Real-time face swapping based on head posture using particle filter towards the understanding of infant self-recognition

22 February 2010

Osaka University
Graduate School of Engineering
Department of Adaptive Machine Systems
Emergent Robotics Laboratory
Asada Laboratory

Nguyen Thi Sao Mai
Asada Laboratory

Abstract

As human beings, we take self-recognition for granted, but this seemingly simple ability represents one of the most complex mysteries of cognitive science. Indeed, even though human adults immediately recognise their image in the mirror as themselves', a dog or a cat would treat it as another animal to be played with or confronted. Even for humans, self-recognition is not innate, and studies on self-recognition in infants still need to shed light on the mechanisms of its developmental process. From this perspective, we designed an experiment for young infants, to study the preference of children between contingency and familiarity factors in self-consciousness.

For that purpose we developed a complete system that realises face replacement in videos. Our system is based on a 3D head posture estimation with a sparse-template based particle filter. So as not to disturb the behaviour of the subject or change the naturalness of their appearance for the purpose of our recognition experiment, our solution does not require any special installation other than a camera, and most of all, does not require any object to be attached to the subject. Our non-constraint real-time system is a large scale integration of existing face-tracking, eye direction estimation systems, doubled with an original face-swapping in video. Our real-time visual tracker combines sparse-template-based particle filter and parallel processing to obtain a real-time system. Indeed, the use of particle filter parameters that take into account the velocity increases the robustness of the tracker. Moreover, the design of the instruction pipeline to optimise parallel processing had enabled the decrease of the total delay of the face tracker to 66ms. The precision and the real-time tracking of the system also enables to detect the gaze direction of the subject. The novelty of the work also lies in the the lightweight of the initialisation phase. The head tracker needs only a frontal face picture of the subject, and uses as 3D model of the face, a simple ellipsoid.

Keywords: *self-consciousness, self-recognition, 3D visual tracker, face replacement in video, particle filtering, appearance based, real-time system, GPGPU*

Contents

List of Figures

Chapter 1

Introduction

1.1 Motivation

I know that I exist. The question is, what is this 'I' that I know
René Descartes, *Meditations on First Philosophy, Second Meditation, 1641*

We define "self-consciousness", also described as "self-awareness" as the awareness of one's own cognition, or in other words, the recognition by the subject of his own acts or affections.[16] Human infants' self-consciousness is very different from human adults'. Children's self-awareness builds up during his development through different levels[22] (fig 1.1) . Indeed human neonates can detect contingency between their movements and what they see. From 2 months to 18 months is the second level of self-exploration, that includes 2 stages. By 2 months, babies know their body position in relation to other objects and people. And by 5 months, they can discriminate their own image from the image of another infant. Still, it is only at around 18 months that they can recognise the image in the mirror as themselvesf, and pass the rouge mirror test. First set up by Amsterdam,[23] this test is widely considered to assess whether children have acquired an adult-like level of self-consciousness. The experimenter marks the subject with a dye spot, unbeknown to him, not visible directly but only through a mirror. Only children above 18 months will touch and remove the mark. The period from 6 to 18 months of age is therefore a decisive developmental stage.

From a behavioural point of view, self-recognition is based on familiarity and contingency. Sanefuji et al.[24] have shown that infants display a preference for familiar faces, while Bahrick et al.[25] have shown that discrimination is also based on contingency of actions, that children are more sensitive to actions that occur at the same time as their own movements than delayed actions. Children are sensitive to both familiarity and contingency, and we propose to investigate which they prefer. For this purpose, we designed an experiment to see whether they would prefer contingent videos or videos with familiar faces. The subject is placed in front of a screen that shows videos (fig 1.2) :

- video of the subject live

Fig. 1.1 The main developmental stages in self-consciousness until children can pass the rouge mirror test according to Rochat

- video of the subject with a delay
- video of the subject live, but with face-swap: his face would be replaced by some one else's face. The video is therefore that of someone else imitating live what the subject does.
- video of the subject with a delay and with face swap. The video is therefore that of someone else imitating what the subject does with a delay.

Fig. 1.2 Setting of the familiarity-contingency experiment

For the purpose of the experiment, we needed to develop a face-swapper, that does not require special installation or attachment to the subject that would risk to introduce unnaturalness to the appearance. A non-constraint face-swapper is thus required.

A longer-term motivation for developing a face-swapper in videos is to build an objective and systematic imitation system based on face changes. Children with Autistic Spectrum Disorders (ASD) have impaired social interaction from an early stage in their development.[15] Several studies have suggested that imitating children with ASD is effective in facilitating their social behaviours[18],[17] These studies have until now been using the caregiver as imitator

of the child to imitate his or her movements. Therefore the imitation is not fully accurate and subject to subjective re interpretation by the caregiver. We would need a more objective and systematic imitator to evaluate the change of behaviour of children with ASD.

A real-time head tracker where no special material is required to study autistic children's attention system and behaviour during social interaction. Indeed, children with Autism Spectrum Disorder (ASD) have different attention patterns to non-ASD children in their face scanning strategy[20] and in joint attention.[19] This difficulty has been coined as the "weak central coherence theory" by U. Frith.[15] For that purpose ,tracking children's head direction to infer their attention point during social interaction becomes essential. However autistic people can be obsessively sensitive to particular details and objects.[15] The use of traditional tracking systems like motion capture for observing their natural behaviour and tracking their movements often turn out to disturb their behaviour and destabilise them. A non-constraint method is thus needed.

1.2 Constraints chosen

The constraints we impose on this tracker are:

- our system should need no special installation or tracking objects to be attached to the subject for the reasons mentioned in the previous section
- it must be automatic: we require a face-swapper that automatically replaces the subject's face with someone else's with a minimum amount of data and calibration. As children, especially autistic children do not easily comply with calibration constraints, we need to have the face-tracking and face-swapping exploitable with a minimum of calibration. Our system is very light in that sense, since it only requires the front face pictures of the subjects to calibrate.
- real-time. To study contingency between his own movements and the video that he is shown, the delay between reality and the screen output video must be minimised.
- it must be robust against rapid movements: the tracker needs to follow the head movements of the subject, especially the head turning from one side of the screen to the other. Therefore rotation around the vertical axis needs special attention.
- it must be robust to partial occlusion: children often bring their hands or toys to their face or mouth. Therefore our face tracking system must resist partial occlusion.

Besides, we evaluated the precision needed of the tracking for a realistic swapping, with regards with position error. For that, we used a frontal face image where we defined the face region. The set of images (fig. 1.3 was produced by vertical and horizontal translation, then blur of the edges, as the face-swapping program does automatically. This set of images

show that in the first approximation, translations of less than 10 pixels appear completely realistic, whereas translations of more than 15 pixels are noticed as strange. Therefore, the tracker should have a precision of 10 pixels to be used to build a face-swapper. If we related this value to the size of the face given by the distance between the 2 extremums of the eyes, measured here as 130 pixels, the error of the tracker should be less than 10%.

Fig. 1.3 Set of images used to evaluate the precision needed for a realistic swapping. The images have been obtained by translation of the face by the indicated number of pixels horizontally (Tx) or vertically (Ty). The translations smaller than 15 pixels still look realistic, whereas the translations of more than 15 pixels begin to look strange.

1.3 General presentation

The overall system we aim for therefore includes a face tracker, an eye direction detector, and a face swapper. The whole system is built around the 3D face tracker, and needs only

a frontal face picture of the subject (fig. 1.4). Once the face position and orientation is detected, the eye image can be extracted and the eye position estimated. Likewise, once the face posture is known, we can superimpose a replacement face of a subject B on the face of the current subject A. The replacement faces are prepared beforehand and are tagged images of the face of another subject B, but have been obtained automatically from the same face tracker. Therefore our whole system is per se a very lightweight ensemble that needs for first input only front face pictures of two subjects.

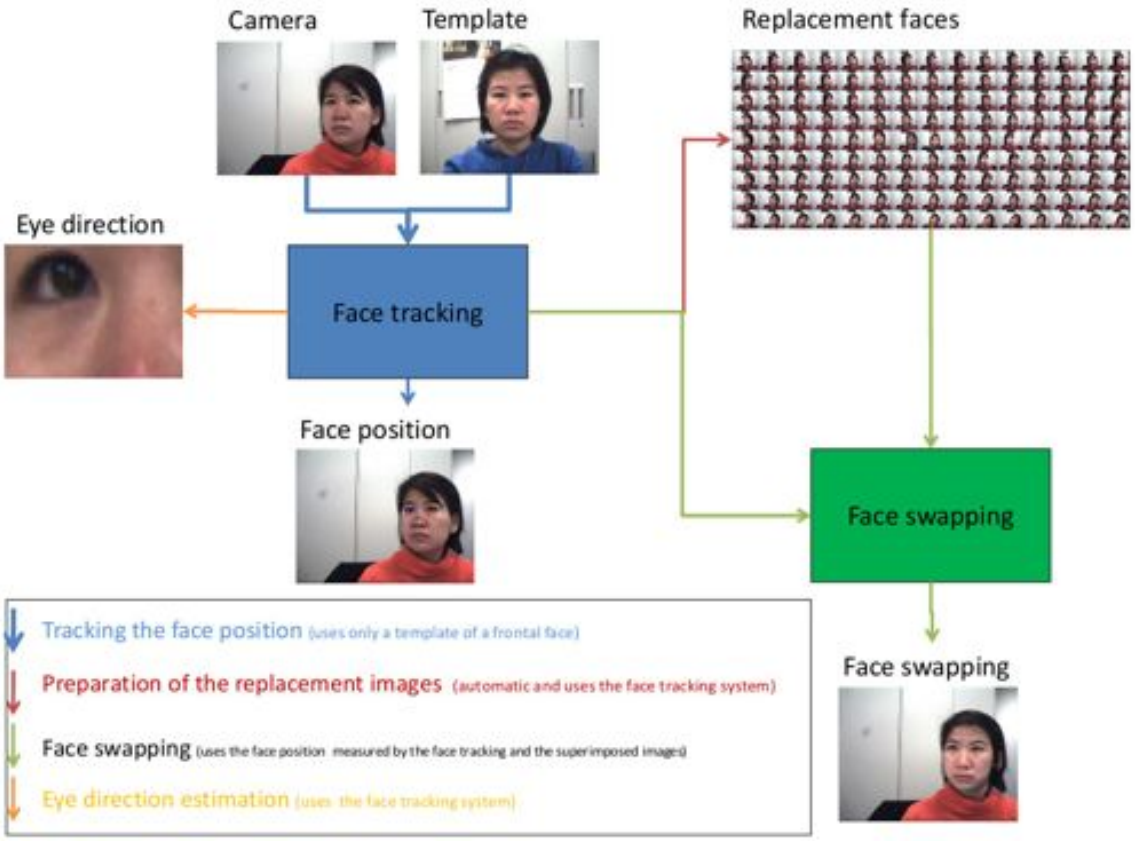

Fig. 1.4 The overall system is composed of a face tracker, an eye direction detector, and a face swapper

Chapter 2

Related work

Face replacement requires first the detection of the position and orientation of the face in the input image. Gaze movement means shift of the gaze direction but is always preceded by a head movement. The estimation of the head posture is therefore also a first estimation of the gaze direction.

