Universitat
Autònoma
de Barcelona

REAL-TIME FACE TRACKING METHODS

Memòria del Projecte Fi de Carrera
d'Enginyeria en Informàtica realitzat per

Dan Casas Guix

i dirigit per

Maria Vanrell Martorell
Universitat Autònoma de Barcelona

Fernando de la Torre Frade
Carnegie Mellon University

Bellaterra, 29 de Juny de 2009

A l'Adela, al Josep Maria i al Marc,
pel seu suport incondicional.

El sotasignant, Maria Vanrell Vilanova, professora de l'Escola Tècnica Superior d'Enginyeria de la UAB,

CERTIFICA:

Que el treball a què correspon aquesta memòria ha estat realitzat sota la seva direcció per en Dan Casas Guix.

I per tal que consti firma la present.

Signat:

Bellaterra, 29 de Juny de 2009.

El sotasignant, Fernando de la Torre Frade, professor associat de la Carnegie Mellon University, USA,

CERTIFICA:

Que el treball a què correspon aquesta memòria ha estat realitzat sota la seva direcció per en Dan Casas Guix.

I per tal que consti firma la present.

Signat:

Pittsburgh (USA), 29 de Juny de 2009.

Abstract

Face tracking has become an increasingly important research topic in the computer vision field, mainly due to its large amount of real-world situations where such methods can be applied. Although the definition of the problem to be solved is very easy to understand, it is very difficult to come up with a robust solution due to variations in illumination, pose, appearance, etc. Initially, this project gives a brief introduction to the current state-of-the-art of both face detection and face tracking techniques. Moreover, the most important publications in this area, such as the Viola-Jones method for object recognition or the Lucas-Kanade algorithm for the optical flow, are presented. Finally, five different face tracking algorithms are implemented, each of them giving a robust solution for a specific context.

Resumen

Las técnicas de seguimiento facial se han convertido en los últimos años en uno de los temas de investigación más populares en el campo de la visión por computador, principalmente gracias a la gran cantidad de situaciones en el mundo real donde pueden ser aplicadas. Aunque la definición del problema es muy fácil de entender, encontrar una solución robusta resulta ser muy difícil, en parte por culpa de los cambios de ilumunación, pose, apariencia, etc. que se suceden durante una secuencia de vídeo. Inicialmente, este proyecto hace una rápida introducción al estado del arte tanto de las técnicas de seguimiento facial como de las de detección de caras. A continuación, se muestran las publicaciones más relevantes de los últimos años en este campo, como por ejemplo el método para reconocimiento de objectos de Viola y Jones, o el algoritmo de Lucas y Kanade para calcular el flujo óptico. Finalmente, se implementan cinco diferentes algoritmos de seguimiento facial, cada uno de los cuales aporta una solución robusta para un determinado contexto.

Resum

Les tècniques de seguiment facial s'han convertit durant els darrers anys en un dels temes de recerca més populars en el camp de la visió per computador, principalment gràcies a la gran quantitat de situacions on poden ser aplicades en el món real. Tot i que la definició del problema és molt fàcil d'entendre, trobar una solució robusta resulta ser molt difícil, en part per culpa de, per exemple, els canvis d'il·luminació, posició, aparença, etc., que se succeeixen al llarg d'una seqüència de vídeo. Inicialment, aquest projecte fa una ràpida introducció a l'estat del art tant de les tècniques de seguiment facial com de les de detecció de cares. A continuació, es mostren los publicacions més rellevants dels últims anys en aquesta àrea, com per exemple el mètode de reconeixement d'objectes de Viola i Jones, o l'algoritme de Lucas i Kanade per calcular el flux òptic. Finalment, s'implementen cinc diferents algoritmes de seguiment facial, cada un dels quals ofereix una solució robusta per a un determinat context.

Contents

Chapter 1

Introduction

1.1 Motivation

Object tracking has become one of the most popular research fields in computer vision during the last years, mainly because of the large amount of real-world scenarios where such techniques can be used. Face tracking is one of the most challenging problems of the object tracking field, due to the large variability of faces and facial expressions that exist and the number of context where they can be found.

Although many tracking methods and algorithms have been published in the last few decades, nowadays there is still a lot of research to do related to face detection and face tracking. Most of the state-of-the-art methods work well only in some specific situations. Thus, an *universal method* for face detection and tracking has not been found yet.

Exploring this gap of knowledge and the work in progress in this field is the main motivation for this project. The idea of working in something that is currently being researched and not very well solved makes this project both challenging and attractive.

1.2 Goal

This project pretends to be a good introduction to the world of face detection and face tracking, which is nowadays an area being researched by dozens of universities and research institutes around the world.

The most famous tracking methods and algorithms that have been published will be shown, discussed and compared, together with some basic mathematical background needed in order to work with them. Once the current state of the art has been understood, new approaches to face tracking will be suggested.

As Chapter 2 shows, currently the general face detection problem is pretty much solved, though the face tracking is not. In other words: it is not possible to track a face by detecting it in a frame-by-frame basis, due to issues related with occlusions and pose variations that force the failure of the face detector. In order to deal with this handicap, new tracking methods are required.

Therefore, the goal of this project can be summarized in the following points:

- Understanding what face tracking is and why it is important to research.
- Understanding the methods published so far.
- Applying state-of-the-art techniques to develop new approaches to the face tracking problem.

1.3 Framework

This project has been developed during my 12-month stay as a Visiting Scholar at the Carnegie Mellon University, Pittsburgh (USA), where I joined the Human Sensing Lab, leaded by Dr. Fernando de la Torre.

Chapter 2

State of the art

This chapter gives a brief introduction to the origin of computer vision and its current status. Moreover, it presents the two main areas where this project is focused: face detection and face tracking.

2.1 Computer vision

Computer vision is a subarea of the artificial intelligence field which goal is to make computers to understand a given scene or image. To achieve this goal it seeks to apply the theories and models of computer vision to the construction of computer vision systems. Some examples of applications of computer vision systems include process control, event detection, object recognition and pose estimation.

2.1.1 Origins

Computer vision is an immature and diverse field. It is possible to start talking about computer vision from late 1970s. Even though earlier research exists, it was not until then when the technology that enables to process large data sets appeared. Also, these earlier studies usually were originated from other computer science fields, which delayed a few years the standardization of a formulation of "the computer vision problem".

Another important issue of the field of computer vision is the lack of standard formulations for solving the basic problems. This is the reason why an abundance of methods which solve the same well-defined computer vision task have been reported, all with the drawback of being too specific. Furthermore, the inherent specificity of this methods makes it very difficult to apply them in situations that differ much from those to which they were created for.

2.2 Face detection

Face detection is a computer technology that detects the locations and sizes of human faces in digital images. The goal of face detection is to locate the facial features of a person, although there are many applications which focus uniquely in finding the position of the face, rather than finding its specific features.

2.2.1 Challenges in face detection

Any given natural image usually contains a lot more of background patterns than face patterns. This huge difference in amount of patterns makes the desire of finding a face very difficult, because the classifier that decides whether if a given pattern is a face or is not has to be very specific. Figure 2.1 shows a painting made by Octavio Ocampo, and in it is possible to find up to 9 different faces. In this case, even for a human eye this is a tedious work. Of course, this is an extreme difficult context, but it is just an example about how difficult can be to find faces in a picture.

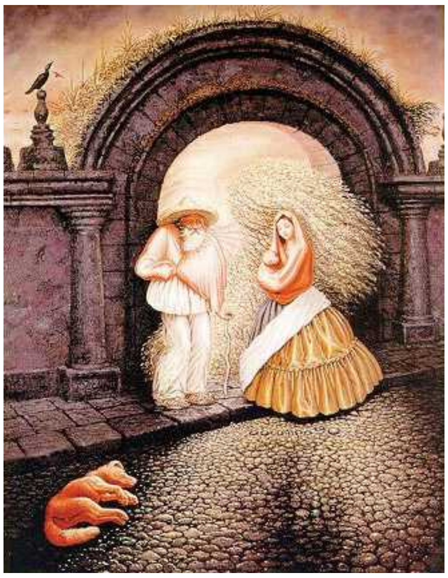


Figure 2.1: Finding faces can be difficult.

Although it is a very hard problem, recent publications have achieved face detection with high accuracy. The most popular method was published in 2001 by Viola and Jones [15] and it is explained in Section 3.2

2.2.2 Methods

Since detecting a face is a very challenging problem, lots of different approaches have been tried to solve it. Although the Viola and Jones [15] method, based in Haar-like features and weak classifiers is currently the most robust, here it is a list of some of the other important methods tried during the last few years.

- **Color based.** This method is based on the color histogram of an image. Initially, a training step is needed in order to learn how the skin color of a face looks like, being afterwards able to classify what is a face and what it is not, according to the range of colors of any given area.

  Although this method seems very interesting because it is face pose independent, it has three very strong constraints: it only works for color images, it does not work well with all kind of skin color and it is not very robust under varying lighting conditions

- **Edge based.** These methods are based on an edge orientation matching followed by a candidate verification using a classifier. [5] is one of the most famous publications that uses this approach. The main handicap of these methods is the difficulty to detect the edges of a face with a complex background. Figure 2.2 shows an example using a Canny edge detector to detect a face. In this example is very clear how important is to have simple backgrounds, being otherwise impossible to distinguish its edges from the face. A simple solution to this problem is provided by a background substraction technique.

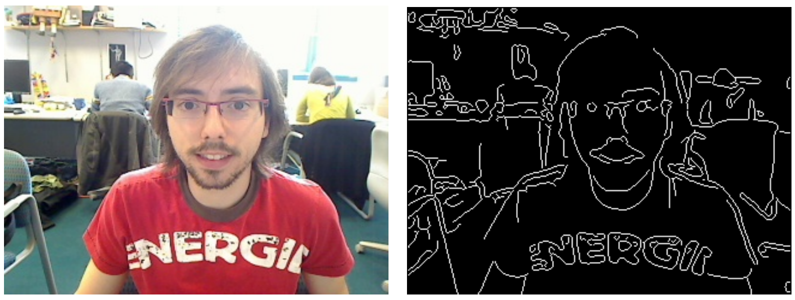


Figure 2.2: Example of edge based face detector using canny edge detection.

