compared with the eigenfaces approach. It is shown that the 2D approach requires less principal components to reconstruct correctly faces from a dataset. It uses for these, the image covariance matrix, and it estimates from it the eigenvectors that will represent the face. Another example can be found in [27], where the authors propose an approach to use a 2D SVD to reconstruct faces and map images. This is extended in [49] where the authors propose a multilinear SVD, that is a generalization of the SVD problem but for N dimensions.

Another approach that is being used because of its application to non linear and non Gaussian problems is given by the particle filters. Particle filters, as explained in chapter 3 consist in using multiple samples of the state vector that have a weight associated to it, and that evolve in time. This can be used to improve the tracking when severe occlusions are present, because some particles are supposed to be close to the true face position when the occlusion will end.

Fusion of several cues can also be used as in [104] where contour information and color are fused in order to track shapes. The morphable model described in [68; 69; 70] also uses multiple cues to obtain a cost function that is better behaved, as the 3D model information, the appearance and the edges. We can also talk about the work presented in [66], where color, motion and sound are fused to track talking people.

As we can see there have been a lot of improvements in the face tracking algorithms, and the fact that the technology evolves every day, will led us to develop algorithms more and more complex that will be applied in real time applications. These applications are also growing as the algorithm become more and more complex, letting us not only to estimate simple 3D parameters, but also detect emotions or strange behaviors, and they will change the way human interact with computers.

Appendix A

Canonical Correlation Analysis (CCA).

A.1 Introduction

The signification of the correlation can be viewed as the linear relation that exists between two variables. When the correlation is zero, that means that there is not a linear relation between these two variables, nevertheless it can be a non linear relation between these variables.

The correlation depends very much on the coordinate system used. We can rotate the coordinate system such that the projections in the new system are maximally uncorrelated. That is achieved by the principal component analysis (PCA). Nonetheless, we are interested in finding the correlations between two sets of variables.

Canonical correlation analysis (CCA) is a way of identifying and quantifying the linear relationship between two data sets. CCA can be seen as the problem of finding direction vectors for two data sets such that the correlation between the projections of the variables onto these direction vectors are mutually maximized. The dimensionality of these new bases is equal to or less than the smallest dimensionality of the two variables. The canonical correlation coefficients can be calculated directly from the two data sets.

An important property of canonical correlations is that they are invariant with respect to affine transformations of the variables. In this appendix we will present the formulation of the CCA problem and the procedure to obtain the solution. Finally, the formulation presented by [87] will also be introduced.

A.1.1 Definition and Derivation of Canonical Correlation Analysis equations.

As we have stated, CCA aims to maximize the correlations between two data sets. Let $\mathbf{A_1}$ and $\mathbf{A_1}$ be two centered data sets of dimension $d \times m$ and $p \times m$ respectively. The maximum number of correlations that can be found is equal to

the minimum of the data sets' column dimension $\min(d, p)$. If we project our data in the directions $\mathbf{w_1}$ and $\mathbf{w_2}$ we obtain two new vectors defined as:

$$\mathbf{z_1} = \mathbf{A_1}^T \mathbf{w_1} \tag{A.1}$$

and

$$\mathbf{z_2} = \mathbf{A_2}^T \mathbf{w_2}. \tag{A.2}$$

These vectors are called the *scores* or the *canonical variates*. It is important to mention that in the literatures these definitions could be found in a different way, being the canonicals defined in terms of the transposed vectors $\mathbf{w_1}$ and $\mathbf{w_2}$, [8],[26], or with the matrices $\mathbf{A_1}$ and $\mathbf{A_1}$ containing row vectors, as in [87]. In our case we use column vectors for the data matrices and for the direction vectors.

Once we have defined the *canonical variates*, we are interested in finding the correlation between them, which is defined as:

$$\rho = \frac{\mathbf{z_2}^T \cdot \mathbf{z_1}}{\sqrt{\mathbf{z_2}^T \cdot \mathbf{z_2}}\sqrt{\mathbf{z_1}^T \cdot \mathbf{z_1}}}. \tag{A.3}$$

We must bold that ρ is not affected if we scale the canonical variates. This implies that we can use the following constraints:

$$\mathbf{z_1}^T \cdot \mathbf{z_1} = \mathbf{w_1}^T \mathbf{A_1} \mathbf{A_1}^T \mathbf{w_1} = \mathbf{w_1}^T \Sigma_{11} \mathbf{w_1} = 1 \tag{A.4}$$

and

$$\mathbf{z_2}^T \cdot \mathbf{z_2} = \mathbf{w_2}^T \mathbf{A_2} \mathbf{A_2}^T \mathbf{w_2} = \mathbf{w_2}^T \Sigma_{22} \mathbf{w_2} = 1. \tag{A.5}$$

This can be expressed as the following optimization problems:

$$\begin{cases} \min \|\mathbf{z}_1 - \mathbf{z}_2\|^2 \\ \|\mathbf{z}_1\| = \|\mathbf{z}_2\| = 1 \end{cases} \Leftrightarrow \begin{cases} \max \mathbf{z}_2^T \mathbf{z}_1 \\ \|\mathbf{z}_1\| = \|\mathbf{z}_2\| = 1 \end{cases}. \tag{A.6}$$

If we define $\Sigma_{21} = \mathbf{A_2} \mathbf{A_1}^T$ we can write our problem in a Lagrangian form:

$$L(\rho_1, \rho_2, \mathbf{w_1}, \mathbf{w_2}) = \mathbf{w_2}^T \Sigma_{21} \mathbf{w_1} - \frac{\rho_1}{2}(\mathbf{w_1}^T \Sigma_{11} \mathbf{w_1} - 1) - \frac{\rho_2}{2}(\mathbf{w_2}^T \Sigma_{22} \mathbf{w_2} - 1). \tag{A.7}$$

Taking the derivatives of (A.7) with respect to $\mathbf{w_1}$ and $\mathbf{w_2}$ we arrive to:

$$\frac{\partial L}{\partial \mathbf{w_1}} = \Sigma_{12} \mathbf{w_2} - \rho_1 \Sigma_{11} \mathbf{w_1} = 0 \tag{A.8}$$

and

$$\frac{\partial L}{\partial \mathbf{w_2}} = \Sigma_{21} \mathbf{w_1} - \rho_2 \Sigma_{22} \mathbf{w_2} = 0. \tag{A.9}$$

Taking these equations and multiplying (A.8) by $\mathbf{w_1}^T$ and subtracting the result from $\mathbf{w_2}^T$ times (A.9), we arrive to:

$$\mathbf{w_2}^T\Sigma_{21}\mathbf{w_1} - \mathbf{w_1}^T\Sigma_{12}\mathbf{w_2} + \rho_1\mathbf{w_1}^T\Sigma_{11}\mathbf{w_1} - \rho_2\mathbf{w_2}^T\Sigma_{22}\mathbf{w_2} = 0$$