2.1 3D visual tracker

Several kinds of commercial products exist to detect a person's had position and orientation, such as magnetic sensors and link mechanisms. Unfortunately such devices constraint the user's movements, and reduce the naturalness of his or her behaviour because of their discomfort. Likewise, as we said before, motion capture systems can be disturbing for children with ASD, or else give a strange appearance to the subject in the mirrored video. We therefore have to opt for a non intrusive method. FaceAPi[2)] is a non-invasive commercial product for face tracking. It tracks efficiently with head rotations covering a wide range, and respects the real-time constraint. Nevertheless, as a commercial product, there are some limitations due to needed information being inaccessible, in order to adapt our system to our use for children and extend it to a face-swapper. Particularly, faceAPI tracks 3 points on each eyebrow and 8 points around the lips. While the adults' eyebrows are distinct, those of babies under 18 months are less distinctive, and could lead to poor tracking. Besides, we intend to extend our system to track heads of infants during social interaction. Our proposed system is not limited to tracking frontal face only, but is easily extended to track the side of the head too.

Many researchers have proposed methods for real-time tracking. Matsubara et al[?] have reported a study on sparse template matching for object tracking, The key idea of the method is to reduce the calculation cost by introducing sparse templates, and choosing feature points matching both the parallel and the condensation algorithm. This method proved efficient in tracking faces for several image sequences. More complex algorithms involving a robust non-parametric technique for climbing density gradient to find the mode (peak) of proba-

bility distribution called the mean shift algorithm. Bradski et al[6] applied the mean shift algorithm to find the mode of a colour distribution with a video sequence. The modified algorithm called CAMSHIFT, was developed into a 4 degree of freedom colour object tracker and applied to flesh-tone-based face tracking. CAMSHIFT was shown to handle irregular object motion, image noise distractors and occlusion. However, the distance to the camera estimation is subject to noise and spurious values. Besides, CAMSHIFT only detects four (position and roll) of the six of freedom, and roll is the least useful control variable since it is the least "natural" head movement. Matsumoto et al.[1] proposes an estimation method of the 6-DOF motion of the face by means of a single camera and a 3D facial model.The method relies on the decomposition of 2D motion field of facial features into the linear combination of predicted unit motion vector fields generated from the 3D facial model to achieve a real-time estimation of the facial position and orientation The use of a single camera instead of a stereo camera enables a larger freedom for camera placement, and especially avoids difficult calibration of the stereo camera pair. But the technique proposed here requires a heavy 3D model of a head that is difficult to generate and might limit the tracking to rigid objects. Thus, only unexpressive and rigid faces are efficiently tracked. However, the most important issue is to generate the 3D model, which requires a stereo camera or a 3D scanner for building the 3D facial model.

To help with costly computation, there has recently strong interest from researchers and developers in exploiting the power of graphics hardware for general purpose computing (GPGPU). Computer Vision, the inverse of computer graphics, has been pointed out as well suited for GPGPU, and some have even studied the application of pure GPGPU techniques to particle filtering tracking algorithms.[5] Mateo Lozano et al.[8] presents a real-time visual tracker by stream processing that targets the position and 3D pose of faces in video sequences. The technique is based on stream processors for computations and sparse-template-based particle filtering. The real-time constraint has been obtained thanks to the use of a GPU and the NVIDIA CUDA technology. The advantage of this method is that though it can track the 3D pose of the face, it only requires a generic 3D model of the human face, that it personalises to each detected face through the proportions of the head and through rendering.

Our 3D head tracker also adopts this approach and uses an even simpler model for the face, namely it considers the head as a simple ellipsoid. Our tracker also relies on multi-processing and template-based particle filtering. Though, for our face model, we did not use a 3D model of human face, but a simple ellipsoid model. To increase the robustness of our tracker to movements, we redefine the parameters of the particle filters and increase its number.

However, the early assumptions that the head direction was a reasonable estimation of the direction of the child's focus of attention proved unsatisfying, so we needed to add the detection of the gaze attention to the system.

2.2 Eye direction

In order to determine more efficiently children's attention point during infant-parent interaction, our head tracker also enables to compute the eye direction. Several devices exist such as goggles or other head mounted devices. Non intrusive systems for eye direction estimation usually use an external camera filming the user and detecting the direction of the eyes with respect to a known position. Several modern systems use infra-red lighting to extract the eye orientation by geometric calculation of the infra-red reflections. But to avoid distracting the infant, neither goggles nor IR leds should be used. More specifically for young children as we target, Noris et al.[10] developed a solution for eye gaze direction from a head mounted camera designed for children between 6 and 18 months. The solution is a wearable gaze direction detection system based on image appearance. The method has the advantage of not requiring any calibration from the wearer, and has been designed for use in a free play environment. Unfortunately, it still requires the child to wear the head mounted device, and a hat to support it, which would alter the child's appearance and his self-recognition. Matsumoto et al[?] proposes a non intrusive method for gaze direction measurement. The algorithm tracks the face and detects the gaze simultaneously and in real-time. Unfortunately, the key aspect of the system is the use of real-time stereo vision, which we ruled out because of its numerous disadvantages and restrictions. We propose a non intrusive method of gaze direction estimation based on the 3D visual tracker, that does not require the child to wear any device, and would not distract him.

2.3 Face-swapper

While there exists a rich body of work on replacing parts of images with new data, the replacement of faces in images has only developed recently. Zhu et al[14] proposed an approach to unsupervised facial image alignment, that is finding a transformation between two facial images so that they can be matched as well as possible.Their approach involves the extraction of a non-rigid mapping between facial images. Based on regularised face model, they use the Lucas-Kanade image registration approach to align faces without supervision. The proposed deformable Lucas-Kanade algorithm has been successfully applied to swap faces. LHowever, this method has only been tested on face images that have similar appearance, and most of all, has been designed only for still images and not video sequences.Indeed, the processing speed obtained, 6 images per second, is far below the 30 frames per second of a video. Bitouk et al[13] build an automatic face replacement system in images. It builds up a large library of faces from a maximum of people in different postures and under different light conditions. Given an input image, it detects all the faces present, select the candidate face images that are similar to the input face in appearance and pose. Then, it adjust the pose, lighting and colour, to finally blend the candidate face replacements to the input image. The method does not require 3D model and is automatic. This means that the approach is com-

patible with our 3D visual tracker. Our face-swapper targets videos and not only images. Therefore, the continuity and movement factors are as important as the appearance, and add further constraints to our face-swapper. For that purpose we will use the construction of our 3D visual tracker to blend the candidate face image to the head pose detected.

Chapter 3

3D visual tracker :method overview

3.1 Particle filtering

Particle filtering is a model estimation technique based on Monte Carlo simulations within a Bayesian framework, and has been developed for tracking curves in dense visual clutter.[9)]

The particle filter's goal is to estimate the sequence of hidden parameters $\mathbf{x} = (x1, ..xk)$, based on the observed data $\mathbf{y} = (y1, ..yl)$, the observation model $\mathbf{y} = \mathbf{f}(\mathbf{x})$ and the posterior distribution $p(\mathbf{x}|\mathbf{y})$. Random particles of the state-space variable $\mathbf{x}$ are generated, and checked against the current measured data to estimate the likeliness of that particle describing the real current state of the system. Namely, the particle filter calculates the probability function $p(\mathbf{x}|\mathbf{y})$.

The particle filtering algorithm is designed to apply iteratively to successive images in a sequence. The output of each iteration at time step t will be a weighted, time-stamped sample-set, denoted $\{\mathbf{s}_t^{(n)}, n = 1, \ldots, N\}$ with weights $\pi_t^{(n)} \sim p(\mathbf{y}_t|\mathbf{s}_t^{(n)})$, representing approximately the conditional state-density $p(\mathbf{x}_t|\mathbf{y}_1, \ldots, \mathbf{y}_t)$ at time t. For each time step, the first operation is to sample N times from the set $\{\mathbf{s}_{t-1}^{(n)}\}$ choosing a given element with probability $\pi_{t-1}^{(n)}$. Each element chosen from the new set is now subjected to the predictive steps, that is, first, a deterministic drift where identical elements in the new set undergo the same drift, then, a Brownian diffusion. The sample set $\{\mathbf{s}_t^{(n)}\}$ is thus generated but, as yet, without its weights. Finally the observation step generates weights from the observation density $p(\mathbf{y}_t|\mathbf{x}_t)$ to obtain the weighted sample-set $\{ (\mathbf{s}_t^{(n)}, \pi_t^{(n)})\}$

The particle filter therefore differs from the Kalman filter in that it can represent multiple hypotheses simultaneously. Particle filter uses learned dynamical models, together with visual observations, to propagate the random set over time. The result is a robust tracker that, despite the use of stochastic methods, runs in real-time.

Particle filter algorithm
for each step at time-stamp t

1. Select: For n=1,. . . ,N, draw samples $\mathbf{s'}_t^{(L)}$ from the estimated distribution

$$\mathbf{s'}_t^{(n)} \sim \pi_{t-1}^{(n)} = \frac{p(\mathbf{y}_1, \ldots \mathbf{y}_{t-1}|\mathbf{s'}_t^{(n)})}{\sum_{j=1}^{N} p(\mathbf{y}_1, \ldots \mathbf{y}_{t-1}|\mathbf{s'}_t^{(j)})}$$

2. Predict: drift and diffusion according to the dynamical model. The sample set $\{\mathbf{s}_t^{(n)}\}$ is generated.

3. Measure: For n = 1, ..., N update the weights of each particle based on the observed data $\mathbf{y}_k$

$$\pi_t^{(n)} \sim p(\mathbf{y}_t|\mathbf{s}_t^{(n)})$$

Fig. 3.1 Particle filter algorithm

3.2 Sparse Template Condensation Tracking

For our head tracking system, we define the state parameters as

$$\mathbf{x} = (T_x, T_y, T_{x_{dot}}, T_{y_{dot}t}, S, R_x, R_y, R_z, R_{y_{dot}}, \alpha) \tag{3.1}$$

where T_x, T_y are the translation coordinates of the target object $T_{x_{dot}}, T_{y_{dot}}$ are the velocity along the x and y axes S is the scale, R_x, R_y, R_z are the rotations along each axis $R_{y_{dot}}$ is the velocity of the rotation along axis y and α is a global illumination variable.