- **Feature based.** This includes a huge group of different approaches. Some of them have not been very successful at all, but others are very popular. Their performance highly depends on the kind of feature that is used. For instance, some early approaches use facial features (eyes, mouth, nose, etc.) and they do not obtain good results due to the difficulty to define and compare them. However, other approaches use more sophisticated and person independent features, which result in a more robust and accurate system.

In 2001 Viola and Jones [15] defined the so-called Haar-like features, which are both fast and easy to compute. This method is currently the most used, although it only works for frontal faces with no occlusions.

- **Other methods.** There have been published many other methods, but most of them are no longer used. Among these, it is worth mentioning the technique based on template matching. This method, reported by Lanitis in 1995, was one of the first developed and it consists in deforming an average template of a face, which was hand-coded, using the edges of an image. Its main problem relies on the initialization; in order to detect a face in a given frame, the face template has to be initialized in a frame close to the current one. Moreover, it only works for frontal faces. Figure 2.3 shows an example of its performance.

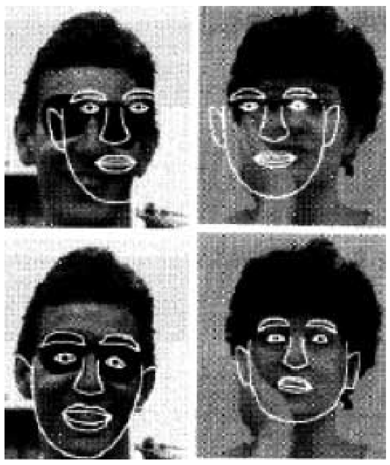


Figure 2.3: Example of face detection using a hand-coded template

Other weak face detection techniques are the so-called knowledge-base. These methods use rules like “the center part of a face has uniform intensity values”, “the difference between the average intensity values of the centre part and the upper part is significant.” or “a face often appears with two eyes that are symmetric to each other, a nose and a mouth”. These knowledge-based methods do not work well, basically because of the difficulty to translate human knowledge into rules precisely: detailed rules fail to detect faces and general rules may find many false positives. Also, it is very difficult to extend this approach to detect faces in different poses.

2.2.3 Applications

Face detection is used in many of different scenarios, with multiple purposes. Here are a few examples of real-world applications:

- **Biometrics.** Video-based face recognition systems require that the data they work with is aligned before it can be compared, in other words, the faces in the images should have the same properties of scale and location. A fast way to achieve this requirement is to detect the faces of the images so that the location and the scale of each of them is automatically found. Once this is done, they all can be aligned and thus have the same properties.
- **Video surveillance.** Since faces are usually the most easily recognizable signature of identity from a distance, face detection can be applied in video surveillance to detect and track people. This can be especially useful in contexts like airports or train stations where the system must work in real-time and it cannot be done manually.
- **Video communications and multimedia systems.** Face tracking can be applied in video communications to estimate the face motion and remove the inter-frame redundancy in video compression schemes like MPEG and H.26x. In multimedia systems like sports video, face tracking can be applied to focus on a region of interest in the image.
- **Human computer interaction.** Human computer interaction, also known as HCI, is the study of the interaction between people and computers. As in many cases this interaction is based on how the user's face is moving, firstly the HCI system needs to detect where the face is, regardless of its three-dimensional position, orientation and lighting conditions.

2.3 Face tracking

Face tracking is a computer technique which goal is to keep the trace of similar detected faces within a video sequence. This is not an easy task because the faces may vary its pose and appearance along the time, which means that what is being track will not look the same in every frame. These changes are the reason why face tracking can not be understood as a “continuous face detection”.

2.3.1 Challenges in face tracking

As it has been mentioned earlier, face tracking is a very challenging problem due to the large appearance variability of a face. This section briefly describes the most important challenges that makes tracking a face a hard problem.

- **Illumination variations.** Illumination variations often lead to a lost of track. Depending on the context where the tracker is running, illumination variations may be very likely. For example, if the camera is situated inside a car trying to track the driver’s head, and suddenly he/she drives into a tunnel, the illumination conditions would change a lot.

  Section 6.4 shows one of the numerous techniques developed in order to deal with this problem. It this case, a histogram equalization enhancement is used in every frame and therefore the video is somehow illumination independent.

  Another well-known method was presented in [6], where the authors propose to use a parameterized function to describe the movement of the image points, taking into account illumination variation by modifying the brightness constancy constraint of optical flow.

- **Pose variations.** In some situations the face that is being tracked is not frontal. These pose changes make the work more challenging; if the face movements are just frontal, applying an affine transformation based tracker, like the one shown in Section 5.2, can be enough to track the face. However, this technique does not work when pose variation occurs. For example, when the subject turns his face to the right, the whole pattern that is being tracked changes, hence, the tracker has to deal with it and follow a new pattern.

  As [14] shows, using Active Appearance Models enables to deal with pose variation, although the results are not totally satisfactory. Another approach to deal with it is presented in [10], where the authors use a Kernel Principal Component Analysis (KPCA) of local parts for pose independent object recognition.

- **Facial deformations.** Tracking faces through changes of expressions is another challenging problem. Figure 2.4 shows an example of facial expression changes.

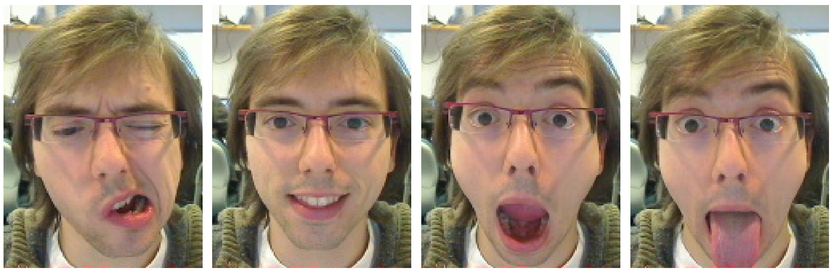


Figure 2.4: Examples of different face expressions.

- **Occlusion and clutter.** As with most tracking problems, occlusion and clutter affect the performance of face trackers. A simple approach to recover from a loss of track is to compute the mean face that has been tracked so far. With this technique, once the occlusion disappears the tracker would be able to find again the face using the mean face. Section 5.4 shows a tracker working with this technique and evaluates its performance.

2.3.2 Applications

Although most of the face tracking applications are the same ones that were explained in Section 2.2.3, where the face detection applications were discussed, in face tracking there are some other ones.

In some contexts a face can not be detected due to, for example, an occlusion, but it can be tracked because we already know where the face was in the previous frame and how it looked like. In this particular scenarios face detection techniques can not be applied, but it is still possible to follow the face using the face tracking ones.

Some of this particular scenarios where face tracking can be used are:

- **Face modeling.** Reconstruction of the 3D model of a face from a video sequence. In this case is clear that at some point the face will get lost if face detection techniques are used, because the camera will go around the face and face detection only works for frontal faces.

- **Driver face tracking.** When tracking a face of a person who is driving, sudden face pose variations are very likely, for example when the driver turns its face to one side in a stop sign. A face tracking system based just in face detection from a frontal camera would lose the face at this point.

Chapter 3

Technical background

This chapter gives an introduction to the whole mathematical and technical knowledge that is required to understand the tracking methods detailed in Chapter 5.

It starts with a basic introduction to the affine transformations, concept required by many of the trackers, and then it continues with more specific techniques and publications, such as optical flow, Lucas-Kanade method, Viola-Jones object detection and template matching, all of them widely used in both face detection and face tracking areas.

3.1 Affine transformation

Affine transformations are used to move and transform images, or parts of them, along the two-dimensional spaces where they are represented.

3.1.1 Definition

Before giving a more technical definition of what an affine transformation is, it is necessary to define two other concepts: vector space and linear transformation.

A vector space is defined as a system consisting of a set of generalized vectors and a field of scalars, having the same rules for vector addition and scalar multiplication as physical vectors and scalars. These vectors have to satisfy certain properties such as associativity, commutativity, identity element, inverse element and distributivity, among others.

A linear transformation is a function between two vector spaces that preserves the operations of vector addition and scalar multiplication. When a translation, which is moving every point a constant distance in a specified direction, is added to a linear transformation, it causes what is known as an affine transformation.

Hence, as equation 3.1 shows, an affine transformation between two vector spaces consists in a linear transformation followed by a translation.

$$x \mapsto Ax + b. \tag{3.1}$$

In a finite-dimensional space, an affine transformation is given by a matrix A and a vector b, and preserves the collinearity relation between points and the ratio of distances along a line.

3.1.2 Representation with homogeneous coordinates

As it has been just mentioned, an affine transformation is composed by a linear transformation and a translation. To represent such operations, ordinary vector algebra says that using matrix multiplication is possible to represent linear transformations, and for the translations vector addition is used.

To represent both operations in just one, homogeneous coordinates are used. This representation requires that a "1" is added at the end of all vectors, and all matrices are augmented with an extra row of zeros at the bottom, an extra column, which is the translation vector, to the right, and a "1" in the lower right corner. Being A a matrix, the result will look like

$$\begin{bmatrix} \vec{y} \\ 1 \end{bmatrix} = \begin{bmatrix} A & \vec{b} \\ 0, \ldots, 0 & 1 \end{bmatrix} \begin{bmatrix} \vec{x} \\ 1 \end{bmatrix} \tag{3.2}$$

which is equivalent to

$$\vec{y} = A\vec{x} + \vec{b}. \tag{3.3}$$

As it has been shown, the use of homogeneous coordinates makes possible to combine any number of affine transformations into one by multiplying matrices. This fact makes affine transformation very attractive. Among other advantages, it speeds up a lot its computational time.