Since the correlation is symmetric, we can reduce the expression to:

$$\rho_1\mathbf{w_1}^T\Sigma_{11}\mathbf{w_1} - \rho_2\mathbf{w_2}^T\Sigma_{22}\mathbf{w_2} = 0$$

and using the constrains of equations (A.4) et (A.5) we can see that $\rho_1 = \rho_2 = \rho$. If Σ_{11} is invertible, we can obtain from equation (A.8) the relation between $\mathbf{w_1}$ and $\mathbf{w_2}$:

$$\mathbf{w_1} = \frac{\Sigma_{11}^{-1}\Sigma_{12}\mathbf{w_2}}{\rho}. \tag{A.10}$$

and if we replace $\mathbf{w_1}$ in (A.9) we obtain the equation:

$$(\Sigma_{21}\Sigma_{11}^{-1}\Sigma_{12} - \rho^2\Sigma_{22})\mathbf{w_2} = 0. \tag{A.11}$$

In the same way, if Σ_{22} is invertible, we can obtain from (A.9)

$$\mathbf{w_2} = \frac{\Sigma_{22}^{-1}\Sigma_{21}\mathbf{w_1}}{\rho} \tag{A.12}$$

and replacing it in (A.8) we arrive to:

$$(\Sigma_{12}\Sigma_{22}^{-1}\Sigma_{21} - \rho^2\Sigma_{11})\mathbf{w_1} = 0. \tag{A.13}$$

As we have assumed that Σ_{11} and Σ_{22} are invertible, we can accommodate these equations as:

$$\Sigma_{22}^{-1}\Sigma_{21}\Sigma_{11}^{-1}\Sigma_{12}\mathbf{w_2} = \rho^2\mathbf{w_2} \tag{A.14}$$

and

$$\Sigma_{11}^{-1}\Sigma_{12}\Sigma_{22}^{-1}\Sigma_{21}\mathbf{w_1} = \rho^2\mathbf{w_1}. \tag{A.15}$$

Using the equations (A.14) and (A.15) we can obtain the *canonical variates* by obtaining the eigenvalues and the eigenvectors of this equation. The number of non zero eigenvalues of this equation are limited to the smallest dimensionality of $\mathbf{A_1}$ and $\mathbf{A_2}$ [8] and the corresponding eigenvectors $\mathbf{w_1}$ and $\mathbf{w_2}$ are the *canonical correlation direction vectors*.

From these equations, we can see that to obtain these vectors first we need to perform three matrix multiplications to obtain the covariance's matrix, and then we have to do two matrix inversions and then three matrix multiplication. Finally one Singular Value Decomposition is needed. However, as we have the data matrices $\mathbf{A_x}$ and $\mathbf{A_y}$ we will develop the method proposed in [87] that reduce the number of operation and thus, is supposed to be more robust numerically.

A.1.2 Solution of the Canonical Correlation Analysis equations from Data Matrices.

The first thing that we need to do is to perform a singular value decomposition of the data matrices $\mathbf{A_1}$ and $\mathbf{A_2}$:

$$\mathbf{A_1} = \mathbf{U}_{A_1}\mathbf{D}_{A_1}\mathbf{V}_{A_1}^T \tag{A.16}$$

$$\mathbf{A_2} = \mathbf{U}_{A_2}\mathbf{D}_{A_2}\mathbf{V}_{A_2}^T \tag{A.17}$$

We use these decompositions to obtain the matrices:

$$\begin{array}{rcccl} \Sigma_{11} & = & \mathbf{A_1}\mathbf{A_1}^T & = & \mathbf{U}_{A_1}\mathbf{D}_{A_1}^2\mathbf{U}_{A_1}^T \\ \Sigma_{22} & = & \mathbf{A_2}\mathbf{A_2}^T & = & \mathbf{U}_{A_2}\mathbf{D}_{A_2}^2\mathbf{U}_{A_2}^T \\ \Sigma_{12} & = & \mathbf{A_1}\mathbf{A_2}^T & = & \mathbf{U}_{A_1}\mathbf{D}_{A_1}\mathbf{V}_{A_1}^T\mathbf{V}_{A_2}\mathbf{D}_{A_2}\mathbf{U}_{A_2}^T \\ \Sigma_{21} & = & \mathbf{A_2}\mathbf{A_1}^T & = & \mathbf{U}_{A_2}\mathbf{D}_{A_2}\mathbf{V}_{A_2}^T\mathbf{V}_{A_1}\mathbf{D}_{A_1}\mathbf{U}_{A_1}^T, \end{array} \tag{A.18}$$

and the inverse of the covariance matrix are obtained using the property that $\mathbf{A_1}^{-1} = \mathbf{V}_{A_1}\mathbf{D}_{A_1}^{-1}\mathbf{U}_{A_1}^T$, what gives:

$$\begin{array}{rcl} \Sigma_{11}^{-1} & = & (\mathbf{A}_{A_1}\mathbf{A}_{A_1}^T)^{-1} \\ & = & \mathbf{A}_{A_1}^{-T}\mathbf{A}_{A_1}^{-1} \\ & = & \mathbf{U}_{A_1}\mathbf{D}_{A_1}^{-1}\mathbf{V}_{A_1}^T\mathbf{V}_{A_1}\mathbf{D}_{A_1}^{-1}\mathbf{U}_{A_1}^T \\ & = & \mathbf{U}_{A_1}\mathbf{D}_{A_1}^{-2}\mathbf{U}_{A_1}^T. \end{array}$$

Similarly we obtain that $\Sigma_{22}^{-1} = \mathbf{U}_{A_2}\mathbf{D}_{A_2}^{-2}\mathbf{U}_{A_2}^T$. If we replace in (A.11) we obtain

$$(\mathbf{U}_{A_2}\mathbf{D}_{A_2}\mathbf{V}_{A_2}^T\mathbf{V}_{A_1}\mathbf{D}_{A_1}\mathbf{U}_{A_1}^T\mathbf{U}_{A_1}\mathbf{D}_{A_1}^{-2}\mathbf{U}_{A_1}^T\mathbf{U}_{A_1}\mathbf{D}_{A_1}\mathbf{V}_{A_1}^T\mathbf{V}_{A_2}\mathbf{D}_{A_2}\mathbf{U}_{A_2}^T - \rho^2\mathbf{U}_{A_2}\mathbf{D}_{A_2}^2\mathbf{U}_{A_2}^T)\mathbf{w_2} = 0$$

After simplifying we arrive to:

$$(\mathbf{U}_{A_2}\mathbf{D}_{A_2}\mathbf{V}_{A_2}^T\mathbf{V}_{A_1}\mathbf{V}_{A_1}^T\mathbf{V}_{A_2}\mathbf{D}_{A_2}\mathbf{U}_{A_2}^T - \rho^2\mathbf{U}_{A_2}\mathbf{D}_{A_2}^2\mathbf{U}_{A_2}^T)\mathbf{w_2} = 0. \tag{A.19}$$