For our visual tracking system, the observation data $\mathbf{y} = (y1, ..yl)$ is based on sparse template matching, i.e. based on the difference error between the template and the input image at each time step. The matching error ϵ is calculated based on the difference in intensity values between selected pixels in the template (feature points) and the corresponding pixels in the image at time t (fig. 3.2).

A feature point *M_template* in the template is tracked in the image plane as point *M_image* based on the state estimate $\mathbf{x}^{(L)} = (T_x, T_y, S, R_x, R_y, R_z, \alpha)$ of a particle L=1,. . . ,P and according to an ellipsoid model of the face. *M_template* is first considered to be the projection of a point *M_ellipsoid* of a 3D ellipsoid so that

$$\begin{cases} \text{M\_template\_x} = \text{M\_ellipsoid\_x} \\ \text{M\_template\_y} = \text{M\_ellipsoid\_y} \\ \frac{M\_ellipsoid\_x^2}{a^2} + \frac{M\_ellipsoid\_y^2}{b^2} + \frac{M\_ellipsoid\_z^2}{c^2} = 1; \end{cases}$$

The point projected on the image can be written

$$\mathbf{M\_image}^{(L)} = S \cdot \mathbf{R}_x^{(L)}\mathbf{R}_y^{(L)}\mathbf{R}_z^{(L)}\mathbf{M\_ellipse}^{(L)} + \mathbf{T}^{(L)}$$

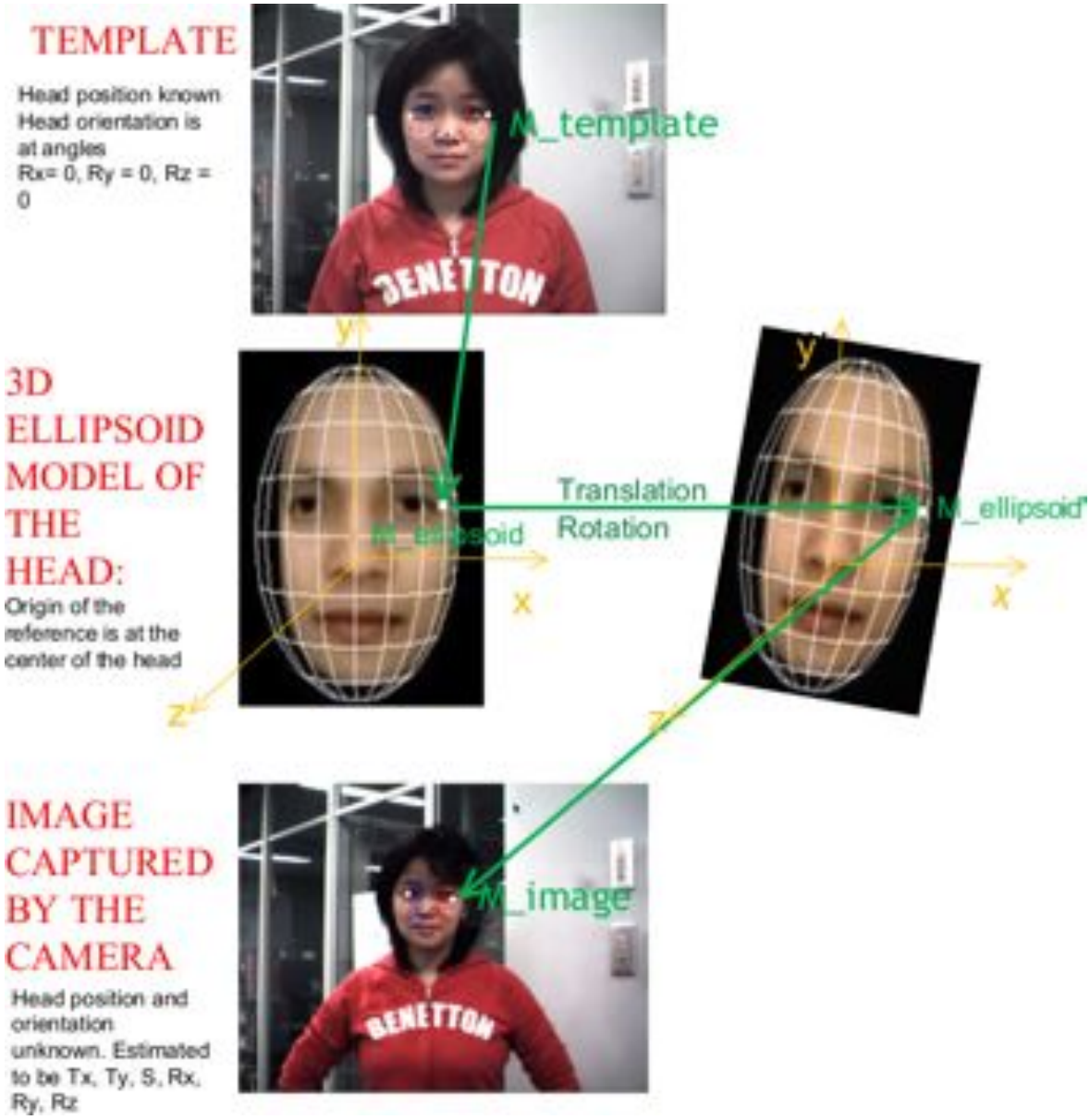

Fig. 3.2 Projection from the template to the image captured by the camera

where

$\mathbf{R}_x^{(L)}\mathbf{R}_y^{(L)}\mathbf{R}_z^{(L)}$ are the rotations matrices along each axis of the particle L
and $\mathbf{T}^{(L)}$ is the translation vector of the particle L

The average matching error is thus the difference in intensity between the two points *M_template* and *M_image*. If the intensity in the template image for a feature point m is I(*M_template*) and in the current image is I(*M_image*),

$$\epsilon^{(L)} = \frac{1}{l}\sum_{m=1}^{N}(I(M\_template_m) - I(M\_image_m^{(L)})) \tag{3.2}$$

and the probability that the particle represents the real current state is

$$p^{(L)} \propto \frac{1}{\epsilon^{(L)}} \tag{3.3}$$

Finally, if we can estimate the current state as the average of the particles by

$$\mathbf{x}_{estimated} = \frac{\sum_{L=1}^{N} p^{(L)}\mathbf{x}^{(L)}}{\sum_{L=1}^{N} p^{(L)}} \tag{3.4}$$

3.3 CUDA Implementation

3.3.1 Introduction to CUDA

CUDA (Compute Unified Device Architecture) is a parallel computing architecture developed by NVIDIA.[3)] CUDA is the computing engine in NVIDIA graphics processing units (GPUs) that is accessible to software developers through industry standard programming languages. Until the recent years, the use of GPUs has been mostly limited to graphics processing. However, with the development of General-Purpose Computing on Graphics Processing Units (GPGPU), it has become possible to use a GPU for more common tasks such as mathematical calculations. The use of GPUs parallel computing architecture enables dramatic increases in computing performance by harnessing the power of the GPU that can be viewed as a data-parallel computing device that operates in collaboration with the Central Processing Unit (CPU). The CPU would play the role of host, and send data to the GPU (device) to compute. The use of GPU is therefore best for independent calculations that can be carried in parallel, like the calculations in our particle filter of the independent particles. More precisely, we use the GPU in our particle filter algorithm for each time step for the prediction phase and the measure phases. For our implementation, we chose from amongst the various programming libraries, Nvidia's GPGPU technology for Nvidia GeForce-based GPUs CUDA.

Fig. 3.3 Comparison between the architectures of the CPU and the GPU (courtesy of Kirk et al.[4])

A CUDA program consists of one or more phases that are executed on either the host (CPU) or a device (GPU). The major difference in architecture between a CPU and a GPU, as illustrated in figure 3.3 is that while a CPU devotes a lot of space to memory and cache, a GPU is almost entirely dedicated to data processing with over 200 cores in current day devices. This has led to a overall number of GFLOPS in a GPU to be more than 10 times more important than a current day CPU. Nevertheless, in CUDA, host and devices have separate memory spaces. This reflects the reality that devices are typically hardware cards that come with their own Dynamic Random Access Memory (DRAM). In order to execute a kernel on a device, the programmer needs to allocate memory on the device and transfer the pertinent data from the host (CPU) memory to the device(GPU). Similarly, after the device execution, the programmer needs to transfer result data from device back to the host. Thus, while a CUDA program can be speeded up by parallel processing power of the GPU, it can also be considerably slowed down by the data transfer. Therefore, from a GPU to another, the program execution speed can vary considerably depending on the transfer bandwidth from host to device, device to host and device to device.

In more detail, the way CUDA allows the user to access the GPU is through sections of code called kernelsB The code blocks are signalled by the CPU and then executed entirely within the GPU. When a kernel is invoked or ”launched” (fig. 3.4), it is executed as grid of parallel threads. Each CUDA thread grid typically comprises thousands to millions of lightweight GPU threads per kernel invocationB Threads in a grid are organised into a two-level hierarchy. At the top level, each grid consists of M thread blocks. Each block is in turn organised as a 3 dimensional array of N threads. The GPU is implemented as a set of MxN multi-processors, each composed of many multi processing units. Each of these units computes at every time step the matching error of the feature points and updates the particles.

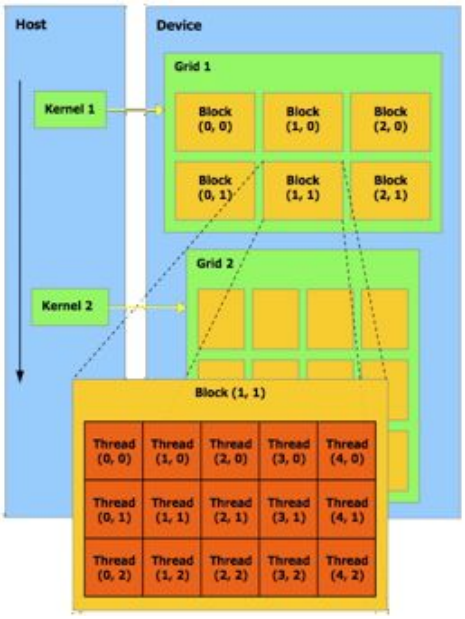

Fig. 3.4 Threads organisation with CUDA (courtesy of Kirk et al.[4])

3.3.2 Details of the particle filter implementation with CUDA

In the initialisation phase, both the host and the device initialise twin particles, and the host gives the device the information on the feature points to track. These points do not change during the execution of the program, so will be kept in the GPU memory to be used for the measure phase of the particle filter algorithm.