Equation 3.5 shows a more specific example where a translation, t_x and t_y, is added to a given point.

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}. \tag{3.4}$$

which is equivalent to

$$\begin{aligned} x' &= x + t_x \\ y' &= y + t_y \end{aligned} \tag{3.5}$$

In other words, each coordinate has been increased by its translation factor. Another example, now showing how to rotate θ degrees is shown below.

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}. \tag{3.6}$$

Finally, a more complex example that shows how to integrate rotation, translation and scale in one single affine transformation.

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} s_x \cdot \cos\theta & s_y \cdot \sin\theta & t_x \\ -s_x \cdot \sin\theta & s_y \cdot \cos\theta & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}. \tag{3.7}$$

3.1.3 Practical examples

For a better understanding on how the affine transformations are going to be used, this section presents a practical example where a square represented in a two-dimensional space is transformed using affine transformations. Later on, in Chapter 5, a very similar approach will be applied in order to warp images and faces during the tracking process.

In this particular case, shown in Figure 3.1, we want to rotate the original square (blue) $\frac{\pi}{8}$ and increase its size 1.5 times. It must be taken into account that in order to rotate the square it is first necessary to bring it to the origin.

Being B_1 the original square in homogenous coordinates,

$$B_1 = \begin{bmatrix} 2 & 2 & 3 & 3 \\ 2 & 3 & 3 & 2 \\ 1 & 1 & 1 & 1 \end{bmatrix} \tag{3.8}$$

it can be brought to the origin using the affine transformation A_1

$$A_1 = \begin{bmatrix} 1 & 0 & \frac{B_1(1,2)+B_1(1,3)}{2} \\ 0 & 1 & \frac{B_1(2,1)+B_1(2,2)}{2} \\ 0 & 0 & 1 \end{bmatrix} \tag{3.9}$$

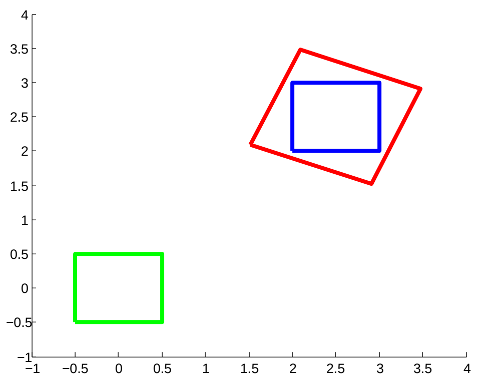


Figure 3.1: Rotating and scaling a square using affine transformations

which result is B_2, and it appears in figure 3.1 as a green square.

$$B_2 = A_1 \cdot B_1 = \begin{bmatrix} -0.5 & -0.5 & 0.5 & 0.5 \\ -0.5 & 0.5 & 0.5 & -0.5 \\ 1 & 1 & 1 & 1 \end{bmatrix} \tag{3.10}$$

Once the center of the square is in the origin, it is possible to rotate and scale it using the affine transformation matrix A_2

$$A_2 = \begin{bmatrix} 1.5 \cdot \cos(\frac{\pi}{8}) & 1.5 \cdot \sin(\frac{\pi}{8}) & 0 \\ -1.5 \cdot \sin(\frac{\pi}{8}) & 1.5 \cdot \cos(\frac{\pi}{8}) & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{3.11}$$

Finally, using another affine transformation matrix, the square is brought back to its original location. The red square in Figure 3.1 is the final result.

3.2 Viola-Jones object detection

This section gives a technical description about another important requirement when tracking a face: how to detect a face. As it has already been said in Section 2.2.1, detecting faces in images is a tedious and hard job, but after the development of many different approaches with poor performance, in 2001 Paul Viola and Michael Jones came up with a solution [15].

With their revolutionary method, Viola and Jones were able to detect frontal faces in images in real-time, with a very low false positive rate. The only drawback was the previous training step before running the classifier, which is very slow but it only have to be run once.

To understand how their algorithm works, firstly it is necessary to review a few other concepts, some of them also introduced by them, like Haar-like feature, "integral image" and cascade classifier.

3.2.1 Haar-like features

Historically, the most common feature set used for digital images was the intensity, for example, the RGB value of any given pixel, but such a set made the task computationally expensive. In 1998 a publication by Papageorgiou et al [12] introduced an alternative feature set based on rectangular regions of the image and its addition. The idea was, somehow, to categorize the images depending on the sum of the value of its pixels in a particular area.

A few years later, in 2001, Viola and Jones proposed in their paper a modification about how to compute this Haar-like features. They defined what is called the "2 rectangle Haar-like feature", although they also defined features with 3 and 4 rectangles. In their approach they defined Haar-like features composed by two rectangles, and its value was the difference of the sum of pixels of each rectangle. Using this values, it is possible to indicate certain characteristics of a particular area of the images, such as edges or changes in texture. Figure 3.2 shows some of the original features used in their paper.

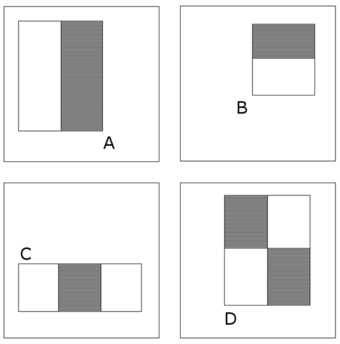


Figure 3.2: Some of the features used in the original Viola-Jones paper.

Another new concept that Viola and Jones introduced was an ultra fast way to compute the sum of any rectangular area in the image, at any position or scale, called "integral image".

3.2.2 Integral image

In order to compute very fast the value of each rectangle feature, Viola and Jones introduced the concept of "integral image". The integral image at the location x,y contains the sum of the pixels above and to the left of x,y, inclusive. Equation 3.12 shows its mathematical expression,

$$ii(x,y) = \sum_{x' \leq x, y' \leq y} i(x',y') \tag{3.12}$$

where *ii(x,y)* is the integral image and *i(x,y)* is the original image.

Figure 3.3 shows a practical example about how it works, and how the value of any given square can be computed with just four array references. The value of the integral image at location 1 is the sum of the pixels in rectangle A. The value at location 2 is A + B, at location 3 is A + C, and at location 4 is A + B + C + A.

Hence, the sum within D can be computed as 4 + 1 - (2 + 3) and clearly the difference between two rectangular sums can be computed in eight references. Since the two-rectangle features defined in Figure 3.2 involve adjacent rectangle sums they can be computed in six array references, eight in the case of the three-rectangle features, and nine for four-rectangle features.

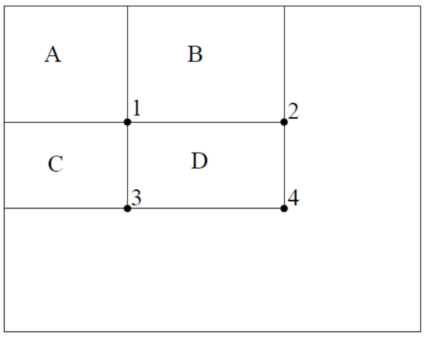


Figure 3.3: The sum of the pixels within rectangle D can be computed with just four array references.

3.2.3 Feature selection

Once Haar-like feature has been defined, and it is clear how integral images enable to compute their value in a fast way, the following step is apply these concepts to the learning set of the data base. The idea would be to compute the values for each feature in all the possible positions in every positive example, but this would be prohibitively expensive to compute.

Because of this huge number of possible features, it is needed to perform a feature selection process, during which only the most important ones will be selected. This selection is made by an AdaBoost classifier, which in each round of boosting selects one feature. After this classification, it is possible to check which features has been selected for being more important than the other, and how they look like.

Figure 3.4 shows the first two features selected by Viola and Jones in their paper from 2001, and it is easy to notice that the first one measures the difference in intensity between the regions of the eye and the regions across the upper cheeks, because the eye area is often darker than the cheeks.

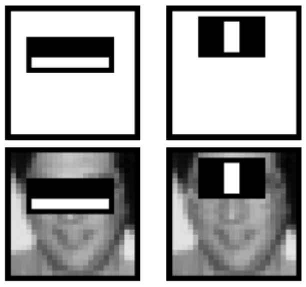


Figure 3.4: First and second more important selected features selected by Viola and Jones[15].

3.2.4 Cascade classifier architecture

One of the most relevant characteristics in this publication was that the classifier was able to run in real time, at about 15 frames/second in a 700MHz CPU. In other words, the system could analyze up to 15 images per second, detecting all the frontal faces that appear in each of them.

To achieve such performance, they used what is called a cascade classifier, which consists of a series of classifiers arranged in a cascade in order of complexity, where each successive classifier is trained only on those examples which pass through the preceding classifiers. When any of these classifiers rejects the image that is being classified, this one does not goes further and the classifier starts over again with another sample.

With all the techniques detailed in this section, Viola and Jones were able to detect up to 95% of the frontal faces in images, with a very low false positive rate.

3.3 Template matching

Template matching is a technique used in digital image processing for comparing portions of images between them, *sliding* the patch or portion over the input image using one of the methods described in Section 3.3.1. Once the patch has been tested in all the possible locations in a pixel-by-pixel basis, a matrix containing a numerical index according to how good the patch matches in each location is created.

In order to see how this matrix looks like, Figure 3.6 shows the result after applying the template matching technique to the images shown in Figure 3.5. In this case, the template image contains the yellow cartoon character that appears in the upper left corner of the original image.

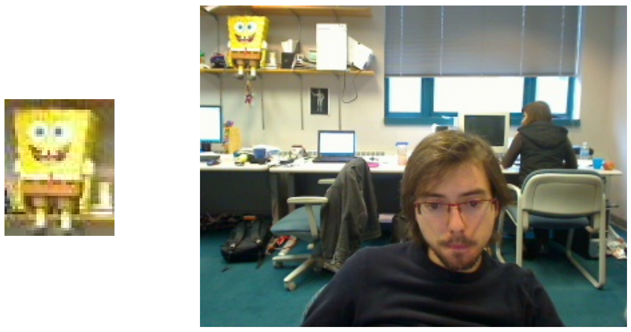

(a) Template (b) Image where to look for the template

Figure 3.5: Input images for template matching.