If we multiply from the left by $\mathbf{D}_{A_2}^{-1}\mathbf{U}_{A_2}^T$ we obtain:

$$(\mathbf{D}_{A_2}^{-1}\mathbf{U}_{A_2}^T\mathbf{U}_{A_2}\mathbf{D}_{A_2}\mathbf{V}_{A_2}^T\mathbf{V}_{A_1}\mathbf{V}_{A_1}^T\mathbf{V}_{A_2}\mathbf{D}_{A_2}\mathbf{U}_{A_2}^T - \rho^2\mathbf{D}_{A_2}^{-1}\mathbf{U}_{A_2}^T\mathbf{U}_{A_2}\mathbf{D}_{A_2}^2\mathbf{U}_{A_2}^T)\mathbf{w_2} = 0$$

that reduces to

$$((\mathbf{V}_{A_1}^T\mathbf{V}_{A_2})^T(\mathbf{V}_{A_1}^T\mathbf{V}_{A_2}) - \rho^2\mathbf{I})\mathbf{D}_{A_2}\mathbf{U}_{A_2}^T\mathbf{w_2} = 0 \tag{A.20}$$

We obtain then the singular value decomposition of $\mathbf{V}_{A_1}^T\mathbf{V}_{A_2} = \mathbf{U}\mathbf{D}\mathbf{V}^T$ and we solve the equation as:

$$\begin{array}{rcl} 0 & = & ((\mathbf{U}\mathbf{D}\mathbf{V}^T)^T(\mathbf{U}\mathbf{D}\mathbf{V}^T) - \rho^2\mathbf{I})\mathbf{D}_{A_2}\mathbf{U}_{A_2}^T\mathbf{w_2} \\ & = & (\mathbf{V}\mathbf{D}^2\mathbf{V}^T - \rho^2\mathbf{I})\mathbf{D}_{A_2}\mathbf{U}_{A_2}^T\mathbf{w_2}, \end{array}$$

and after some rearrangement takes us to:

$$(\mathbf{D}^2 - \rho^2\mathbf{I})\mathbf{V}^T\mathbf{D}_{A_2}\mathbf{U}_{A_2}^T\mathbf{w_2} = 0 \tag{A.21}$$

This equation has eigenvalues $\mathbf{D^2}$ and the eigenvectors can be obtained from the columns of $\mathbf{U}_{A_2}\mathbf{D}_{A_2}^{-1}\mathbf{V}$

With a similar procedure we can obtain the simplification of (A.13).

$$(\mathbf{U}_{A_1}\mathbf{D}_{A_1}\mathbf{V}_{A_1}^T\mathbf{V}_{A_2}\mathbf{V}_{A_2}^T\mathbf{V}_{A_1}\mathbf{D}_{A_1}\mathbf{U}_{A_1}^T - \rho^2\mathbf{U}_{A_1}\mathbf{D}_{A_1}^2\mathbf{U}_{A_1}^T)\mathbf{w_1} = 0$$

$$(\mathbf{D}_{A_1}^{-1}\mathbf{U}_{A_1}^T\mathbf{U}_{A_1}\mathbf{D}_{A_1}\mathbf{V}_{A_1}^T\mathbf{V}_{A_2}\mathbf{V}_{A_2}^T\mathbf{V}_{A_1}\mathbf{D}_{A_1}\mathbf{U}_{A_1}^T - \rho^2\mathbf{D}_{A_1}^{-1}\mathbf{U}_{A_1}^T\mathbf{U}_{A_1}\mathbf{D}_{A_1}^2\mathbf{U}_{A_1}^T)\mathbf{w_1} = 0$$

$$(\mathbf{V}_{A_1}^T\mathbf{V}_{A_2}\mathbf{V}_{A_2}^T\mathbf{V}_{A_1} - \rho^2\mathbf{I})\mathbf{D}_{A_1}\mathbf{U}_{A_1}^T\mathbf{w_1} = 0$$

$$((\mathbf{V}_{A_1}^T\mathbf{V}_{A_2})(\mathbf{V}_{A_1}^T\mathbf{V}_{A_2})^T - \rho^2\mathbf{I})\mathbf{D}_{A_1}\mathbf{U}_{A_1}^T\mathbf{w_1} = 0$$

$$\begin{aligned} 0 &= ((\mathbf{U}\mathbf{D}\mathbf{V}^T)(\mathbf{U}\mathbf{D}\mathbf{V}^T)^T - \rho^2\mathbf{I})\mathbf{D}_{A_1}\mathbf{U}_{A_1}^T\mathbf{w_1} \\ &= (\mathbf{U}\mathbf{D^2}\mathbf{U}^T - \rho^2\mathbf{I})\mathbf{D}_{A_1}\mathbf{U}_{A_1}^T\mathbf{w_1}, \end{aligned}$$

to finally obtain:

$$(\mathbf{D}^2 - \rho^2\mathbf{I})\mathbf{U}^T\mathbf{D}_{A_1}\mathbf{U}_{A_1}^T\mathbf{w_1} = 0 \tag{A.22}$$

where the eigenvectors are the columns of $\mathbf{U}_{A_1}\mathbf{D}_{A_1}^{-1}\mathbf{U}$.

Appendix B

Implementation details.

In this chapter we present the details of the matrix dimensions, in order to make straightforward the implementation of the proposed algorithms. In order to do that, we will present first the dimensions of the data matrices, then the dimensions of the matrices involved in the computation of CCA coefficients and finally the dimensions of the matrices involved in the computation of the i-CCA coefficients. As an important remark, we have used the truncated SVD for the incremental CCA, that consists in taking only the most important eigenvalues of the data matrices. Let the SVD of a matrix be $(\mathbf{A})_{\{d\times m\}} = (\mathbf{U})_{\{d\times k\}}(\mathbf{D})_{\{k\times k\}}(\mathbf{V}^T)_{\{k\times m\}}$. We have taken only the biggest k eigenvalues and the corresponding column vectors of the $\mathbf{U}$ and $\mathbf{V}$ matrices, being $k \leq \min(d, m)$. Although this is an approximation of the original data matrices, we have seen that it was a more robust approach especially because of the matrix inversion that must be performed for the CCA coefficients computation.

B.1 Data matrices

Lets the data matrices be written as:

$$(\mathbf{A}_1)_{\{d\times m\}} = (\mathbf{U}_{A_1})_{\{d\times k\}}(\mathbf{D}_{A_1})_{\{k\times k\}}(\mathbf{V}_{A_1}^T)_{\{k\times m\}} \tag{B.1}$$

and

$$(\mathbf{A}_2)_{\{p\times m\}} = (\mathbf{U}_{A_2})_{\{p\times l\}}(\mathbf{D}_{A_2})_{\{l\times l\}}(\mathbf{V}_{A_2}^T)_{\{l\times m\}}. \tag{B.2}$$

As we have said before, we have retained the k more representatives eigenvectors for $\mathbf{A}_1$ and the l more representatives eigenvectors for $\mathbf{A}_2$, with $k \leq \min(d, m)$ and $l \leq \min(p, m)$. It is from these matrices that we will obtain the CCA coefficients.

B.2 CCA coefficients.