Then, at each time step (fig 3.7):

- the host sends the current video frame to the device
- measure phase: the device calculates the likelihood of each particle based on the intensity of the feature points in the current video frame (Listing 3.5)
- the device sends back to the host the information about the each particle and their likelihood
- the host can estimate then the current state $\mathbf{x}_{estimated}$ as the average of the particles.
- prediction phase: the drift and diffusion of the particles are realised by the device (Listing 3.6).

void cudaLikelihood(unsigned char *imgGray):

1: arrayFeature ← featureData
2: arrayImageGray ← imgGray
3: texFeature ← arrayFeature
4: texImageGray ← arrayImageGray
5: calculateLIkelihoodByKernel
6: likelihoodDataHost ← likelihoodDataDevice
7: particleDataHost ← particleDataDevice

likelihoodKernel(float* particleDataD, float* liData) :

1: calculateTransformationParameters
2: **for** all feature points k **do**
3: $(x, y, z) \leftarrow$ the coordinates of k;
4: $(r, g, b) \leftarrow$ the intensity of the template image at point (x, y, b) ;
5: $(px, py, pz) \leftarrow$ the transformed coordinates;
6: $(pr, pg, pb) \leftarrow$ the intensity of the current image at point (px, py, pb) ;
7: $error \leftarrow error + (pb - b) * (pb - b) + (pg - g) * (pg - g) + (pr - r) * (pr - r)$
8: **end for**

Fig. 3.5 Particle filter's measure phase is processed by the GPU

void cudaUpdate(void)

1: particleDataDevice ← particleDataNew
2: updateLikelihoodByKernel

updateKernel(float *particleDataD, float *randRangeD) :

1: particleDataD ← particleDataD + random*randRangeD;

Fig. 3.6 Particle filter's prediction phase is processed by the GPU

CUDA initialisation

```
1:  system noise setting
2:  initialisation of the particles of the GPU.
3:  initialisation of the particles of the CPU
4:  get data of template
5:  loop
6:     imgGray ← grayscale of current camera frame
7:     cudaLikelihood(unsigned char *imgGray)
8:     maxLikelihood ← max ( likelihood of all particles)
9:     if maxLikelihood ¡ threshold1 then
10:       particleDataH ← the position detected by haar detection
11:    end if
12:    if maxLikelihood ¿ threshold2 then
13:       global position and orientation ← average of particleDataH
14:    end if
15:    draw the results
16:    particleDataNew ← particleDataH
17:    cudaUpdate();
18:    if maxLikelihood inf threshold3 then
19:       system noise setting
20:       initialisation of the particles
21:    end if
22: end loop
```

Fig. 3.7 Processing of the particle filter by the CPU

3.4 Multi-processing use

Our tracker used 4 main processing threads to track the head position:

- video capture from the camera
- face detection using Viola-Jones algorithm
- tracking
- post-processing, which is for our experiment face swapping
- show on screen

The scheme of how these processes are related, and how a captured camera is dealt processed by the different threads is shown in fig. 3.8. The relationship shown here is data-wise, for the same captured camera, how it is going to be handled by each of the threads. The graph does not show the time-wise synchronisation of the threads. Fig. 3.9 show the organisation of the same threads in the system if parallel processing were not used.

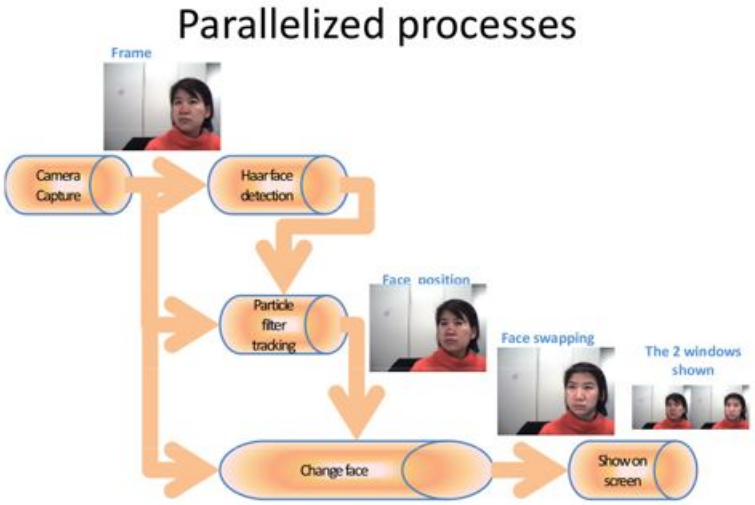

Fig. 3.8 Main processing streams used by our face-swapper in parallel processing

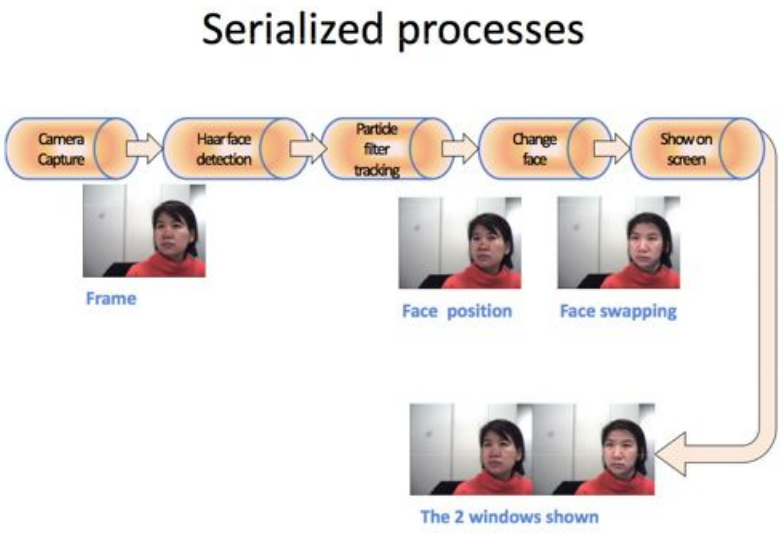

Fig. 3.9 Graph of the serialized processing streams: this configuration would lead to a longer processing time

3.4.1 Video capture

The video capture from the camera was revealed to be one of the slowest nodes of the tracker, as our firewire camera captures only 30 frames per second, which means a delay of at least 33ms. For this reason, we isolated the video capture task into a separate thread, so that the tracking calculation can take place simultaneously.

3.4.2 Face detection

The captured image is scanned to look for new faces, using the haar classifier first developed by Viola and Jones.[21] It is classifier based on the value of rectangle features in a cascade structure. It organises the classification as a rejection cascade of nodes, where each node is a multitree Adaboost classifier. True face detection is declared only if the computation makes it through the entire cascade

This face detection is essential for the initialisation, and to give an estimation of the head position to the tracking system when the estimated states have a too low probability.

3.4.3 Tracking

This is the actual particle filtering that is performed. When the particle filter's probability function $p(\mathbf{x}_t|\mathbf{y}_1,\ldots,\mathbf{y}_t)$ is above a threshold, position and orientation of the head is obtained as the average of the particles, and passed to the display thread (fig. 3.7). When the particle

filter's probability function $p(\mathbf{x}_t|\mathbf{y}_1, \ldots, \mathbf{y}_t)$ is below this threshold, the particles will be re-generated around the latest position found by the haar classifier. Our tracker is successful for different rotation angles of the head (fig. 3.10 and fig 5.3, as well as at tracking a face with background distractor faces (fig 3.11).

Fig. 3.10 Swap faces with tilted head

Fig. 3.11 Swap faces with background distractor faces

3.4.4 Post-processing

The post-processing thread takes the head position and orientation output by the tracking thread to process the required task specific to the application developed. The simplest post-processing is to simply display the results of the tracking. In our case, we chose to build a face-swapper over the tracker, that is automatically replacing the subject's face by someone

else's, as illustrated in fig.3.12. We can superimpose the face of another person over the face detected, so that the video displayed is that of a person exactly mirroring oneself and with the same motions but with another face.

Fig. 3.12 Swap faces as an example application

3.4.5 Management of the parallel processing

The management of multiple threads running in parallel implies not only management of the data sharing, how to protect against multiple writes, but also thread synchronisation. Different threads run at different speed depending on their computing load, but to make the parallel processing meaningful, to maximise the overall speed of the system, we made the threads have equivalent computing loads(cf. Annex). And so as not to overload the processor with useless runs of fast threads on the same already processed data, we used condition variables to synchronise threads and make the fast threads like the show-on-screen thread wait until arrival of new data. Fig. 3.13 indicates how the different processes run parallel and synchronise with each other, and how they share data.

3.5 Eye direction computation

Our efficient face tracking enables us to track the eyes efficiently and therefore to infer the eye direction (fig.3.14). From the 3D posture of the head (a), we have access to the approximate position of the eyes and we can extract the image of the eyes(b). A hue-image and a threshold filter singles out the eye from the rest of the skin (c), and enables to locate more precisely the eye itself (d). From this image, we can simply detect the position of the pupil in the eye to get the position of the pupil and the lateral eye direction (e).