It is easy to see that where the template best fits into the image is somewhere in the upper left corner of the plot shown in Figure 3.6. To know the exact location of the best match, it is just necessary to look for the coordinates of the red peek, and by using them, locate the template in the original image.

Using this technique can be useful when the object to be tracked, for instance, a face, is moving along the scene but always keeping the same appearance. If the object changes its appearance, the matrix resulting after applying the template matching will not have that important peek because there will not be a very clear location where the object best matches, and this will lead to errors.

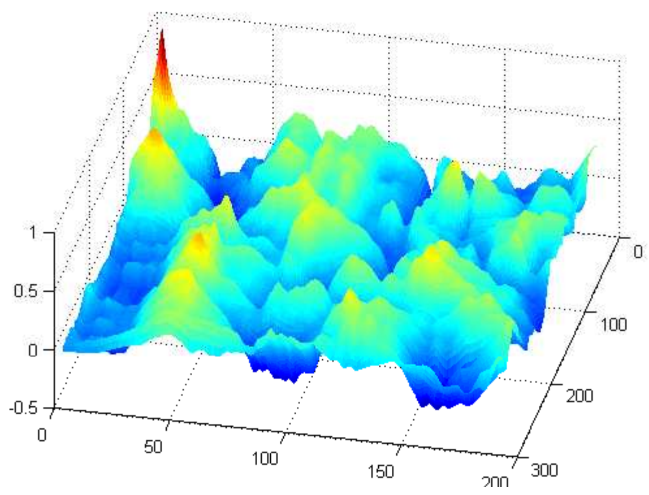


Figure 3.6: Output matrix of the template matching, the highest peek is where the patch best fits.

3.3.1 Template matching methods

The "matching error" between the patch and any given location inside the image where this is being searched can be computed using different methods. This section gives a brief description of each of them.

In the following mathematical expressions, I denotes the input image, T the template, and R the result.

- **Square difference matching methods.** These methods match the squared difference, which means that the perfect match would be 0 and bad matches would lead to large values. Equation 3.13 shows its mathematical expression.

$$R_{sq\_diff}(x,y) = \sum_{x',y'} [T(x',y') - I(x+x', y+y')]^2 \tag{3.13}$$

- **Correlation matching methods.** These methods multiplicatively match the template against the image, which means that a perfect match would be the largest. Equation 3.14 shows its mathematical expression.

$$R_{ccorr}(x,y) = \sum_{x',y'} [T(x',y') \cdot I(x+x', y+y')]^2 \tag{3.14}$$

- **Correlation coefficient matching methods.** These methods match a template relative to its mean against the image relative to its mean. The best match would be 1 and the worst one would be -1. Value 0 means that there is no correlation. Its mathematical expressions are shown below.

$$\begin{aligned} R_{ccoeff}(x,y) &= \sum_{x',y'} [T'(x',y') \cdot I'(x+x',y+y')]^2 \\ I'(x+x'.y+y') &= I(x+x',y+y') - \frac{1}{(w \cdot h) \sum_{x'',y''} I(x+x''.y+y'')} \\ T'(x',y') &= T(x',y') - \frac{1}{(w \cdot h) \sum_{x'',y''} T(x''.y'')} \end{aligned} \tag{3.15}$$

- **Normalized methods.** It is also possible to normalize each of the three methods that have been just described. It is useful to normalize if there is interest in reducing the effect of lighting differences between the template and the image. In every case, the normalization coefficient is the one shown in Equation 3.16

$$Z(x,y) = \sqrt{\sum_{x',y'} T(x',y')^2 \cdot \sum_{x',y'} I(x+x',y+y')^2} \tag{3.16}$$

3.4 Good features to track

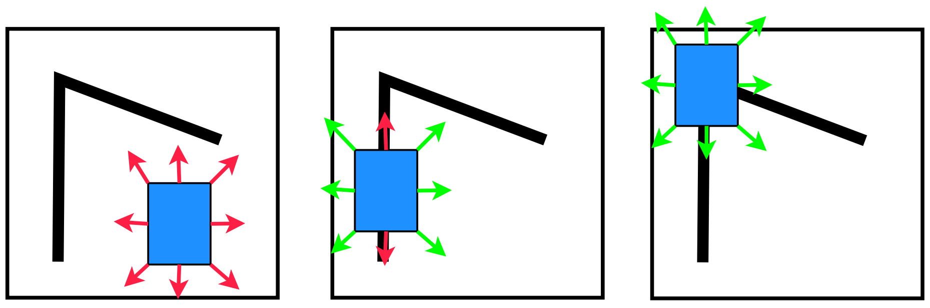
Besides template matching, there are plenty of other ways of tracking objects in a video sequence. In many of them the interest will be in finding parts, or even single pixels, from the previous frame that are recognizable in the present frame. The key questions here are two: which parts are easier to track and how they can be detected.

It is easy to realize that not all the points in a given frame are equally easy to track. For example, if a point on a large white wall is picked, it will be hard to track in the following frame because the whole wall looks the same. On the other hand, if the picked point is unique, as for example the edge of the wall could be, it might be not very hard to track in the following frame.

Thinking in a more technical way in how this unique points can be detected can lead to conclude that looking for points that have a significant change, for example a strong derivative, is a good idea. Indeed, this is a good start but not the definite solution. Usually, a point with a strong derivative belongs to a sort of edge in the image. However, finding the points of the edges is not enough because these points look very similar to each other. It will be thus necessarily to find the most relevant points among them: the corners.

3.4.1 Harris approach

One of the most common definitions of what a corner is was defined by Harris [7] in 1988. In his paper Harris said that a unique point in any given picture has to be invariant to translation, rotation, scale and lighting. For this reason, he defined three kinds of regions: flat, edge and corner. As Figure 3.7 shows, a flat region does not present any change in any direction, an edge region does not change along the edge direction, and finally a corner region shows significant changes in all the directions. In the figure, arrows are green if some change happens when the window is moved in this directions and red otherwise.



(a) Flat (b) Edge (c) Corner

Figure 3.7: The kinds of pixels defined by Harris, regarding its location in the image.

Harris' definition relies on the matrix of the second-order derivatives $(\partial^2 x, \partial^2 y, \partial x \partial y)$ of the image intensities. It is possible to create new "second-derivative images" from the second-order derivatives of images, taken at all the points in the image, or when combined together, creating a new "Hessian" image. The terminology that has been used comes from the definition of the Hessian matrix around a point, which is showed in Equation 3.17

$$H(f) = \begin{bmatrix} \frac{\partial^2 I}{\partial x^2} & \frac{\partial^2 I}{\partial x \partial y} \\ \frac{\partial^2 I}{\partial x \partial y} & \frac{\partial^2 I}{\partial y^2} \end{bmatrix} \tag{3.17}$$

Moreover, Harris defined what is called *autocorrelation matrix* of the second derivative images over a small window around each point as follows

$$M(x,y) = \begin{bmatrix} \sum\limits_{-K \le i,j \ge K} w_{i,j} I_x^2 & \sum\limits_{-K \le i,j \ge K} w_{i,j} I_x I_y \\ \sum\limits_{-K \le i,j \ge K} w_{i,j} I_x I_y & \sum\limits_{-K \le i,j \ge K} w_{i,j} I_y^2 \end{bmatrix} \tag{3.18}$$

where I_x and I_y are partial derivatives of I. According to Harris' definition, corners are places in the image where the autocorrelation matrix of the second derivatives has two large eigenvalues. It basically means that there is texture, or edges, going in at least two separate directions, just as real corners have at least two edge meetings in a point. If λ_1 and λ_2 are the two eigenvalues of a given point:

- If $\lambda_1 \approx 0$ and $\lambda_2 \approx 0$ the point corresponds to a flat region, as Figure 3.7(a) shows.
- If $\lambda_1 \approx 0$ and λ_2 has some large positive value, the point corresponds to a edge region, as Figure 3.7(b) shows.
- If λ_1 and λ_2 have large positive values, the point corresponds to a corner region, as Figure 3.7(c) shows. Thus, this is going to be a good point to track.

The fact of using the eigenvalues of the autocorrelation matrix has also the advantage of working with quantities that are invariant to rotation, which is important because the object that are going to be tracked might rotate while they move.

The final step in this Harris method consists in analyzing the eigenvalues λ_1 and λ_2 that have just been found, to decide the corner strength, using Equation 3.19.

$$M_c = \lambda_1 \lambda_2 - \kappa \left(\lambda_1 + \lambda_2\right)^2 = \det(A) - \kappa \operatorname{trace}^2(A) \tag{3.19}$$

3.4.2 Shi and Tomasi approach

In 1994, a few years after the Harris' publication [7], Jianbo Shi and Carlo Tomasi [13] found out a way to improve the method. Instead of using the Equation 3.19 to compute the strength of a corner, they realized that it was enough comparing the small of the two eigenvalues λ_1 and λ_2 with respect to a minimum threshold. The results using this new approach were not only sufficient but also much more accurate than the Harris one.

3.4.3 Examples of good features

In Chapter 5, some of the tracking methods use the good features to track technique. This section shows a few examples to see the performance of the implementation of the Shi and Tomasi approach that will be used later on, starting with a toy example to check if it really works with an input image where the result is trivial, and then moving to more complex tests.

Figure 3.8 shows the performance after analyzing a simple chess table, where the red dots indicate the good features to track found by the Shi and Tomasi. As expected, in this toy example the good features to track are located at the edge of every square, and nowhere else.