For the computation of the CCA coefficients the first thing we have to do is the multiplication of the matrices $\mathbf{V}_{A_1}$ and $\mathbf{V}_{A_2}$ and obtain the SVD of this product.

$$(\mathbf{V}_{A_1}^T)_{\{k\times m\}}(\mathbf{V}_{A_2})_{\{m\times l\}} = (\mathbf{U})_{\{k\times j\}}(\mathbf{D})_{\{j\times j\}}(\mathbf{V}^T)_{\{j\times l\}} \tag{B.3}$$

with $j \leqq \min(k, l)$.

With these matrices and those containing the data we can thus obtain the CCA coefficients encoded in the $\mathbf{G}$ matrix given by:

$$(\mathbf{G})_{\{p\times d\}} = (\mathbf{U}_{A_2})_{\{p\times l\}}(\mathbf{D}_{A_2})_{\{l\times l\}}(\mathbf{V})_{\{l\times j\}}(\mathbf{U}^T)_{\{j\times k\}}(\mathbf{D}_{A_1}^{-1})_{\{k\times k\}}(\mathbf{U}_{A_1}^T)_{\{k\times d\}} \tag{B.4}$$

Now, based on these results, we will present the matrices involved in the computation of the incremental CCA.

B.3 Incremental CCA coefficients.

In order to perform the computation of the incremental CCA, we will use the algorithm presented in 7.1.3.

1. Every n new images we will form two new matrices

   $$(\mathbf{B}_1)_{\{d\times n\}} \tag{B.5}$$

   and

   $$(\mathbf{B}_2)_{\{p\times n\}} \tag{B.6}$$

   containing respectively the new images $(\mathbf{I_i})_{\{d\times 1\}}$ and the corresponding perturbations $(\Delta\mathbf{b}_i)_{\{p\times 1\}}$.

2. We compute the mean vectors

   $$(\bar{\mathbf{I}}_{\mathbf{B_1}})_{\{d\times 1\}} = \frac{1}{n}\sum_{i=m+1}^{m+n}(\mathbf{I}_i)_{\{d\times 1\}}, \tag{B.7}$$

   $$(\bar{\mathbf{I}}_{\mathbf{A_1}}^{(\mathbf{new})})_{\{d\times 1\}} = \frac{n}{m+n}(\bar{\mathbf{I}}_{\mathbf{B_1}})_{\{d\times 1\}} + \frac{m}{m+n}(\bar{\mathbf{I}}_{\mathbf{A_1}})_{\{d\times 1\}}, \tag{B.8}$$

   $$(\bar{\mathbf{\Delta}}\mathbf{b}_{\mathbf{B_2}})_{\{p\times 1\}} = \frac{1}{n}\sum_{i=m+1}^{m+n}(\Delta\mathbf{b}_i)_{\{p\times 1\}}, \tag{B.9}$$

   and

$$(\bar{\mathbf{\Delta b}}^{(\mathbf{new})}_{\mathbf{A_2}})_{\{p\times 1\}} = \frac{n}{m+n}(\bar{\mathbf{\Delta b}}_{\mathbf{B_2}})_{\{p\times 1\}} + \frac{m}{m+n}(\bar{\mathbf{\Delta b}}_{\mathbf{A_2}})_{\{p\times 1\}}. \tag{B.10}$$

3. Then we form the matrix

$$(\hat{\mathbf{B}}_1)_{\{d\times n+1\}} = \left[(\mathbf{I}_{m+1}-\bar{\mathbf{I}}_{\mathbf{B_1}})_{\{d\times 1\}}\cdots(\mathbf{I}_{m+n}-\bar{\mathbf{I}}_{\mathbf{B_1}})_{\{d\times 1\}} \;\; \sqrt{\tfrac{mn}{m+n}}(\bar{\mathbf{I}}_{\mathbf{B_1}}-\bar{\mathbf{I}}^{(\mathbf{old})}_{\mathbf{A_1}})_{\{d\times 1\}}\right] \tag{B.11}$$

and

$$(\hat{\mathbf{B}}_2)_{\{p\times n+1\}} = \left[(\Delta\mathbf{b}_{m+1}-\bar{\Delta\mathbf{b}}_{B_2})_{\{p\times 1\}}\cdots(\Delta\mathbf{b}_{m+n}-\bar{\Delta\mathbf{b}}_{B_2})_{\{p\times 1\}} \;\; \sqrt{\tfrac{mo}{m+n}}(\bar{\mathbf{\Delta b}}_{\mathbf{B_2}}-\bar{\mathbf{\Delta b}}^{(\mathbf{old})}_{\mathbf{A_2}})_{\{p\times 1\}}\right] \tag{B.12}$$

4. Compute the matrices

$$(\mathbf{H_1})_{\{d\times n+1\}} = ((\mathbf{I})_{\{d\times d\}} - (\mathbf{U}_{A_1})_{\{d\times k\}}(\mathbf{U}^T_{A_1})_{\{k\times d\}})(\hat{\mathbf{B}})_{\{d\times n+1\}} \tag{B.13}$$

and

$$(\mathbf{H_2})_{\{p\times n+1\}} = ((\mathbf{I})_{\{p\times p\}} - (\mathbf{U}_{A_2})_{\{p\times l\}}(\mathbf{U}^T_{A_2})_{\{l\times p\}})(\hat{\mathbf{B}})_{\{p\times n+1\}} \tag{B.14}$$

and their SVD decomposition

$$(\mathbf{H}_1)_{\{d\times n+1\}} = (\mathbf{U}_{H_1})_{\{d\times s\}}(\mathbf{D}_{H_1})_{\{s\times s\}}(\mathbf{V}^T_{H_1})_{\{s\times n+1\}} \tag{B.15}$$

and

$$(\mathbf{H}_2)_{\{p\times n+1\}} = (\mathbf{U}_{H_2})_{\{p\times t\}}(\mathbf{D}_{H_2})_{\{t\times t\}}(\mathbf{V}^T_{H_2})_{\{t\times n+1\}}. \tag{B.16}$$

5. Using

$$(\mathbf{L_1})_{\{k\times n+1\}} = (\mathbf{U_{A_1}}^T)_{\{k\times d\}}(\hat{\mathbf{B}}_{\mathbf{1}})_{\{d\times n+1\}} \tag{B.17}$$

and

$$(\mathbf{L_2})_{\{l\times n+1\}} = (\mathbf{U_{A_2}}^T)_{\{l\times p\}}(\hat{\mathbf{B}}_{\mathbf{2}})_{\{p\times n+1\}} \tag{B.18}$$

we form the matrices

$$(\mathbf{R}_1)_{\{k+s\times k+n+1\}} = \begin{bmatrix} (\mathbf{D_{A_1}})_{\{k\times k\}} & (\mathbf{L_1})_{\{k\times n+1\}} \\ (\mathbf{0})_{\{s\times k\}} & (\mathbf{U}^T_{H_1})_{\{s\times d\}}(\mathbf{H_1})_{\{d\times n+1\}} \end{bmatrix} \tag{B.19}$$

and

$$(\mathbf{R}_2)_{\{l+t\times l+n+1\}} = \begin{bmatrix} (\mathbf{D_{A_2}})_{\{l\times l\}} & (\mathbf{L_2})_{\{l\times n+1\}} \\ (\mathbf{0})_{\{t\times l\}} & (\mathbf{U}^T_{H_2})_{\{t\times p\}}(\mathbf{H_2})_{\{p\times n+1\}} \end{bmatrix} \tag{B.20}$$