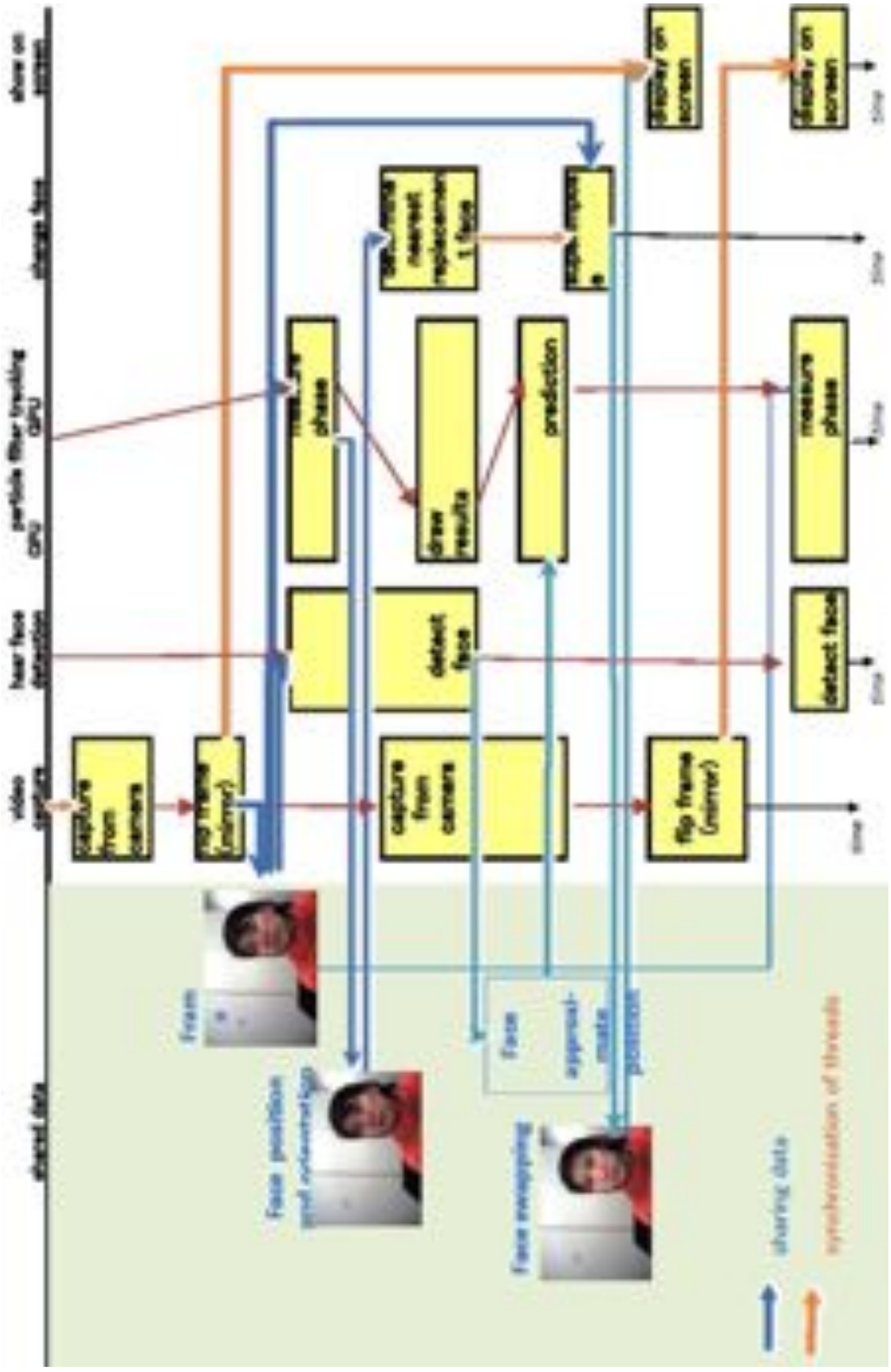

Fig. 3.13 Synchronisation and sharing of data between the different processes

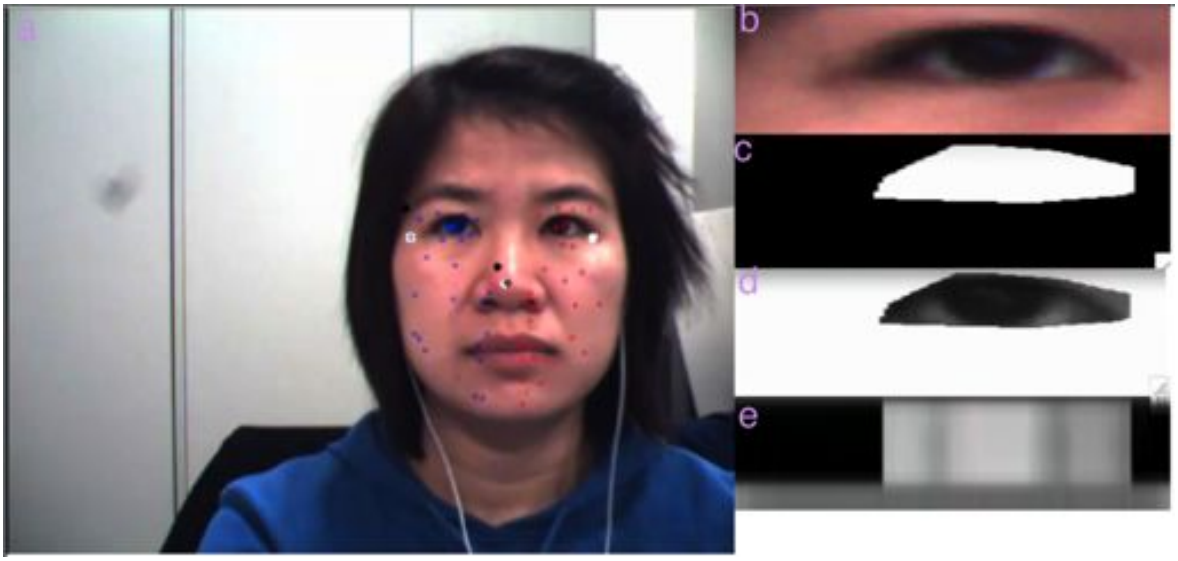

Fig. 3.14 Face tracking enables eye direction computation. (a) input image with 3D head pose tracked.(b)image of the eyes trough the approximate position of the eyes. (c) mask for the eyes, computed through threshold on the hue image. (d) grayscale image of the eye. (e) histogram of intensity that detects the pupil position

In order to evaluate the eye direction estimation, we set up a measure in a controlled environment where the subject's attention is attracted by a flashing figure on a screen. To better evaluate the gaze direction (and not the head direction) estimation, we tried to constraint the head movement of the subject to a minimum, but could not totally get rid of them. The flashing object appears at given angles as presented on fig. 3.15. For these measures, we needed two calibration measures: the measures for when the subject looks straight forward (angle 0 degree) and another angle $\theta 1$ non null. During the estimation phase, the head rotation Ry and the relative eye direction θ are both computed. The (global) eye direction is the sum of the head rotation and the eye direction compared to the calibration measures, i.e.

$$eyedirection = \theta - \theta_{straightforward} - (Ry - Ry_{\theta 1}) \tag{3.5}$$

The measures show that errors on the estimation of the global eye direction is smaller than 3 degrees, with an average of 2 degrees. The accuracy obtained by this method is inferior to the existing eye trackers. However our system is a non invasive method, designed to be used in a free play environment, with no constraint on the subject. Moreover, due to the constraint of working with young children, especially children with ASD, the system does not seek to be as accurate as other state-of-the-art eye trackers, but rather focuses on how to enhance natural behaviour, and not disturb the subject.

angle of flashing object (degree)	18.2	9.3	0	-9.3	-18.2	global
relative eye direction	18.2	10.5	0.7	-4.9	-11.1	
head rotation	9.0	7.4	8.3	9.9	10.9	
eye direction	18.2	11.3	0.7	-6.6	-13.7	
standard deviation of eye direction	2.1	2.6	2.3	1.1	1.4	1.9
error	(calibration).	2.0@	0.7	2.7	4.4	1.9

Fig. 3.15 Results of the measures in our estimation of gaze direction. Angles presented here are average angles, expressed in degrees. Calibration has been done for $\theta 1$ = 18.19

Chapter 4

Face-swapping

In order to superimpose the face of another person B on the detected face of the subject A, we prepare a set of replacement faces tagged with the head position and orientation x of the person B in advance (fig 4.1) , using the simple tracking system to save the template images and tag them with the head position and orientation. We therefore automatically generate, using only a frontal picture of a person B, a set of replacement faces.

Fig. 4.1 Set of replacement faces tagged with head position and orientation of the person B. Horizontal axis represents the horizontal rotation Ry taking values -35, -30, -25 ,.... 25, 30. Vertical axis represents the lateral rotation Rz taking values -20, -15,...15, 20.

During the execution of the program, the post-processing thread will compare the state parameters $\mathbf{x}^A$ computed on A's face with the state parameters of the template images existing of B (fig 4.2). It will select the closest face replacement $\mathbf{x}^{Closest}$ to superimpose it on A's

face. In order to render the movement and the temporal continuity of the displayed video, the replacement faces are first interpolated before the superimposition. Once the closest replacement face has been selected, a point Mclosest of the replacement image will be interpolated into the point M _*image* according to the equation:

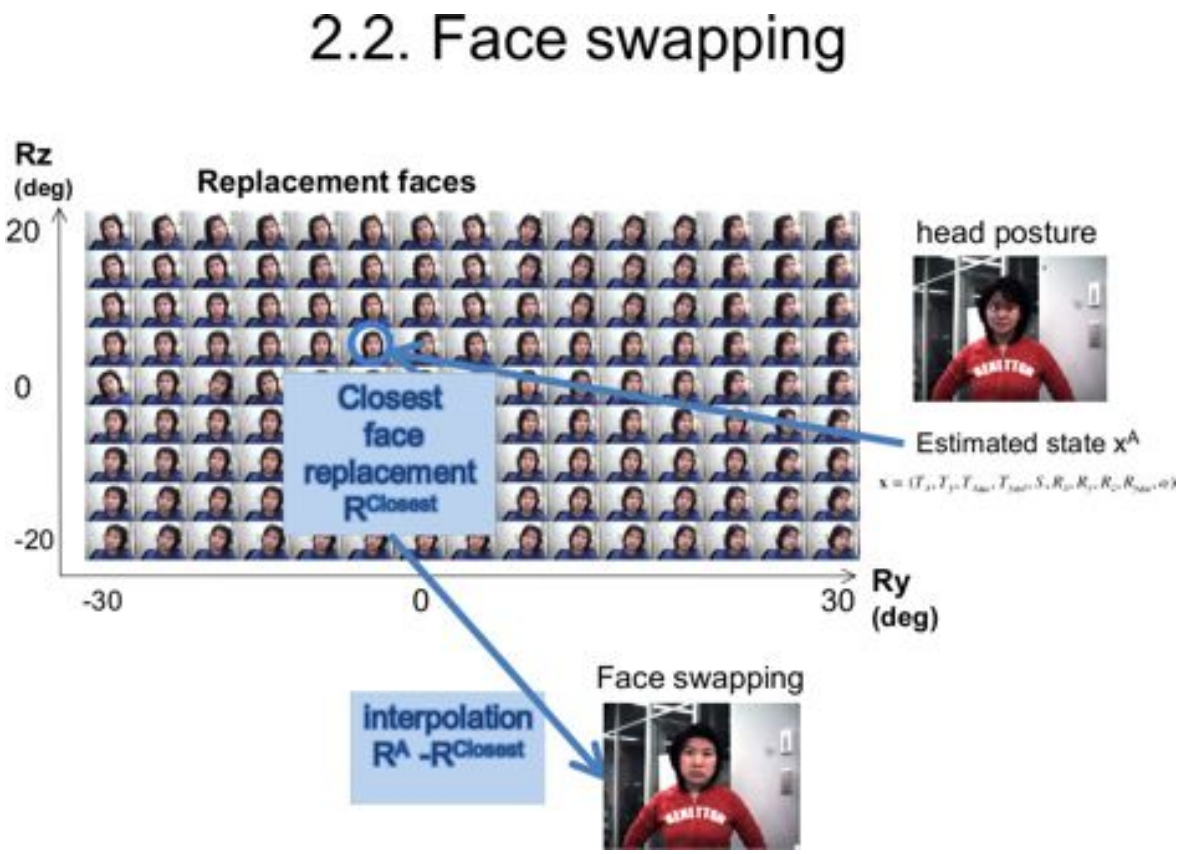

Fig. 4.2 Set of replacement faces tagged with head position and orientation of the person B. Horizontal axis represents the horizontal rotation Ry taking values -35, -30, -25 ,.... 25, 30. Vertical axis represents the lateral rotation Rz taking values -20, -15,...15, 20.

$$\mathbf{M\_image} = S \cdot (\mathbf{R}_x^{(A)} - \mathbf{R}_x^{(Closest)})(\mathbf{R}_y^{(A)} - \mathbf{R}_y^{(Closest)}) \quad (4.1)$$
$$(\mathbf{R}_z^{(A)} - \mathbf{R}_z^{(Closest)})\mathbf{Mclosest} + (\mathbf{T}^{(A)} - \mathbf{T}^{(Closest)})$$

This interpolation allows the replacement face to look dynamic, not only a static image stuck on the video, but moving with the face of the original video. We therefore have a comprehensive system for automatically replacing faces in videos, that renders the dynamic of the movements of the head.