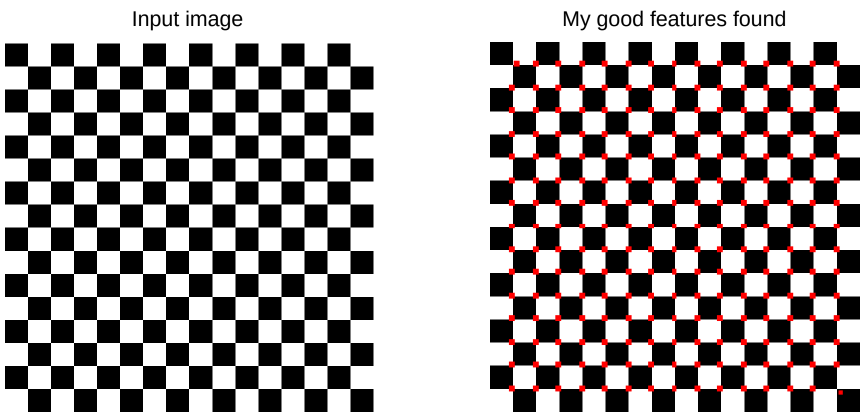


Figure 3.8: Good features to track of a chessboard. Obviously, the edges give more information than any other point.

Once it is clear that the implementation of this algorithm works, it is time to try it with a more complex example. It is also important to notice that in the previous toy example both Shi and Tomasi method and Harris method gave the same output, but this will not always happen.

A more complex example is shown in Figure 3.9, where the input image is a face. In this particular case the Shi and Tomasi method was used with three different quality level values, or thresholds. It is easy to notice that the lower the threshold is, the higher it is the amount of good features selected, because the threshold indicates the minimal acceptable lower eigenvalue for a point to be included as a corner.

Figure 3.10 shows the same input image being analyzed by the Harris method. In the case the κ value was 0.04 and the threshold goes from 0.001 to 0.01.

In these examples, the performance of both methods is very similar. After running a few more tests, it is possible to extract as a conclusion that tuning the parameters will be an important issue when developing a tracker. In the examples that have been just discussed, the best option for both methods would be the middle one, because those detect perfectly areas such as the eyes, the glasses, the nose and the outline of the face, that obviously are easier to track than, for example, points in the middle of cheek.

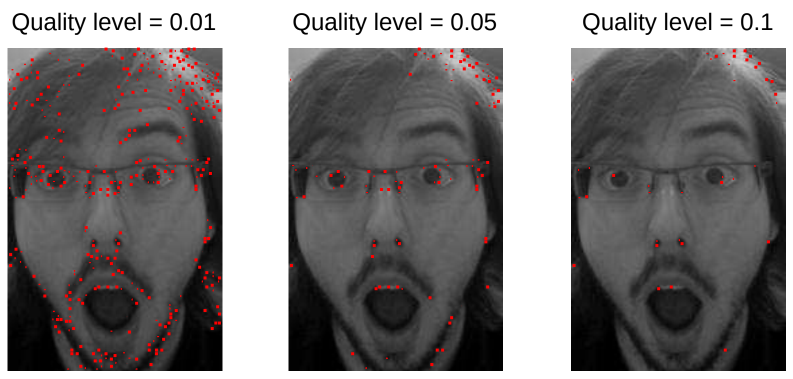


Figure 3.9: Good features found using the Shi and Tomasi method.

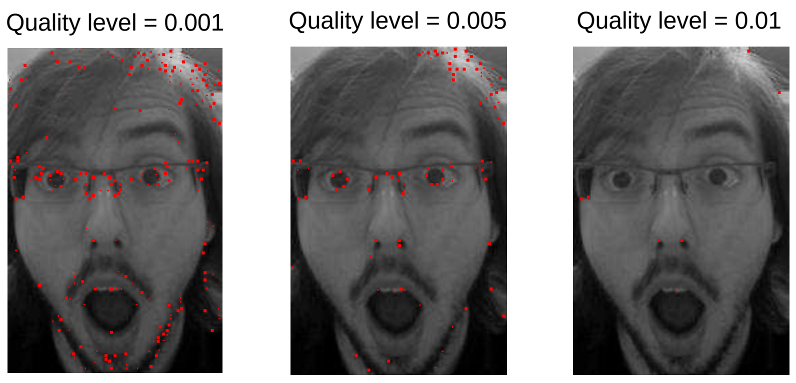


Figure 3.10: Good features found using the Harris method.

It is also very important to notice how relevant is the illumination is this algorithm. For example, the upper right corner of the input image in Figures 3.9 and 3.10 has more light than the other parts and, because of that, the algorithm founds more edges. This is something that has to be taken into account when selecting the final good features.

Finally, let's check if the implemented algorithm is really rotation invariant, as theoretically suppose to be. This is a key feature for the future tracker, because the objects to be tracked along the video, a face in this case, may rotate to one side along the sequence.

Figure 3.11 shows the result after running the Shi and Tomasi edge detector in the same image, but with a different rotation each time. Although the results are not completely similar in all the cases, it is clear that the edge detector works good with any given rotation. Also, as it has been mentioned earlier, it is very remarkable how important is the illumination in this algorithm.

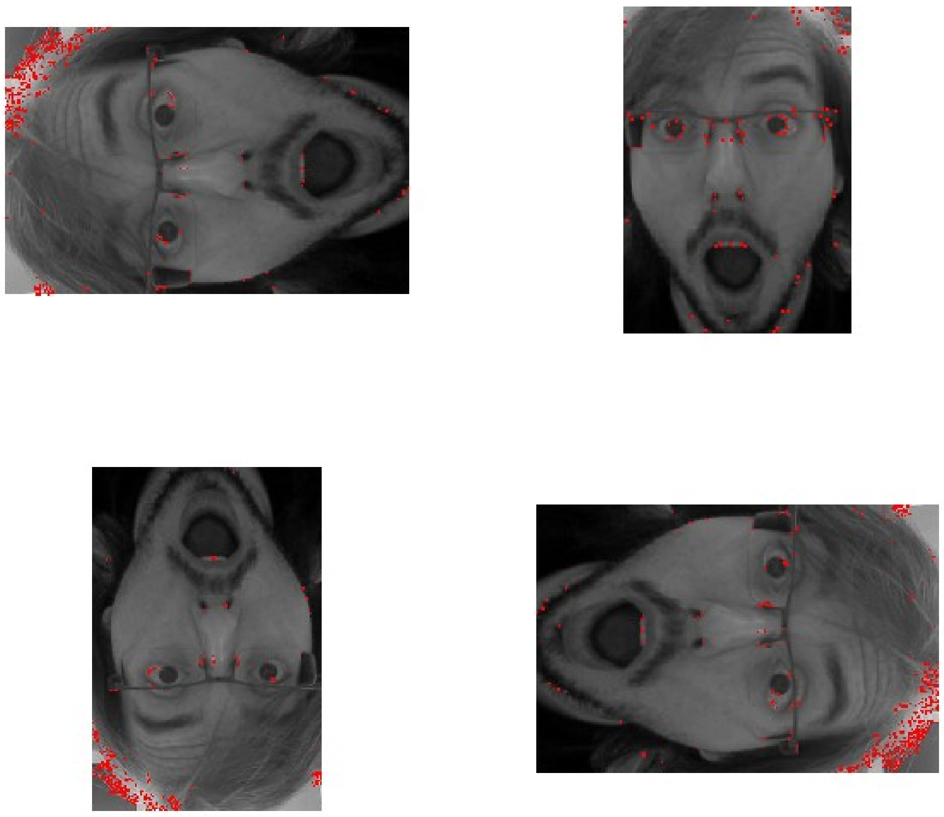


Figure 3.11: Results after running the Shi and Tomasi edge detector in the same image, but with different rotations.

3.5 Optical flow

Tracking consists in figuring out how the things are moving, and generally without any prior knowledge about the content of any given frame. Hence, detecting the motion between two frames is one of the requirements to develop a tracker.

It is possible to associate some sort of velocity with every pixel in an image or, in other words, some displacement that represents the distance that a pixel has moved between the previous frame and the current frame, and this is exactly what *dense optical flow* means.

Although the theory looks fantastic, in practice to calculate the dense optical flow is not easy. As it has been discussed in Section 3.4, not all the pixels in an image are equally difficult to track. Hence, compute the *dense optical flow*, which means take into account every single pixel, is a hard task. A possible solution is to apply an interpolation to the pixels that are hard to track, using the information from the ones that can be well tracked. This solution leads to a higher computational cost though.

The other solution is called *sparse optical flow*, and it relies in specifying the subset of points to be tracked before running the optical flow algorithm. This subset is composed by points that are good to track, such as the corners detected by the Harris or Shi and Tomasi methods earlier discussed in Section 3.4.

Even though it is not possible to work in real-time with dense optical flow because of its computational cost, in some context there is more interest in getting an excellent visual quality rather than a fast performance, for example in the movie production. One of the most dense flow methods used nowadays was published by Black and Anadan [1] in 1993, but it is out of the focus of this project.

3.5.1 Lucas-Kanade method

The Lucas-Kanade algorithm has become one of the most important sparse optical flow techniques, mainly because it can be easily applied to a subset of points in the input image.

This method relies only on the local information that comes from the surrounding area of each point of interest. The drawback of using such small windows is that in some situations large motions can move points outside of the local windows and they thus become impossible to track. To fix this problem, the *pyramidal* Lucas-Kanade algorithm was developed, which tracks starting from low level detailed images and working down to higher detailed ones. Image pyramids allow that large motions can be detected by local windows.

The main idea of the Lucas-Kanade algorithm is based on three assumptions:

- **Brightness constancy.** A pixel of an object from a video sequence does not change its appearance from frame to frame. In other words, for gray-scale images it is assumed that the brightness of a pixel does not change, which means that it keeps its value along the time.