6. Compute the SVD decomposition of

$$(\mathbf{R_1})_{\{k+s\times k+n+1\}} = (\mathbf{U}_{R_1})_{\{k+s\times e\}}(\mathbf{D}_{R_1})_{\{e\times e\}}(\mathbf{V}_{R_1}^T)_{\{e\times k+n+1\}} \tag{B.21}$$

and

$$(\mathbf{R_2})_{\{l+t\times l+n+1\}} = (\mathbf{U}_{R_2})_{\{l+t\times f\}}(\mathbf{D}_{R_2})_{\{f\times f\}}(\mathbf{V}_{R_2}^T)_{\{f\times l+n+1\}}. \tag{B.22}$$

7. Update the matrices

$$(\mathbf{U}_{\mathbf{A_1}}^{(\mathbf{new})})_{\{d\times e\}} = (\begin{bmatrix} (\mathbf{U}_{\mathbf{A_1}})_{\{d\times k\}} & (\mathbf{U}_{H_1})_{\{d\times s\}} \end{bmatrix}(\mathbf{U}_{R_1})_{\{k+s\times e\}}), \tag{B.23}$$

$$(\mathbf{D}_{\mathbf{A_1}}^{(\mathbf{new})})_{\{e\times e\}} = (\mathbf{D}_{R_1})_{\{e\times e\}}, \tag{B.24}$$

$$(\mathbf{U}_{\mathbf{A_2}}^{(\mathbf{new})})_{\{p\times f\}} = (\begin{bmatrix} (\mathbf{U}_{\mathbf{A_2}})_{\{p\times l\}} & (\mathbf{U}_{H_2})_{\{p\times t\}} \end{bmatrix}(\mathbf{U}_{R_2})_{\{l+t\times f\}}), \tag{B.25}$$

and

$$(\mathbf{D}_{\mathbf{A_2}}^{(\mathbf{new})})_{\{f\times f\}} = (\mathbf{D}_{R_2})_{\{f\times f\}}. \tag{B.26}$$

8. Compute the SVD of

$$\begin{aligned}(\mathbf{V}_{\mathbf{R_1}}^{\mathbf{T}})_{\{e\times k+o+1\}}&\begin{bmatrix} ({\mathbf{V}_{\mathbf{A_1}}}^T)_{\{k\times m\}}(\mathbf{V}_{\mathbf{A_2}})_{\{m\times l\}} & (\mathbf{0})_{\{k\times n+1\}} \\ (\mathbf{0})_{\{n+1\times l\}} & (\mathbf{I})_{\{n+1\times n+1\}} \end{bmatrix}(\mathbf{V}_{R_2})_{\{l+n+1\times f\}} \\ &= (\mathbf{U}^{(\mathbf{new})})_{\{e\times g\}}(\mathbf{D}^{(\mathbf{new})})_{\{g\times g\}}({\mathbf{V}^{(\mathbf{new})}}^T)_{\{g\times f\}}\end{aligned} \tag{B.27}$$

9. Finally we update the matrix

$$(\mathbf{G})_{\{p\times d\}}=(\mathbf{U}_{A_2}^{(new)})_{\{p\times f\}}(\mathbf{D}_{A_2}^{(new)})_{\{f\times f\}}(\mathbf{V}^{(new)})_{\{f\times g\}}(\mathbf{U}^{(new)T})_{\{g\times e\}}(\mathbf{D}_{A_1}^{(new)-1})_{\{e\times e\}}(\mathbf{U}_{A_1}^{(new)T})_{\{e\times d\}}. \tag{B.28}$$

B.4 Video sequences and C libraries.

Here we present the links to the video sequences and the OpenCV library used for the implementation and development of the algorithm.

- ⋆ The OpenCV library that can be found at:
  http : //www.intel.com/technology/computing/opencv/index.htm

- ⋆ To test the tracker we have used the video sequences used by Lacascia in [48]:
  $http://www.cs.bu.edu/groups/ivc/HeadTracking/$.
- ⋆ The annotated talking face video made available from the *FGnet Working Group*:
  $http://www-prima.inrialpes.fr/FGnet/data/01-TalkingFace/talking\_face.html$,
- ⋆ The video sequences from Buenaposada in [11]:
  http://www.dia.fi.upm.es/∼pcr/downloads.html.

Bibliography

[1] Ankur Agarwal and Bill Triggs. Recovering 3D human pose from monocular images. *IEEE Transactions on Pattern Analysis & Machine Intelligence*, 28(1):44–58, January 2006.

[2] Jörgen Ahlberg. Candide-3 – an updated parameterized face. Technical Report LiTH-ISY-R-2326, Linköping University, Department of Electrical Engineering, Sweden, January 2001.

[3] Jörgen Ahlberg. *Model-based Coding - Extraction, Coding, and Evaluation of Face Model Parameters.* PhD thesis, Linköping University, Department of Electrical Engineering, Sweden, 2002.

[4] A-Nasser Ansari and Mohamed Abdel-Mottaleb. 3D face modeling using two views and a generic face model with application to 3D face recognition. In *IEEE Conference on Advanced Video and Signal Based Surveillance*, pages 37–44, Miami, Florida, USA, 21-22 July 2003.

[5] Shai Avidan. Support vector tracking. In *IEEE Transactions on Pattern Analysis & Machine Intelligence*, volume 26, pages 1064–1072, August 2004.

[6] Francis R. Bach and Michael I. Jordan. A probabilistic interpretation of canonical correlation analysis. Technical Report 688, Department of Statistics, University of California, Berkeley, 2005.

[7] Csaba Beleznai, Thomas Schloegl, Bernd Wachmann, Horst Bischof, and Walter G. Kropatsch. Tracking multiple objects in complex scenes. In F. Leberl and F. Fraundorfer, editors, *Proceedings of the Workshop of the Austrian Association for Pattern Recognition*, volume 160, pages 175 – 182, Graz, September 2002. Austrian Computer Society.

[8] Magnus Borga. Canonical correlation a tutorial. Technical report, Linköping University, Sweden, January 2001.

[9] Magnus Borga, Tomas Landelius, and Hans Knutsson. A unified approach to PCA, PLS, MLR and CCA. Report LiTH-ISY-R-1992, Linköping University, SE-581 83 Linköping, Sweden, November 1997.

[10] Matthew Brand. Incremental singular value decomposition of uncertain data with missing values. In *Proceedings of the European Conference on Computer Vision*, volume 4, pages 707–720, Copenhagen, Denmark, 27 May–2 June 2002.

[11] José Miguel Buenaposada, Enrique Muñoz, and Luis Baumela. Efficiently estimating facial expression and illumination in appearance-based tracking. In *Proceedings of the British Machine Vision Conference*, volume 1, pages 57–67, Edinburgh, UK, September 2006.