Chapter 5

Discussion on the performance of the 3D head tracker

Our choices to use particle filter and multi-processing has been made based on precision and speed criteria.

5.1 Comparison with Viola-Jones algorithm for face tracking

The haar classifier is not sufficient in estimating the head position and orientation for three reasons.

One reason is that the haar classifier is slow and lacks stability from one frame to the other. The classifier takes more time to scan the image and find the face position on the image than a condensation algorithm that already has hypotheses about the probable position. Moreover, the faces detected by the haar algorithm from one frame to the other are independent of each other, therefore unstable temporally. The size of the face is especially subject to great variations from a frame to the other. Fig 5.1 and 5.2 show two screenshots of temporally near frames. The image is almost the same, however the rectangles returned by the haar face detection are notably different. Although the difference between the rectangles are not significant in terms of pixels, the rapid variations make it very unstable. Superimposing a replacement face on the face position and size returned by haar would cause too much instability and make the image flicker.

The second reason is that a haar classifier can only detect an object at a certain orientation only, and not at any orientation as we are aiming at. Our goal would require using several haar classifiers, one for each orientation of the face. It would then require the use several haar classifiers, meaning more processing needed. This solution is not acceptable for a real-time tracker. On the other hand, the sparse template tracker can detect a large range of face orientations, as show fig.5.3 with a rotation angle of more than 70 degrees.

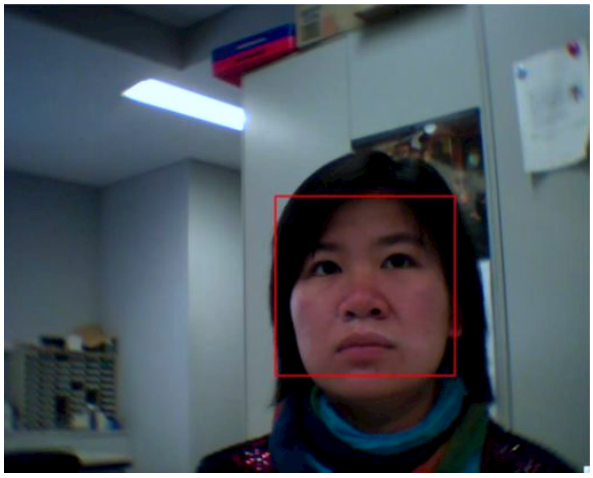

Fig. 5.1 Face detected by haar algorithm: on the left side of the face, a pan of the hair is outside the rectangle

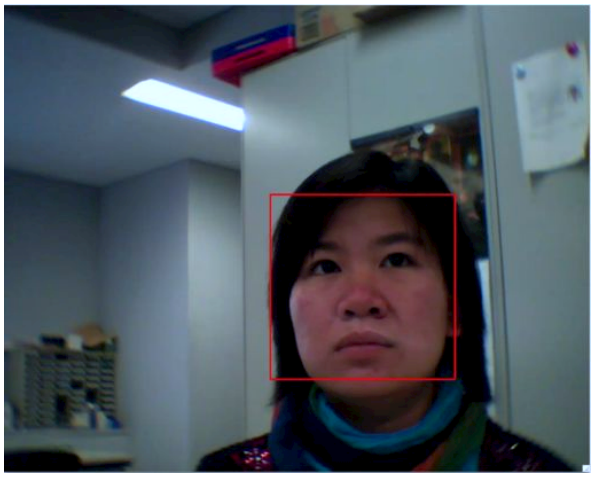

Fig. 5.2 Face detected by haar algorithm a few frames later: on the left side of the face, all the hair is has been bordered by the rectangle returned by haar algorithm

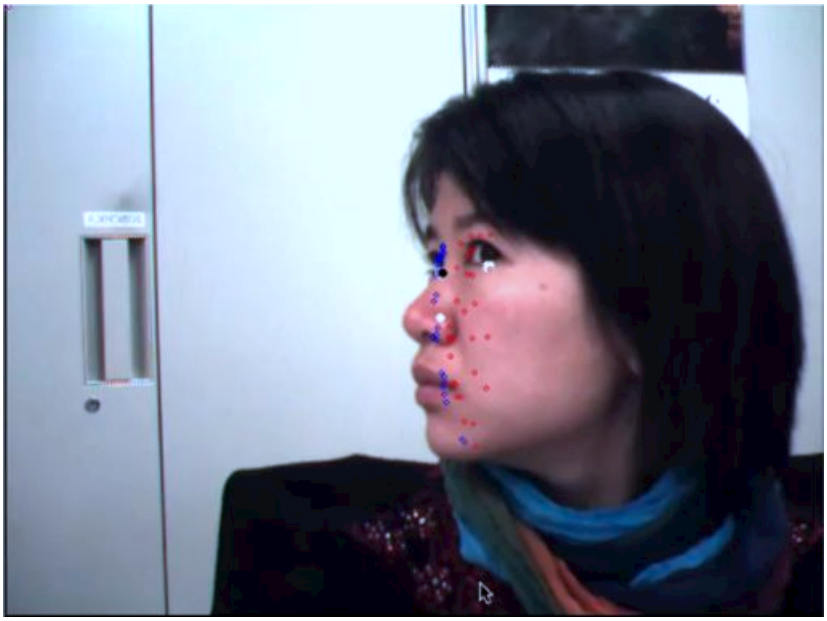

Fig. 5.3 Sparse template tracking can detect a large range of face orientation, contrarily to the haar face detection

The third reason is that haar face detection is very weak to partial occlusion such as hands or toys brought to the mouth or other parts of the face when children are playing. On the contrary, as fig. **??**occlusion shows, our tracker is robust to occlusion.

The face haar detection is therefore not a solution for our system. We are aware that a comparison between face detecter and a tracker would underline huge differences between the two methods, and it would be more interesting to compare our tracking algorithm with another face tracking algorithm such as the one developed by Matsumoto et al.[1)]

5.2 Evaluation of the speed of our system

The speed of the tracker benefits greatly from multi-processing that enables the camera capture and the haar face detection to run at the same time as the particle filter tracking.

In the example of the face swapping program that uses only haar face detection without particle filter tracking, the single-processing program would have a delay of 400 ms while the multi-processing program has a delay of 60 ms. The use of sparse template particle filter enables to reduce even more this delay. With regard to the processing time of the particle filter only, when the number of particles is 5120, the measure phase takes 5 ms and the prediction phase takes 5 ms, which add up to a total processing time for the particle filter of approximately 10 ms. For the whole tracker system, the total delay of the tracker plus face-swapper measured was 200 ms without the processing power of the GPU, and it decreased

Fig. 5.4 Sparse template tracking is robust to occlusion. The left image shows the result of the face tracking. The right image is the superimposed image.

to 66 ms. The hardware used for these measures are:

- host: a 2*2.8 GHz Quad-Core Intel Xeon running with Mac Os X 10.5.8
- GPU : a NVIDIA GeForce GT120 with 515MB. The transfer bandwidth are: from host to device: 189 MB/s, from device to host : 1678 MB/s, from device to device: 47624 MB/s.

5.3 Choice of the parameters of the particle filter

We chose to increase the number of parameters of our tracker. Mateo Lozano et al.[8] uses 6 cinematic variables, we added $T_{x_{dot}}$, $T_{y_{dot}}$ and $R_{y_{dot}}$ to gain robustness to quick movements. To assess this gain in robustness, we prepared sample videos consisting of the translation of a still image at different speed, ranging from 1 pixel per frame to 10 pixels per frame. The result is that:

- for low speed translations, both systems with either 6 (fig.5.5) or 9 parameters (fig.5.7) perform well. The figures show the measurement of the face position along the horizontal axis Tx as a function of time. At both translation speeds, we can see the that the slope of the curve Tx is a straight line, showing a detection of the translation.
- for high speed translations, the system with 9 parameters (fig. 5.8) outperforms the system with only 6 parameters. (fig.5.6). Although irregular, the 9 parameters system still measures the face position Tx with a regular slope, thus detecting a translation; whereas the 6 parameters system would measure very irregular Tx, and the slope of the curve is not always observed.

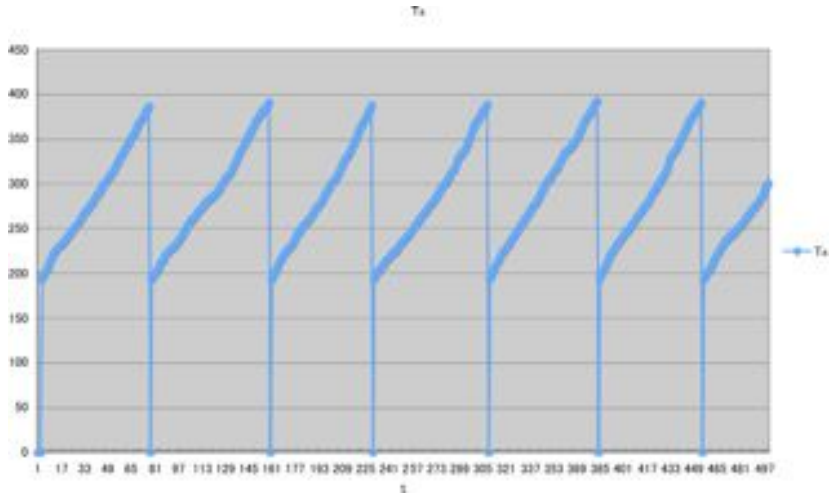

Fig. 5.5 Translation measured by a tracking system with 6 parameters, on a video created by translation of a image at 1 pixel per frame. Vertical axis is Tx in pixels, Horizontal axis is time