  Being $I(x,t)$ the intensity of the pixel x at time t, the mathematical expression of this assumption is:

$$f(x,t) = I(x(t),t) = I(x(t+dt),t+dt) \tag{3.20}$$

which means, as Equation 3.21 shows, that the tracked pixel intensity does not change over time.

$$\frac{\partial f(x)}{\partial t} = 0 \tag{3.21}$$

- **Temporal persistence.** Also known as "small movements", it assumes that the image motion of a surface patch changes slowly. It is possible to view this change as approximating a derivative of the intensity with respect to time. To better understand this assumption, firstly a simple one-dimensional case will be shown.

  Using the brightness constancy assumption that has been just described, substituting the definition of brightness $f(x,t)$ while taking into account the implicit dependence of x on t, $x(x(t),t)$, and then applying the chain rule for partial differentiation showed in Equation 3.22,

$$\frac{dz}{dt} = \frac{\partial f}{\partial x}\frac{dx}{dt} + \frac{\partial f}{\partial y}\frac{dy}{dt} \tag{3.22}$$

  where $z = f(x,y)$, it yields to the following expression:

$$\underbrace{\left.\frac{\partial I}{\partial x}\right|_{t}}_{I_x} \underbrace{\left(\frac{\partial x}{\partial t}\right)}_{v} + \underbrace{\left.\frac{\partial I}{\partial t}\right|_{x(t)}}_{I_t} = 0 \tag{3.23}$$

  where I_x is the spatial derivative across the first image, I_t is the derivative between images over time, and v is the velocity to be found. Hence, now it is possible to define a simple equation for the optical flow velocity for the one-dimensional case as follows:

$$v = -\frac{I_t}{I_x} \tag{3.24}$$

  Figure 3.12 shows an example in the one-dimensional scenario that has been just described, where an "edge" moves along the x-axis from the time t until the time $t+1$, and the velocity v has to be found.

  Figure 3.13 shows how the measurement of the velocity is done, taking into account the first two assumptions, brightness constancy and small motion, highlighting how the spatial derivative I_x and the temporal derivative I_t are represented.

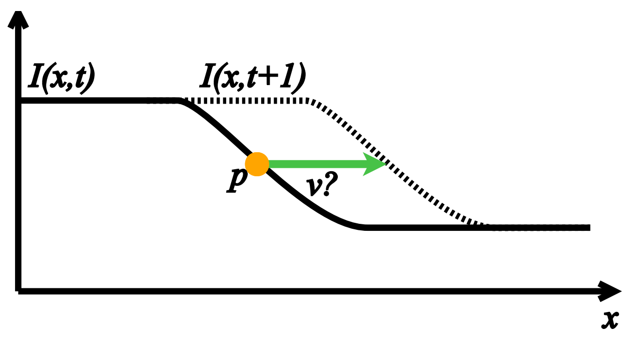


Figure 3.12: Lucas-Kanade optical flow on one-dimensional space problem.

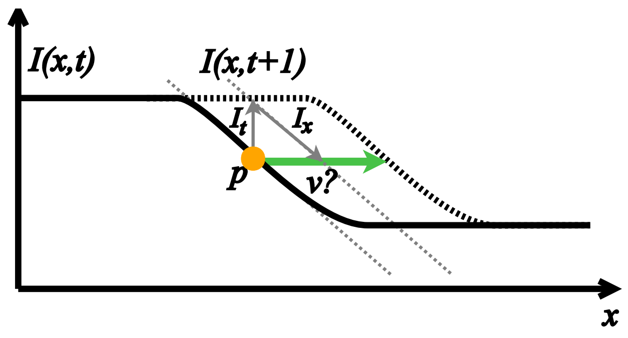


Figure 3.13: I_x and I_t representation in the Lucas-Kanade optical flow on a one-dimensional space.

Now that the one-dimensional case has been discussed, it is time to move on to the two-dimensional one, because it is the one needed for representing images. Firstly another coordinate and its velocity, are added to the Equation 3.23. The new equation looks as follows

$$I_x u + I_y v + I_t = 0 \tag{3.25}$$

where u and v are the x and y components of the velocity, respectively. The problem of Equation 3.25 is that it has two unknowns for any given pixel. This problem is called *aperture problem* and it occurs as a consequence of the ambiguity of one-dimensional motion of a simple striped pattern viewed through an aperture. Figures 3.5.1 and 3.5.1 show an example.

In order to solve this problem, the last optical flow assumption is used.

- **Spatial coherence.** It assumes that neighboring points that belong to the same surface have similar motion and project to nearby points on the image plane. In other words, if a local patch of pixels have the same motion, then

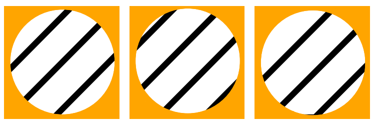


Figure 3.14: Aperture problem: it is not possible to know how the lines are moving just looking through the hole. To know the correct answer it is necessary to see the end of the lines.

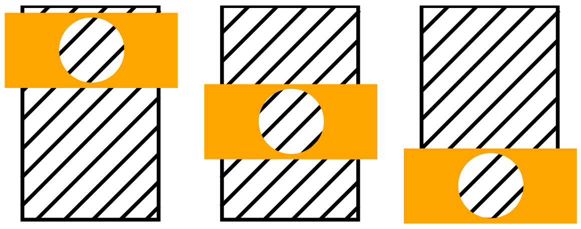


Figure 3.15: Aperture problem: here it is clear how the lines are moving. Even though in Figure 3.5.1 seemed that the flow was going down and to the right, here it is clear that lines are moving upwards.

it is possible to find the motion of the central pixel by using the surrounding ones. Taking this into account, now Equation 3.25 can be solved.

The equation 3.25 can also be written as

$$\nabla I^T \cdot \mathrm{u} = -I_t \tag{3.26}$$

where $\mathrm{u} = \begin{bmatrix} u \\ v \end{bmatrix}$ and $\nabla I = \begin{bmatrix} I_x \\ I_y \end{bmatrix}$, and if, for example, a 5-by-5 pixels window is used around a given pixel to compute its motion, it is possible to set up 25 equations as follows.

$$\underbrace{\begin{bmatrix} I_x(p_1) & I_y(p_1) \\ I_x(p_2) & I_y(p_2) \\ \vdots & \vdots \\ I_x(p_{25}) & I_y(p_{25}) \end{bmatrix}}_{A} \underbrace{\begin{bmatrix} u \\ v \end{bmatrix}}_{d} = - \underbrace{\begin{bmatrix} I_t(p_1) \\ I_t(p_2) \\ \vdots \\ I_t(p_{25}) \end{bmatrix}}_{b} \tag{3.27}$$

This is an over-determined system of equations, and it can be solved using the least-square method, which will compute an approximated solution. Hence, to solve the equation the following expression is used

$$(A^T A)d = A^T b \tag{3.28}$$

which in this case will look as follows:

$$\underbrace{\begin{bmatrix} \sum I_x I_x & \sum I_x I_y \\ \sum I_x I_y & \sum I_y I_y \end{bmatrix}}_{A^T A} \begin{bmatrix} u \\ v \end{bmatrix} = - \underbrace{\begin{bmatrix} \sum I_x I_t \\ \sum I_y I_t \end{bmatrix}}_{A^T b} \tag{3.29}$$

and which solution is:

$$\begin{bmatrix} u \\ v \end{bmatrix} = (A^T A)^{-1} A^T b \tag{3.30}$$

It is important to highlight that Equation 3.30 has solution only when $(A^T A)$ is invertible, and this happens only when it has two large eigenvectors. Hence, it is important that the center of the tracking window is a corner, because, as it was discussed in Section 3.4, corners are the only regions of an image that fulfil such requirements.

In other words, the subset of pixels that are used to compute the optical flow are the ones found by the good features to track.

Chapter 4

Work environment

To achieve the goals of this project, it is firstly necessarily to select which tools are going to be used. This is a key step because without the proper tools many additional difficulties will appear while developing the methods.

After a research period, where the tools used in similar projects were tested and evaluated, a decision was made. The two following sections explain the reasons why such tools were selected, and how they work, splitting them in two subgroups: hardware and software.

4.1 Software

Taking into account that this project belongs to the fields of computer vision, machine learning and image processing, it was quite clear that the implementation had to be done using MATLAB and C++, complemented by the OpenCV library, because these are the main tools for such areas. These tools have been widely used in the last few years, and are in a constant development. Other less trustable tools might lead to errors.

- **C++**

  This general-purpose programming language offers both robustness and high execution speed, which are key characteristics for a real-time face tracker. Even though some more specific tools for dealing with computer vision algorithms exist, which are much faster to code, C++ offers a higher performance due to its higher execution speed. It is important to keep in mind that to achieve real time it is more critical the execution speed rather than the time needed to code the algorithm.

Another remarkable detail about the C++ is the environment that will be used. In this case the Microsoft Visual C++ 2008 Express Edition was selected, basically because it runs in Microsoft Windows, it offers all the characteristics that this project requires, such as compiler, editor an debugger, and it is free to download.

- **OpenCV**

  OpenCV, which stands for Open Source Computer Vision Library, is a computer vision library originally developed by Intel, but now it is open to everyone. It is mainly focused on real-time image processing and it is a cross-platform library, which means that it can be used in many different operating systems. OpenCV includes implementations of the most famous computer vision algorithms, some of them commented in Chapter 3, which will be very useful in this project.

  OpenCV is ready to be used in C++ just including its libraries in the source code but, in this project, there is also interest in using OpenCV from MATLAB, and this is going to be done by a *MEX* compiler. MEX-files, which stands for MATLAB Executable files, are dynamically linked subroutines produced from C source code that, when compiled, can be run from MATLAB as of they where normal MATLAB functions. The main reasons to use MEX-files are speed, because they are much more efficient than normal MATLAB code, and the advantage of not having to rewrite existing C++ code.

- **MATLAB**

  MATLAB is a numerical computing environment that allows easy matrix manipulation, plotting of functions and data, and implementation of algorithms. It also has a lot of *toolboxes*, such as image capturing, that makes very easy to work with images. The main advantage of MATLAB is that it has a very high-level programming language which makes it very faster to program but, of course, it has the drawback of being much slower than C++ regarding the execution speed. Here is where the MEX-files mentioned in the previous point have an important role, because through them it is possible to speed up a bit the MATLAB execution speed when running OpenCV code compiled as a MEX.