[12] Andrea Cavallaro and Touradj Ebrahimi. Video object extraction based on adaptive background and statistical change detection. In Bernd Girod, Charles A. Bouman, and Eckehard G. Steinbach, editors, *Visual Communications and Image Processing*, volume 4310 of *Proceedings of SPIE*, pages 465–475, San Jose, California, USA, 24 January 2001.

[13] Andrea Cavallaro, Olivier Steiger, and Touradj Ebrahimi. Multiple video object tracking in complex scenes. In *Proceedings of the ACM International Conference on Multimedia*, pages 523–532, Juan les Pins, France, 1–6 December 2002.

[14] Andrea Cavallaro, Olivier Steiger, and Touradj Ebrahimi. Tracking video objects in cluttered background. *IEEE Transactions on Circuits and Systems for Video Technology*, 15(4):575– 584, April 2005.

[15] Marc Chaumont and Brice Beaumesnil. Robust and real-time 3D-face model extraction. In *Proceedings of the IEEE International Conference on Image Processing*, volume 3, pages 461–464, Genova, Italy, 11–14 September 2005.

[16] Hong Cheng, Nanning Zheng, and Tie Liu. An approach of road recognition based mean shift and feature clustering. In *Proceedings of the International Conference on Signal Processing*, volume 2, pages 1059–1062, Beijing, China, 26-30 August 2002.

[17] Yizong Cheng. Mean shift, mode seeking and clustering. *IEEE Transactions on Pattern Analysis & Machine Intelligence*, 17(8):790–799, August 1995.

[18] Haili Chui and Anand Rangarajan. A new algorithm for non-rigid point matching. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, volume 2, pages 44–51, Hilton Head, SC, USA, 13-15 June 2000.

[19] Dorin Comaniciu and Peter Meer. Robust analysis of feature spaces: color image segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 750–755, San Juan, Puerto Rico, 17-19 June 1997.

[20] Timothy F. Cootes. An introduction to active shape models. Technical report, Imaging Science and Biomedical Engineering Research Group, University of Manchester, 2001.

[21] Timothy F. Cootes, Gareth J. Edwards, and Christopher J. Taylor. Active appearance models. In *Proceeding of the European Conference on Computer Vision*, volume 2, pages 484–498, Freiburg, Germany, 2–6 June 1998.

[22] Timothy F. Cootes, Gareth J. Edwards, and Christopher J. Taylor. Active appearance models. *IEEE Transactions on Pattern Analysis & Machine Intelligence*, 23(6):681–685, June 2001.

[23] Timothy F. Cootes and Christopher J. Taylor. Statistical models of appearance for medical image analysis and computer vision. Technical report, Imaging Science and Biomedical Engineering, University of Manchester, 2001.

[24] David Cristinacce and Timothy F. Cootes. Feature detection and tracking with constrained local models. In *Proceeding of the British Machine Vision Conference*, volume 3, Edinburgh, UK, 4–7 September 2006.

[25] Franck Davoine and Fadi Dornaika. *Real-Time Vision for Human Computer Interaction*, chapter Head and Facial Animation Tracking using Appearance-Adaptive Models and Particle Filters, page 302. Springer Verlag New York, Inc., Secaucus, NJ, USA, 2005.

[26] Catherine Dehon, Peter Filzmoser, and Christophe Croux. Robust methods for canonical correlation analysis. In H.A.L. Kiers, J.P. Rasson, P.J.F. Groenen, and M. Schrader, editors, *Data Analysis, Classification, and Related Methods. Proceedings of the Conference on the International Federation of Classification Societies*, pages 321–326, University of Namur, Belgium, 11–14 July 2000. Springer-Verlag.

[27] Chris Ding and Jieping Ye. 2-dimensional singular value decomposition for 2d maps and images. In *Proceedings of the SIAM International Conference on Data Mining*, pages 32–43, Newport Beach, California, USA, 21–23 April 2005.

[28] Petar M. Djurić, Jayesh H. Kotecha, Jianqui Zhang, Yufei Huang, Tadesse Ghirmai, Mónica Bugallo, and Joaquín Miguez. Particle filtering. *Signal Processing Magazine, IEEE*, 20(5):19–38, September 2003.

[29] Réné Donner, Michael Reiter, Langs Georg, Philipp Pelsochek, and Horst Bischof. Fast active appearance model search using canonical correlation analysis. *IEEE Transactions on Pattern Analysis & Machine Intelligence*, 28(10):1690–1694, October 2006.

[30] Fadi Dornaika and Franck Davoine. Head and facial animation tracking using appearance-adaptive models and particle filters. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 153 – 163, Washington DC, USA, 27 June -2 July 2004.

[31] Fadi Dornaika and Franck Davoine. On appearance based face and facial action tracking. *IEEE Transactions on Circuits and Systems for Video Technology*, 16(9):1107– 1124, September 2006.

[32] Gareth J. Edwards, Christopher J. Taylor, and Timothy F. Cootes. Interpreting face images using active appearance models. In *Proceedings of the IEEE International Conference on Automatic Face and Gesture Recognition*, pages 300–305, Nara, Japan, 14-16 April 1998.

[33] Paul Ekman and Wallace V. Friesen. *The Manual of Facial Action Coding System*. Consulting Psychologist Press, Palo Alto, CA, USA, 1977.

[34] Nicolas Eveno, Alice Caplier, and Pierre-Yves Coulon. New color transformation for lips segmentation. In *IEEE Workshop on Multimedia Signal Processing*, pages 3–8, Cannes, France, 2–5 October 2001.

[35] Keinosuke Fukunaga and L. Hostetler. The estimation of the gradient of a density function, with applications in pattern recognition. *IEEE Transactions on Information Theory*, 21(1):32–40, January 1975.

[36] Timothy F. Gee and Russell M. Mersereau. Model-based face tracking for dense motion field estimation. In *Applied Imagery Pattern Recognition Workshop*, pages 149–153, Washington, DC, USA, 10-12 October 2001.

[37] David R. Hardoon and John Shawe-Taylor. KCCA for different level precision in content-based image retrieval. In *Proceedings of International Workshop on Content-Based Multimedia Indexing*, IRISA, Rennes, France, 22–24 September 2003.

[38] David R. Hardoon, John Shawe-Taylor, and Ola Friman. KCCA for fMRI analysis. In *Proceedings of Medical Image Understanding and Analysis*, London,UK, 23–24 September 2004.

[39] David R. Hardoon, Sandor Szedmak, and John Shawe-Taylor. Canonical correlation analysis; an overview with application to learning methods. Technical Report CSD-TR-03-02, Dept. of Computer Science, May 2003.

[40] Chris J. Harris and Mike Stephens. A combined corner and edge detector. In *Proceedings of The Fourth Alvey Vision Conference*, pages 147–151, Manchester, UK, August-September 1988.

[41] Paul S. Heckbert. Survey of texture mapping. *Computer Graphics and Applications, IEEE*, 6(11):56–67, November 1986.

[42] Bernd Heisele. Motion-based object detection and tracking in color image sequences. In *Proceedings of the Asian Conference on Computer Vision*, pages 1028–1033, Taipei, Taiwan, 8–11 January 2000.