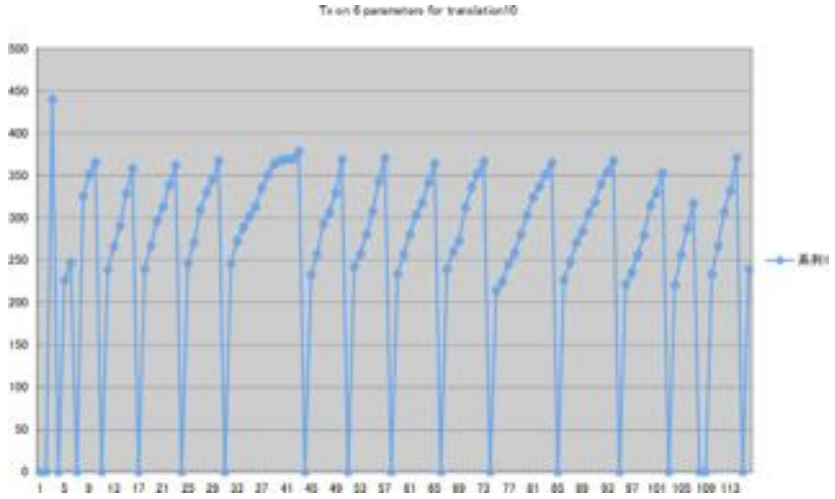

Fig. 5.6 Translation measured by a tracking system with 6 parameters, on a video created by translation of a image at 10 pixel per frame. Vertical axis is Tx in pixels, Horizontal axis is time

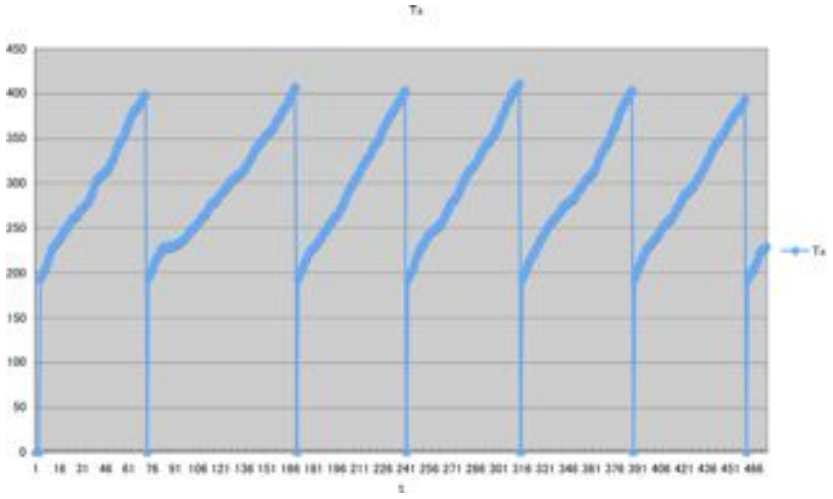

Fig. 5.7 Translation measured by a tracking system with 9 parameters, on a video created by translation of a image at 1 pixel per frame. Vertical axis is Tx in pixels, Horizontal axis is time

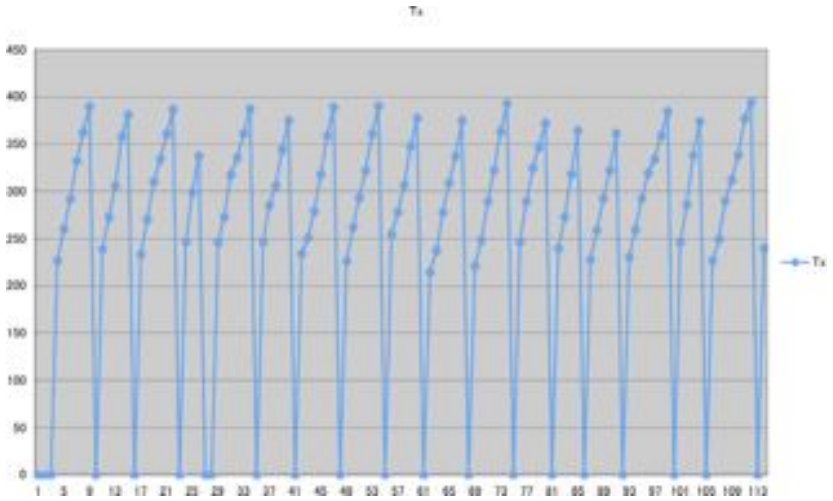

Fig. 5.8 Translation measured by a tracking system with 9 parameters, on a video created by translation of a image at 10 pixel per frame. Vertical axis is Tx in pixels, Horizontal axis is time

This choice to add velocity to the state parameters originate from the idea that we should take into account the variation of the dynamic of the head, the change in velocity of the movement of the head. Under such considerations, a natural choice would be to use 12 parameters (6 for pose and 6 for motion). But given the "natural" movement of the head, rotations around the vertical axis would occur more often than other axis, especially in our experiment to watch the left or right side of the screen. Therefore $R_{y_{dot}}$ will have a major impact whereas $R_{x_{dot}}$ and $R_{z_{dot}}$ have lower importance. Given that each each new parameter introduced has a calculation cost proportional to the number of particles used, we decided to discard $R_{x_{dot}}$ and $R_{z_{dot}}$ to gain in processing speed. A further comparison between 9 and 12 parameters in terms of precision and processing time should be carried out.

5.4 Evaluation against motion capture

The system is under evaluation against motion capture system. We captured movements both with cameras and motion capture to assess the accuracy of our tracking program.

The evaluation has been conducted first on an adult subject for movements in normal speed, with head pitch ranging from -40 to 40 degrees. Fig 5.9 show a comparison between measures by a motion capture system and our tracker. Although timing calibration still needs to be improved, the comparison gives encouraging results, with an average error of 9 degrees.

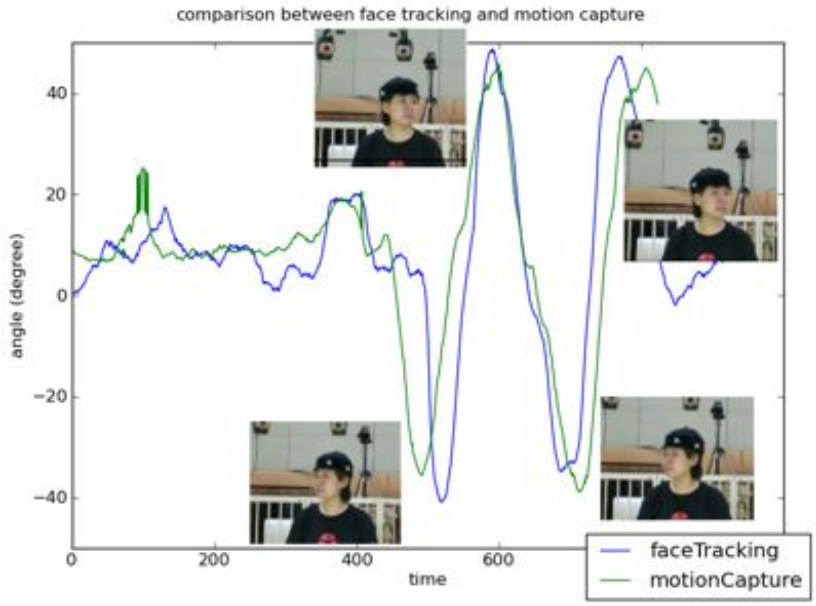

Fig. 5.9 Translation and rotation measured by our tracking system compared with the motion capture data. The screenshots of the video correspond to the extremum of head rotation angle

Chapter 6

Limitations of the face swapper

6.1 Occlusion

Although the tracking is robust in this regard, the face-swapping unfortunately is sensitive to occlusion, such as eyeglasses or hands or toys brought to the face (fig 5.4). This problem could be solved using the advantage of processing a video to build up a model of the face. By gathering information on the face through successive frames of the video, we can build a model of the face to use to detect any occlusion. The drawback would then be the computation time that increases on building a complex face model.

6.2 Large movements

Even though our tracker performs well on large movements, our face-swapper in videos is sensitive to large movements. We use for superimposition a replacement face of person B that we interpolate to fit the current face posture of subject A. This interpolation, realistic if face postures of A and B are close, performs less well as the distance between the face postures increases. To solve this problem of appearance realism, we opted for the use of a set of replacement faces to obtain a more realistic appearance. The drawback of this method is that on large movements of the head, the program will automatically select the replacement faces whose 3D pose is nearest that detected by the tracker, which means that the program will switch from one replacement face to another, and create a temporal discontinuity. Therefore, the more replacement faces we use, the more often temporal discontinuity occur in the video. There is thus a tradeoff between static realism and dynamic realism.

Chapter 7

Conclusion

Fast and robust object tracking is required by many applications in fields such as automated surveillance systems, man-machine interaction or augmented reality systems.

We have presented a system for 3D visual tracking capable of achieving real-time performance thanks to the use of a parallel computation. The tracker is a non-constraint system that needs only a single camera for determining both the position and orientation of the head. We plan to evaluate the system applied to infants and use this face-swapper system in experiments with children to investigate self-recognition, notably the importance of the contingency or familiarity factors in self-recognition. These behavioural experiments could be completed by neuro-scientific data such as the activation in the frontal lobe of the right hemisphere during self-recognition task[26] or the singular activity of the default network during goal-directed task and self-introspection tasks,[27] to model the development of self-consciousness in infants.

In conclusion, we have developed a real-time 3D head tracker, using sparse template matching and parallel processing. The tracker can be used to analyse the subject's attention through the head position and orientation in experiments, or it can be used in a wide range of applications for human-machine interaction or augmented reality systems.

Chapter 8

Appendix

8.1 Video capture thread

```
void* video_capture_thread(void* arg)
{
      struct argument *argth;
      argth = (argument*)arg;
     take = 0;

      while(1)
            {
                  camera_frame(argth->frameCapture );
                     //flip
                  pthread_mutex_lock (&mutexFrame);
                  cvFlip(argth->frameCapture,argth->frame,1);
                  pthread_cond_signal(&cond_frame);
                  pthread_mutex_unlock (&mutexFrame);
                  if(argth->stop == 1)  pthread_exit(NULL);
            }

      pthread_exit(NULL);
      return NULL;
}
```

8.2 Haar face detection thread

```
void* haar_face(void* arg)
{
      struct argument *argth;
      argth = (argument *)arg;
      IplImage* img = cvCreateImage(cvSize(mediumWidth,mediumHeight),
         IPL_DEPTH_8U, 3);
      int num_faces = 0;

      argth->face_detect = new ViolaJones(mediumWidth, mediumHeight);
      argth->face_detect->init();

      while(1)
            {
                  //detect face on the smaller size image
                  pthread_mutex_lock(&mutexFrame);
                  cvResize(argth->frame,img);
                  pthread_mutex_unlock(&mutexFrame);
                  num_faces  = argth->face_detect->detect_face(img);

                  if(num_faces>0)
                        {
                              argth->haarFace.x          = mediumScale*argth->
                                 face_detect->face.x ;
                              argth->haarFace.y          = mediumScale*(argth
                                 ->face_detect->face.y );
                              argth->haarFace.width  = mediumScale*(argth->
                                 face_detect->face.width);
                              argth->haarFace.height = mediumScale*(argth->
                                 face_detect->face.height);
                        }

                  pthread_testcancel();
                  if(argth->stop == 1)  pthread_exit(NULL);
            }
      pthread_exit(NULL);
      return NULL;
}
```