  Basically, MATLAB will be used to run early tests of the algorithms implemented in C++ code through MEX-files, because its interface and high level language programming are perfect when the execution speed is not important, but the result is. Once the tests have finished, the whole algorithm will be written in C++ to get a more robust implementation.

4.2 Hardware

The hardware used in this project consists mainly in webcams employed to capture the videos to be tracked, and personal computers to process the data and run the algorithms.

Webcams

There are two main groups of cameras that have been used.

- **IEEE-1394 based digital cameras**: These cameras are the ideal solution for acquiring high quality images with a regular laptop or PC. Another reason for using such cameras is the famous CMU 1394 Digital Camera Driver, developed by researchers from the Carnegie Mellon University and available to the general public for free. This driver works with any webcam connected to a PC through the IEEE-1394 port, also know as a *firewire*. The fact that this project was developed in the Carnegie Mellon facilities and the existence of this driver was a good combination.
- **USB based digital cameras**: Although IEE-1394 based digital cameras seem to be the best way to get the images that are being tracked, USB cameras are cheapest and much more common and this is something that has to be taken into account when developing a project.

  Regarding the drivers, OpenCV library provides a set of functions to directly capture images from a USB camera. Hence, no special libraries are required and the software will work in any PC with any USB camera.

PC

Since there are not any special requirements regarding the PC hardware, a normal domestic one will suffice. To give an example, most of the tests of this project were run in a PC with a Pentium D 3.2GHz and 2GB DDR2 RAM memory.

Chapter 5

Practical part

This chapter explains the implementation of many different face tracking methods, based on the knowledge given in Chapter 3. For each method, firstly the main algorithm will be detailed, and then the pros and cons will be discussed. It is also important to notice that each tracker method will be implemented to run in a specific context, which means that each of them has a different goal to achieve. For instance, some of them are focused just in frontal movements, because there might be interest in knowing the exact rotation of a face, while some others are able to track the face, whatever movement it makes, but without being able to compute the exact rotation.

5.1 Template matching based face tracker

This is the first and a very simple approach to face tracking, totally based on the template matching technique detailed in Section 3.3. The only reason why this simple technique is included in this chapter is to compare its performance with the more complex methods discussed later on.

5.1.1 Algorithm

Initially, a face detector based on the Viola-Jones object detection, explained in Section 3.2, is executed in order to detect the main face of the first frame of the sequence to track. Depending on how was the result of the face detector, this step can lead to different paths to follow.

- **One face found**. If the face detector found just one face, the area where this face is is going to be taken as the template to match in the following frame.

- **More than one face found**. There is also the possibility that the face detector found more than one face in the initial frame. Imagine the situation where there is one person in front of the computer that wishes to be tracked, but there are also other people behind him. In this case, the biggest of the detected faces is taken as the good one.

- **No face found**. Some times the face detector does not find any face, either because there is not any face in the initial frame, or there is actually a face but with some occlusion or bad illumination. In this case, this frame is ruled out and the following frame will be taken as an initial frame. As long as the face detector does not find any face, a loop will be ruling out frames until a face is found.

Once a face has been found in the initial frame, a region of interest, ROI, will be set up around this area. This ROI will be used to define the region where to run the template matching algorithm in the following frame, because the assumption that a face does not move a lot from any given frame to the next one is taken. If in the following iteration the template matching algorithm runs only inside the ROI, the performance will speed up a lot because there is much less space where to look for the template. The first frame in Figure 5.1 shows an example where the blue square is the detected face found by the face detector, and the green square is its region of interest.

The following step, as it has been already said, will be to look for the template inside the ROI of the following frame using the template matching algorithm. Once the best match is found, this new area is taken as a new template, and a new ROI is set around this area. In other words, in every frame the template is updated with the best match of the previous template inside the present ROI.

It is important to remark that the face detector only runs in the first frame and from then on all the faces are found using match template.

Figure 5.1 shows an example of how a face is detected from the first frame to the second one.

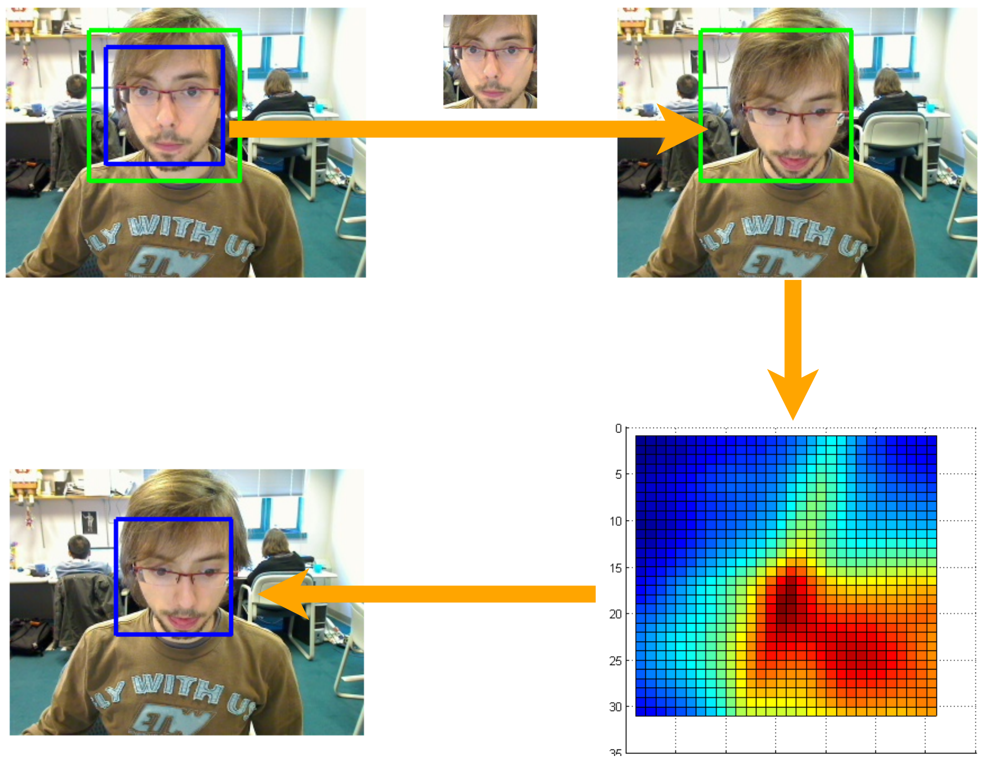


Figure 5.1: Example of how to detect a face in a template based tracker. From left to right and up to down, in the first frame the blue square is the face found by the face detector, and the green square is the region of interest. In the next frame the template matching algorithms look for the best match of the initial face inside the green area. Finally, the blue area shows where the best match was found, corresponding to the hottest point of the matching scores matrix.

5.1.2 Flowchart

Figure 5.2 show the flowchart of this algorithm.

5.1.3 Pros and cons

As it has already been said, this is a very simple tracker and this simplicity is its main strength. Moreover, it is important to say that it is scale, pose and rotation independent, and theoretically it will keep tracking the face even when this one is not in a frontal position.

But not everything is perfect; it has a lot of weaknesses and drawbacks. The first one happens when an occlusion occurs and the face is not totally visible. This situation will corrupt the template to track, because it will contain the occlusion itself,

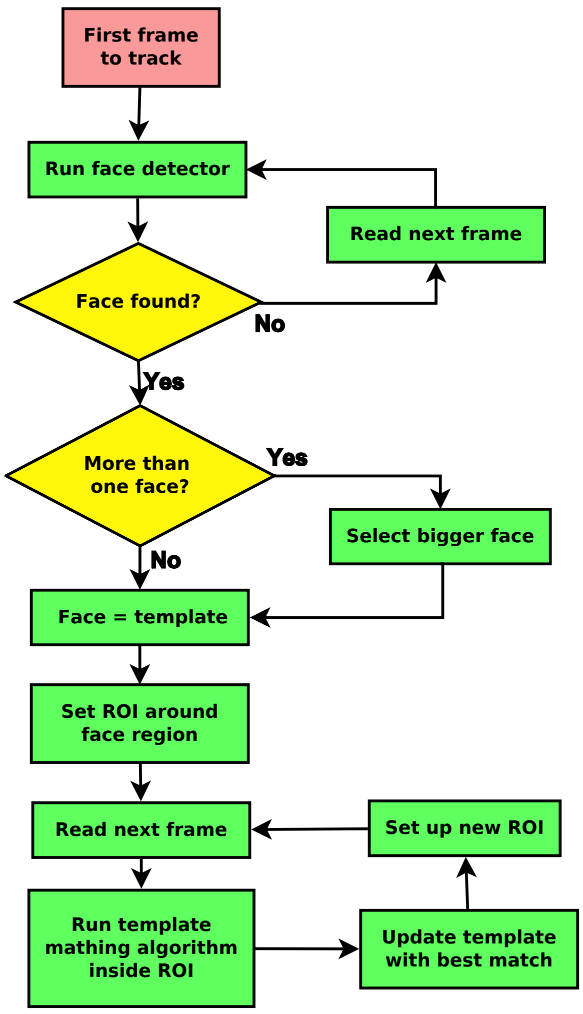


Figure 5.2: Flowchart of the tracker based on the template matching algorithm.

and it is possible that in the following iterations the template matching algorithm would track the occlusion instead of a the face. This could be fixed reinitializing the template to track periodically, using the face detector algorithm.

There is also a problem with the scale factor, because the tracker does not know how far or close the subject is with respect to the camera. The main consequence of this weakness is that the template size is always the same, even if the subject to track moved far away from the camera, which means that the face looks much more smaller.

Finally, this tracker does not give any information about the rotation, because it does not know anything about how the face moves rather than the translation motion.