[43] Su-Yun Huang, Mei-Hsien Lee, and Chuhsing Kate Hsiao. Kernel canonical correlation analysis and its applications to nonlinear measures of association and test of independence. Technical report, Institute of Statistical Science, Academia Sinica, Taiwan, 2006.

[44] Michal Irani, Benny Rousso, and Shmuel Peleg. Computing occluding and transparent motions. *International Journal of Computer Vision*, 12(1):5–16, 1994.

[45] Michael Isard and Andrew Blake. Contour tracking by stochastic propagation of conditional density. In *Proceedings of the European Conference on Computer Vision*, pages 343–356, Cambridge,UK, 15–18 April 1996.

[46] Michael Isard and John MacCormick. Bramble: a bayesian multiple-blob tracker. In J. MacCormick, editor, *Proceedings of the IEEE International Conference on Computer Vision*, volume 2, pages 34–41, Vancouver, BC, Canada, 9–12 July 2001.

[47] Jinman Kang, Isaac Cohen, and Gerard Medioni. Persistent objects tracking across multiple non overlapping cameras. In *Proceedings of the IEEE Workshop on Motion and Video Computing WACV/MOTIONS*, volume 2, pages 112–119, Breckenridge, Colorado, USA, 5–7 January 2005.

[48] Marco La Cascia, Stan Sclaroff, and Vassilis Athitsos. Fast, reliable head tracking under varying illumination: an approach based on registration of texture-mapped 3D models. *IEEE Transactions on Pattern Analysis & Machine Intelligence*, 22(2):322–336, April 2000.

[49] Lieven De Lathauwer, Bart DeMoor, and Joos Vandewalle. A multilinear singular value decomposition. *SIAM Journal on Matrix Analysis and Applications*, 21(4):1253–1278, 2000.

[50] Mun Wai Lee, Isaac Cohen, and Soon Ki Jung. Particle filter with analytical inference for human body tracking. In *Proceedings of the Workshop on Motion and Video Computing*, pages 159–165, Orlando, Florida, USA, 3–4 December 2002.

[51] Vincent Lepetit and Pascal Fua. Towards Recognizing Feature Points using Classification Trees. Technical Report IC/2004/74, Swiss Federal Institute of Technology, 1015 Lausanne, Switzerland, 2004.

[52] Vincent Lepetit and Pascal Fua. *Monocular Model-Based 3D Tracking of Rigid Objects: A Survey*, volume 1. Now Publishers Inc., 2005.

[53] Vincent Lepetit, Pascal Lagger, and Pascal Fua. Randomized trees for real-time keypoint recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, volume 2, pages 775–781, San Diego, CA, USA, 20-25 June 2005.

[54] Vincent Lepetit, Julien Pilet, and Pascal Fua. Point matching as a classification problem for fast and robust object pose estimation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, volume 2, pages 244–250, Washington, DC, USA, 27 June -2 July 2004.

[55] Rainer Lienhart and Jochen Maydt. An extended set of Haar-like features for rapid object detection. In *Proceedings of the IEEE International Conference on Image Processing*, volume 1, pages 900–903, Rochester, New York, USA, 22-25 September 2002.

[56] Marc Lievin and Franck Luthon. Nonlinear color space and spatiotemporal mrf for hierarchical segmentation of face features in video. *IEEE Transactions on Image Processing*, 13(1):63–71, January 2004.

[57] Jongwoo Lim, David A. Ross, Ruei-Sung Lin, and Ming-Hsuan Yang. Incremental learning for visual tracking. In Lawrence K. Saul, Yair Weiss, and Léon Bottou, editors, *Advances in Neural Information Processing Systems 17*, pages 793–800. MIT Press, Cambridge, MA, USA, 2005.

[58] David G. Lowe. Object recognition from local scale-invariant features. In *Proceedings of the IEEE International Conference on Computer Vision*, volume 2, pages 1150–1157, Kerkyra, Corfu, Greece, 20–25 September 1999.

[59] Stephen J. McKenna, Shaogang Gong, and J. J. Collins. Face Tracking and Pose Representation. In R. B. Fisher and E. Trucco, editors, *British Machine Vision Conference*, volume 2, pages 755–764, Edinburgh, September 1996. BMVA.

[60] Thomas Melzer, Michael Reiter, and Horst Bischof. Appearance models based on kernel canonical correlation analysis. *Pattern Recognition*, 36(9):1961–1971, September 2003.

[61] Antoine Monnet, Anurag Mittal, Nikos Paragios, and Visvanathan Ramesh. Background modeling and subtraction of dynamic scenes. In *Proceedings of the IEEE International Conference on Computer Vision*, volume 2, pages 1305–1312, Nice, France, 13-16 October 2003.

[62] Kenji Okuma, Ali Taleghani, Nando de Freitas, James J. Little, and David G. Lowe. A boosted particle filter: Multitarget detection and tracking. In *Proceedings of the European Conference on Computer Vision*, volume 3021, pages 28–39, Prague, 11–14 May 2004.

[63] Frederick I. Parke. Computer generated animation of faces. In *Proceedings of the ACM annual conference*, pages 451–457, Boston, Massachusetts, USA, 1972. ACM, New York, NY, USA.

[64] Frederick I. Parke. Parameterized models for facial animation. *IEEE Computer Graphics and Applications*, 2(9):61–68, November 1982.

[65] Patrick Pérez, Carine Hue, Jaco Vermaak, and Michel Gangnet. Color-based probabilistic tracking. In *Proceedings of the European Conference on Computer Vision*, volume 2350, pages 661 – 675, Copenhagen, Denmark, 27 May-2 June 2002. Springer-Verlag London, UK.

[66] Patrick Pérez, Jaco Vermaak, and Andrew Blake. Data fusion for visual tracking with particles. *Proceedings of the IEEE*, 92(3):495–513, March 2004.

[67] Julien Pilet, Vincent Lepetit, and Pascal Fua. Real-time nonrigid surface detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, volume 1, pages 822–828, San Diego, CA, USA, 20-25 June 2005.

[68] Sami Romdhani. *Face Image Analysis using a Multiple Feature Fitting Strategy*. PhD thesis, University of Basel. Computer Science Department, January 2005.

[69] Sami Romdhani, Volker Blanz, Curzio Basso, and Thomas Vetter. *Morphable Models of Faces*, chapter 10, pages 217–246. Springer-Verlag, January 2004.

[70] Sami Romdhani, Volker Blanz, and Thomas Vetter. Face identification by fitting a 3D morphable model using linear shape and texture error functions. In *Proceedings of the European Conference on Computer Vision*, volume 4, pages 3–19, Copenhagen, Denmark, 27 May–2 June 2002.

[71] Paul L. Rosin. Thresholding for change detection. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 274–279, Bombay, India, 4-7 January 1998.

[72] David Ross, Jongwoo Lim, Ruei-Sung Lin, and Ming-Hsuan Yang. Incremental learning for robust visual tracking. *International Journal of Computer Vision*, 77(1–3):125–141, May 2007.

[73] Mikael Rydfalk. Candide, a parameterised face. Technical Report LiTH-ISY-I-866, Linköping University, Department of Electrical Engineering, Sweden, October 1987.