8.3 Particle filter tracking thread

```
void particle_filter_tracking_thread(void* arg)
{
    struct argument *argth;
    argth = (argument *)arg;

    //CUDA initialisation
    //system noise setting
    //initialisation of the particles of the GPU
    //initialisation of the particles of the CPU
    //get data of template

        while(1)
         {
              pthread_mutex_lock(&mutexFrame);
              cvCopy(argth->frame, image2);
              pthread_mutex_unlock(&mutexFrame);
       @@cvCvtColor(image2, gray, CV_BGR2GRAY);
       @@cuda_Likelihood( (unsigned char *)(gray->imageData));
              /*
              maxLikelihood    max ( likelihood of all particles)
              if maxLikelihood < threshold1 then
                     particleDataH  argth->haarFace
              end if
              */
               if(maxLikelihood>threshold1)
                  {
                       pthread_mutex_lock(&mutexTransf);
                       //average of particleDatH
                       argth->Tx  = (ave_Tx + ave_Tvx);
                       argth->Ty  = ave_Ty;
                       argth->S   = ave_S;
                       argth->Rx  = ave_Rx;
                       argth->Ry  = ave_Ry;
                       argth->Rz  = ave_Rz;
                       argth->alpha = ave_alpha;
                       pthread_mutex_unlock(&mutexTransf);
                  }
              /*
              draw the results particleDataNew    particleDataH
              cudaUpdate();
              if maxLikelihood inf threshold3 then
                     system noise setting
                     initialisation of the particles
              end if
              */
        }
```

```
        pthread_exit(NULL);
        return NULL;
}
```

8.4 Change face thread

```
//other's face thread
void* change_face1(void* arg)
{
      struct argument *argth;
      argth = (argument*)arg;

      while(true)
            {
                  pthread_mutex_lock (&mutexFrame);
                  cvCopy(argth->frame, resultImage1);
                  pthread_mutex_unlock (&mutexFrame);

                  // face in face
                  pthread_mutex_lock(&mutexTransf);
                  // get the face posture Tx, Ty, S, Rx, Ry, Rz
                   pthread_mutex_unlock(&mutexTransf);

                   //determine closest replacement face
                   //superpose images


                  pthread_mutex_lock (&mutexResult1);
                  cvCopy(resultImage1, argth->result1[take]);
                  pthread_cond_wait(&cond_result,&mutexResult2);

                  show =take;
                  take=next_frame(take);
                  pthread_mutex_unlock (&mutexResult1);
                  pthread_testcancel();
                  if(argth->stop == 1)  pthread_exit(NULL);
            }

      pthread_exit(NULL);
      return NULL;
}

//self face (timing management)
void* change_face2(void* arg)
{
      struct argument *argth;
      argth = (argument*)arg;
      int x, y, width,height;

      while(true)
            {
                  pthread_mutex_lock (&mutexFrame);
```

```
                cvCopy(argth->frame, argth->result2[take]);
                pthread_cond_signal(&cond_result);
                pthread_mutex_unlock (&mutexFrame);

                pthread_mutex_lock (&mutexResult2);
                pthread_cond_signal(&cond_result);
                pthread_mutex_unlock (&mutexResult2);

                pthread_testcancel();
                if(argth->stop == 1)  pthread_exit(NULL);
            }

    pthread_exit(NULL);
    return NULL;
}
```

8.5 Show on screen thread

```
void* show_thread(void* arg)
{
      struct argument *argth;
      argth = (argument*)arg;
      int bigWidth  =2*windowWidth;
      int bigHeight = windowHeight;
      IplImage *big =   cvCreateImage( cvSize(bigWidth, bigHeight),
                                       IPL_DEPTH_8U, 3);
      cvZero(big);

      while(1)
            {
                  pthread_cond_wait(&cond_frame, &mutexFrame);

                  //show on left side, frame
                  pthread_mutex_lock (&mutexResult2);
                 cvResize(argth->result2[show], big);
                  pthread_mutex_unlock (&mutexResult2);
                  cvResetImageROI(big);

                  //show on right side, result
                  cvSetImageROI(big,cvRect(windowWidth, 0, windowWidth,
                      windowHeight));
                  pthread_mutex_lock (&mutexResult1);
                  cvResize(argth->result1[show], big);
                  pthread_mutex_unlock (&mutexResult1);
                  cvResetImageROI(big);


                  pthread_mutex_lock (&mutexWin);
                  cvCopy(big,argth->affiche);
                  pthread_cond_signal(&cond_win);
                  pthread_mutex_unlock (&mutexWin);

                  if(argth->stop == 1)  pthread_exit(NULL);
            }
}
```

References

1) Y. Matsumoto, N. Sasao, T. Suenaga, T. Ogasawara, "3D Model-based 6-DOF Head Tracking by a Single Camera for Human-Robot Interaction", *IEEE International Conference on Robotics and Automation*, 2009

2) FaceAPI Library commercialised by SeeingMachines, 2010. http://www.seeingmachines.com/product/faceapi

3) CUDA Homepage on NVIDIA website, 2010. http://www.nvidia.com/object/cuda_home_new.html

4) University of Illinois Nvidia CUDA Course taught by Wen-mei Hwu and David Kirk, Spring 2009

5) A. S. Montemayor, J.J. Pantrigo, R. Cabido, B. Payne. "Bandwidth improved GPU particle filter for visual tracking", *Ibero-American symposyum on computer graphics SIACG*, 2006.

6) G. R. Bradski. "Computer vision face tracking for use in a perceptual user interface", *Intel Technology Journal Q2 f98*, 1998.

7) Y. Matsubara, T. Shakunaga, "Real-time Object Tracking by Sparse Template Matching", *IPSJ SIG Technical Report*, March 2004

8) O. Mateo Lozano, K. Otsuka, "Real-time Visual Tracker by Stream Processing", *Journal of Signal Processing Systems*, vol. 57(2), November 2009

9) M. Isard, A. Blake, "Condensation - Conditional Density Propagation for Visual Tracking", *International Journal of Computer Vision, vol. 29(1)*, 5-28, 1998

10) B. Noris, K. Benmachiche, A.G. Billard, , " Calibration-Free Eye Gaze Direction Detection with Gaussian Processes", *Proceedings of the International Conference on Computer Vision Theory and Applications*, 2008

11) Y. Matsumoto, A. Zelinsky, "An Algorithm for Real-Time Stereo Vision Implementation of Head Pose and Gaze Direction Measurement," *Fourth IEEE International Conference on Automatic Face and Gesture Recognition (FG'00)*, 2000

12) S.I Kawamoto, H. Shimodaira, T. Nitta, T. Nishimoto, S. Nakamura, K. Itou, S. Morishima, T. Yotsukura , A. Kai , A. Lee , Y. Yamashita , T. Kobayashi , K. Tokuda , K.i Hirose , N.i Minematsu , A. Yamada , Y. Den , T. Utsuro , S.i Sagayama, "Open-source software for developing anthropomorphic spoken dialog agent," *Proc. of PRICAI-02, International Workshop on Lifelike Animated Agents,* pp.64-69, Aug 2002

13) D. Bitouk, N. Kumar, S. Dhillon, S.K. Nayar, "Face Swapping: Automatically Replacing Faces in Photographs", *ACM Transactions on Graphics (SIGGRAPH 2008)*, vol. 27(3)

14) J. Zhu, L. Van Gool, S. C.H. Hoi, "Unsupervised Face Alignment by Nonrigid Mapping", *ICCV*, 2009

15) U.Frith, *Autism: Explaining the Enigma*. Malden(MA): Blackwell Publishing, 2003, ch. 5,12

16) J. Keenan , 2004, *The Face in the Mirror*, Harper Collins Publishers

17) L. E. Bahrick, L. Moss , C. Fadil, 1996. "Development of visual self-recognition in infancy" *Ecological Psychology*, 8(3):189-208

18) W. Sanefuji , H. Yamashita, H. Ohgami, "Shared minds: Effects of a mother's imitation of her child on the mother-child interaction", *Infant Mental Health Journal*, vol 30(2):145-157, March 2009

19) A.L. Lewy, G. Dawson, "Social Stimulation and Joint Attention in Young Autistic Children", *Journal of Abnormal Child Psychology*, vol 20(6), 1992

20) T.C.J. de Wit, T. Falck-Ytter, C. von Hosfsten, "Young children with Autism Spectrum Disorder look differently at positive versus negative emotional faces", *Res in Autism Spectr Disord* (2008)

21) P. Viola and M. J. Jones, "Robust real-time face detection", *International Journal of Computer Vision*, 57 : 137-154, 2004.

22) P. Rochat, Ch5, *Others in Mind*, Cambridge, 2009

23) B. Amsterdam. "Mirror self-image reactions before the age two". *Developmental Psychology*, 257-305, 1972

24) W. Sanefuji, H. Ohgamu, K. Hashiya, "Preference for peers in infancy", *Infant Behaviour and Development*, Volume 29, Issue 4, 584-593, December 2006

25) L.E. Bahrick, J.S. Watson, "Detection of intermodal proprioceptive-visual contingency as a potential basis of self-perception in infancy". *Developmental Psychology*, 21 , 963-973, 1985

26) L. Uddin, J. Kaplan, I. Szakcs, E. Zaidel , M. Iacaboni, ”Self face recognition activates a fronto parietal gmirrorh network in the right hemisphere: an event-related fMRI study”, *NeuroImage*, 25(3), 926-935, 2005

27) Z. Goa, K. Giovanello, K. Smith, D. Shen, L. Gilmore, ”Evidence on the emergence of the brainfs default network from 2-week-old to 2-year-old healthy pediatric subjects”, *Proceedings of the National Academy of Sciences*, 106(19), 6790-6795, 2009

Acknowledgement

This thesis was prepared in collaboration with Professor Minoru Asada, Associate Professor Hideyuki Nakanishi and Doctor Yoshio Matsumoto. I would like to thank them for reviewing my thesis and giving me constructive criticisms to improve my thesis. Professor Asada provided me a chance to step in this exiting and multi-disciplinary research area. I am very impressed by him and have great respect for his insight and leadership. Associate Professor Hideyuki Nakanishi and Doctor Yoshio Matsumoto pointed out insufficient contents in my drafts. I would like to thank them for their dedicate perusal.

I am also grateful to Assistant Professor Masaki Ogino for his continuous supervision. Not only did he advise me throughout this project, attentive to my interests, but he also provided me support and help me in the improvement of this system, notably with the use of the graphic card computation.

Thanks to Fabio Dalla Libera and Shuhei Ikemoto for teaching me and helping me use the motion capture system.