[74] Stan Sclaroff and John Isidoro. Active blobs. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1146–1153, Bombay, India, 4–7 January 1998.

[75] Ali Shahrokni, Franọis Fleuret, and Pascal Fua. Classifier-based contour tracking for rigid and deformable objects. In *Proceedings of the British Machine Vision Conference*, Oxford, UK, 5–8 September 2005.

[76] Danijel Skocaj and Ales Leonardis. Appearance based localization using CCA. In *Proceedings of the Computer Vision Winter Workshop*, pages 205–214, Piran, Slovenia, February 2004.

[77] Xuefeng Song and Ram Nevatia. Combined face-body tracking in indoor environment. In *Proceedings of the International Conference on Pattern Recognition*, volume 4, pages 159–162, Cambridge, UK, 23–26 August 2004.

[78] Xuefeng Song and Ram Nevatia. A model-based vehicle segmentation method for tracking. In *Proceedings of the IEEE International Conference on Computer Vision*, volume 2, pages 1124–1131Vol.2, Beijing, China, 17-21 October 2005.

[79] Chris Stauffer and W.E.L Grimson. Adaptive background mixture models for real-time tracking. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, volume 2, pages 245–252, Fort Collins, CO, USA, June 1999.

[80] Mikkel B. Stegmann, Bjarne K. Ersbøll, and Rasmus Larsen. FAME – a flexible appearance modelling environment. *IEEE Transactions on Medical Imaging*, 22(10):1319–1331, October 2003.

[81] Marten Strömberg. The candide software package, release 1. Technical report, Linköpings Universitet, Sweden, August 1994.

[82] Hai Tao, Harpreet S. Sawhney, and Rakesh Kumar. A sampling algorithm for tracking multiple objects. In *ICCV '99: Proceedings of the International Workshop on Vision Algorithms*, pages 53–68, Kerkyra, Corfu, Greece, 20–25 September 2000. Springer-Verlag London, UK.

[83] Benoît Tella and Marie-José Aldon. Interest points detection in color images. In *Proceedings of IAPR Workshop on Machine Vision and its Applications*, pages 550–553, Nara, Japan, 11–13 December 2002.

[84] Cor J. Veenman, Emile A. Hendriks, and Marcel J.T. Reinders. A fast and robust point tracking algorithm. In *Proceedings of the IEEE International Conference on Image Processing*, volume 3, pages 653–657, Chicago, Illinois, USA, 4–7 October 1998.

[85] Cor J. Veenman, Marcel J.T. Reinders, and Eric Backer. Resolving motion correspondence for densely moving points. *IEEE Transactions on Pattern Analysis & Machine Intelligence*, 23(1):54–72, January 2001.

[86] Paul Viola and Michael Jones. Rapid object detection using a boosted cascade of simple features. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, volume 1, pages 511–518, Hawaii, USA, 11–13 December 2001.

[87] David Weenink. Canonical correlation analysis. In *Proceedings of the Institute of Phonetic Sciences of the University of Amsterdam*, volume 25, pages 81–99, Netherlands, 2003.

[88] Oliver Williams, Andrew Blake, and Roberto Cipolla. A sparse probabilistic learning algorithm for real-time tracking. In *Proceedings of the IEEE International Conference on Computer Vision*, volume 1, pages 353–360, Nice, France, 13–16 October 2003.

[89] Oliver Williams, Andrew Blake, and Roberto Cipolla. Sparse bayesian learning for efficient visual tracking. *IEEE Transactions on Pattern Analysis & Machine Intelligence*, 27(8):1292–1304, August 2005.

[90] Lambert E. Wixson and Andrea Selinger. Classifying moving objects as rigid or non-rigid. In *DARPA Image Understanding Workshop*, volume 1, pages 341–347, Monterey, CA, November 1998.

[91] Jing Xiao, Simon Baker, Iain Matthews, and Takeo Kanade. Real-time combined 2d+3d active appearance models. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, volume 2, pages 535 – 542, Washington, DC, USA, 27 June–2 July 2004.

[92] Ming Xu and Tim Ellis. Illumination-invariant motion detection using colour mixture models. In *Proceedings of the British Machine Vision Conference*, pages 163–172, Manchester, UK, 10–13 September 2001.

[93] Jian Yang, D. Zhang, A.F. Frangi, and Jing-yu Yang. Two-dimensional pca: a new approach to appearance-based face representation and recognition. *Transactions on Pattern Analysis & Machine Intelligence*, 26(1):131–137, May 2004.

[94] Jie Yang, Weier Lu, and Alex Waibel. Skin-color modeling and adaptation. Technical Report CMU-CS-97-146, Carnegie Mellon University, Pittsburgh, PA 15213, USA, May 1997.

[95] Jie Yang, Weier Lu, and Alex Waibel. Skin-color modeling and adaptation. In *Proceedings of the Asian Conference on Computer Vision*, volume 2, pages 687–694, Hong Kong, China, 8–10 January 1998.

[96] José Alonso Ybáñez Zepeda, Franck Davoine, and Maurice Charbit. Face tracking using canonical correlation analysis. In *proceedings of the International Conference on Computer Vision Theory and Applications*, pages 396–402, Barcelona, Spain, March 2007.

[97] José Alonso Ybáñez Zepeda, Franck Davoine, and Maurice Charbit. A linear estimation method for 3d pose and facial animation tracking. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–7, Minneapolis, Minnesota, USA, 18–23 June 2007.

[98] José Alonso Ybáñez Zepeda, Franck Davoine, and Maurice Charbit. Linear tracking of pose and facial features. In *Proceedings of the IAPR Conference on Machine Vision Applications*, pages 182–185, Tokyo, Japan, 20–22 May 2007.

[99] José Alonso Ybáñez Zepeda, Franck Davoine, and Maurice Charbit. Local or global 3d face and facial feature tracker. In *Proceedings of the IEEE International Conference on Image Processing*, volume 1, pages 505–508, San Antonio, Texas, USA, 16–19 September 2007.

[100] Alper Yilmaz, Omar Javed, and Mubarak Shah. Object tracking: A survey. *ACM Computing Surveys*, 38(4):1–45, December 2006.

[101] Tao Zhao and Ram Nevatia. Tracking multiple humans in complex situations. *IEEE Transactions on Pattern Analysis & Machine Intelligence*, 26(9):1208–1221, September 2004.

[102] Tao Zhao and Ram Nevatia. Tracking multiple humans in crowded environment. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, volume 2, pages 406–413, Washington, DC, USA, 27 June–2 July 2004.

[103] Wenming Zheng, Xiaoyan Zhou, Cairong Zou, and Li Zhao. Facial expression recognition using kernel canonical correlation analysis (KCCA). *IEEE Transactions on Neural Networks*, 17(1):233–238, January 2006.

[104] Xue Zhou, Weiming Hu, and Xi Li. An adaptive shape subspace model for level set-based object tracking. In *Subspace 2007. Workshop on Asian Conference on Computer Vision*, pages 9–16, Tokio, Japan, 19 November 2007.