As it has been showed, this tracker is a very simple one and it is only useful when a face has to be tracked in a very simple scenario, and no information other than the translation wants to be detected.

5.2 Affine face tracker based on the eye position

This tracker is firstly initialized by the face detector based on the Viola-Jones method discussed in Section 3.2, and then uses the template matching algorithm to find out the translation, rotation and scale factors of a frontal face very precisely. The only way to do so is tracking independently two or more points from the face, and according to how one of these moves with respect to the others it is possible to compute the affine motion of the face.

5.2.1 Algorithm

Idea

As it has just been said, two or more points are required to detect the rotation and scale motion of any object in a video, and these points can be, for example, the eyes. If the initial position of the eyes is known, it is possible to compute the their initial slope, and if the eyes can be tracked, it will be always possible to compute their slope at any moment. Thus, the rotation of the face can be detected in any given frame.

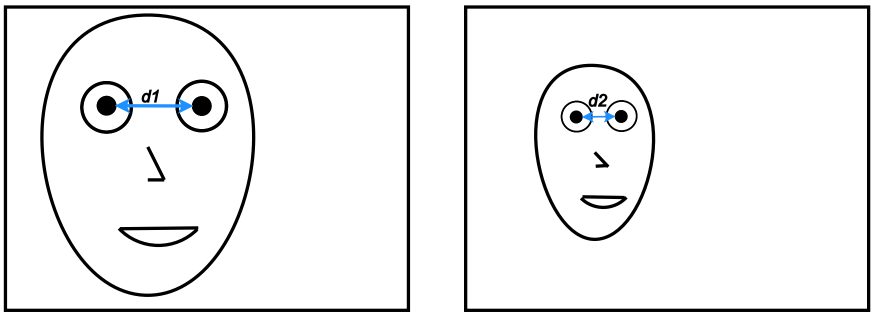
The same idea can be applied to the scale factor. If the absolute position of each of the eyes can be computed, then their distance is also known. Depending on if this distance gets higher or lower, it means that the face is getting closer of farther.

Figure 5.3 shows and example where a face is initially close to the camera and the distance between the eyes is d_{n-1}, and in the following frame the face moved away from the camera and the new eyes distance is d_n. Using affine transformations, discussed in Section 3.1, the scale factor needed to go from frame $n-1$ to frame n would be:

$$s = \frac{d_n}{d_{n-1}} \tag{5.1}$$

where, being $p_2(x, y)$ the center of the left eye and $p_1(x, y)$ the one of the right one, d_n is defined as follows

$$d_n = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2} \tag{5.2}$$

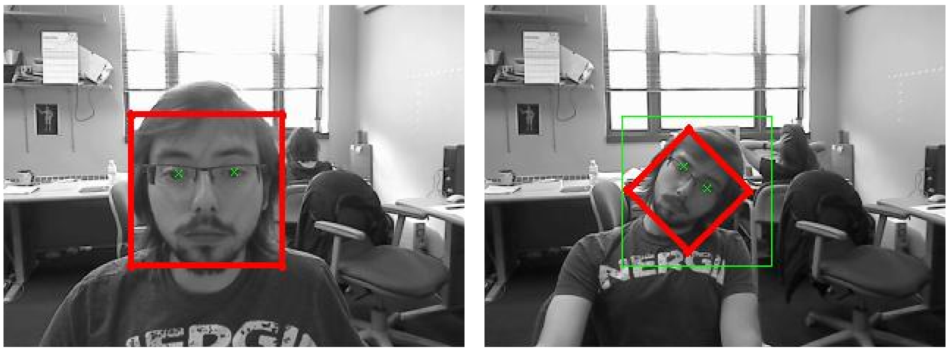


(a) Close (b) Far

Figure 5.3: Computing the scale factor is possible using the initial and the present eyes distance.

Once the scale, rotation and translation factors are known, it will be possible to apply the transformations that the face has been doing to the original square where the face was in the first frame. Hence, only detecting the first frontal face is enough to track the face, because this square will be modified using affine transformations along the time.

Figure 5.4 shows a real example of this tracking method. Figure 5.4(a) is the first frame of the sequence, where the face and eyes have been automatically detected using the Viola-Jones method. A few frames later, as shown in Figure 5.4(b), the original red square has turned to the right and became smaller, exactly like the face did.



(a) Initial frame, face and eyes automatically detected (b) Face automatically tracked

Figure 5.4: Real example of how the affine face tracker based on the eye position works.

Initialization

The initialization is one of the key steps of this tracker, because if the eyes and face are not properly detected the tracker will not work. Four different cascades classifiers based on the Lucas-Kanade method are used for the initialization: one for the left eye, one for the right eye, one for the eye area and one for the whole frontal face.

The reason for using four different cascade classifiers is because each of them have a certain false positive rate, but combining their results it is possible to discard most of them. For example, imagine that the *left eye detector* detects three left eyes in one single face, which means that two of them are false positives. To automatically discard them, some constraints are added: for example the left eye has to be inside the area detected by the *eye area classifier* classifier, and to the left of the *right eye classifier*.

An example of how a false positive looks like, and how to select which ones discard by using multiple cascade classifiers is showed in Figure 5.5.

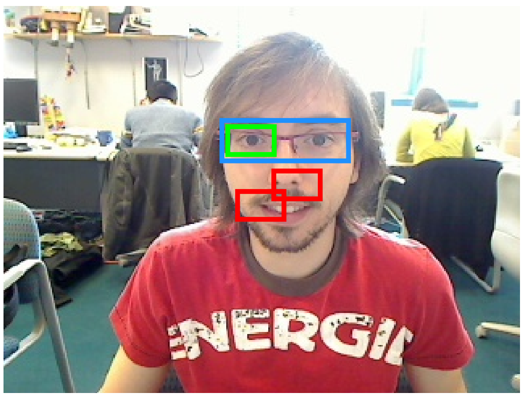


Figure 5.5: Red squares are false positives from the left eye detector, but they can be automatically discarded because they are out of the eye area (in blue) found by another cascade classifier trained to find the region around the eyes.

Hence, the false positives of the cascade classifiers are the main problem during the automatic initialization step. Sometimes it will take a few frames until the algorithm makes sure that the eyes have been correctly detected.

Main loop

The main loop of the algorithm starts when the face area and the center of both eyes have been automatically detected. The goal of this step is to compute the translation, rotation and scale that the detected face has from any given frame with respect to the previous one.

- **Translation**. The translation is found using the template matching method, discussed in Section 3.3, in both eyes. Once both eyes have been detected they are used as templates which, in the following frame, will find its best match inside the eye area. Defining the center of the left and the right eyes in frame n as follows

  $$\begin{aligned} &L_n(x_{ln}, y_{ln}) \\ &R_n(x_{rn}, y_{rn}) \end{aligned} \tag{5.3}$$

  the translation factor from any given frame $(n-1)$ to the following frame n is computed as follows

  $$\begin{aligned} t_x &= \frac{(x_{ln} - x_{l(n-1)}) + (x_{rn} - x_{r(n-1)})}{2} \\ t_y &= \frac{(y_{ln} - y_{l(n-1)}) + (y_{rn} - y_{r(n-1)})}{2} \end{aligned} \tag{5.4}$$

- **Rotation**. The rotation factor is computed using the slope of the eyes, comparing the previous one with respect to the present one. Defining the center of the eyes as Expression 5.3 shows, it is possible to compute the slope of the eyes at frame n as follows

  $$slope_n = \arctan\left(\frac{y_{rn} - y_{ln}}{x_{rn} - x_{ln}}\right) \tag{5.5}$$

  Hence, the rotation from any given frame $n-1$ to the following frame n is

  $$\theta = slope_{n-1} - slope_n \tag{5.6}$$

- **Scale**. As it has been already explained, the scale factor is computed from the distance between the eyes, comparing the one in the previous frame and the present one. Being Equation 5.2 the distance between the two eyes in frame n, the scale factor between frame $n-1$ and frame n is defined in Equation 5.1.

At this point of the main loop, the translation, rotation and scale factors that describe the motion of the face from the previous frame to the present one, are known. All this information can be compress in one single affine transformation matrix A_n, which contains the transformation needed for going from the frame n to frame $n+1$:

$$A_n = \begin{bmatrix} s \cdot \cos\theta & s \cdot \sin\theta & t_x \\ -s \cdot \sin\theta & s \cdot \cos\theta & t_y \\ 0 & 0 & 1 \end{bmatrix} \tag{5.7}$$

Having an affine matrix transformation for each frame, it is possible to compute the total affine transformation needed to go from the initial frame to the present one just by multiplying all of them.

$$A_T = A_n \cdot A_{n-1} \cdot \ldots \cdot A_1 \tag{5.8}$$

And finally, multiplying the initial square found by the face detector in the first frame by this A_T matrix, the new position of the face will be set. Figure 5.4 shows the result, where it is easy to see how accurate is the tracker.

5.2.2 Flowchart

Figure 5.6 shows the flowchart of this algorithm.

5.2.3 Pros and cons

This tracker is, in general, much more accurate than the matching template based one, discussed in Section 5.1, basically because this one gives information about the scale and rotation of the tracked face, and this is a very important feature for applications related with, for example, human computer interaction. Another advantage of this tracker is the eye detector and the eye tracker implemented in it, because this can lead to applications where there is interest in, for example, checking if the subject is falling sleep, or tracking his/her gaze.

But not everything is perfect, this tracker has also some important drawbacks. The most important one is its dependency on the good eye tracking by the template matching algorithm. If one of the eyes is not properly tracked, one of the points used to compute the slope and the rotation factors will not be correct, which will make the tracker to get lost.

Another important drawback is its dependency on tracking frontal faces, because the whole algorithm is based on affine transformations, which means that the tracked object has to have always the same *information*. If at some point the face turns to one side, there is not any affine transformation to guess how the face has moved because the face has changed its appearance. As the computation of the

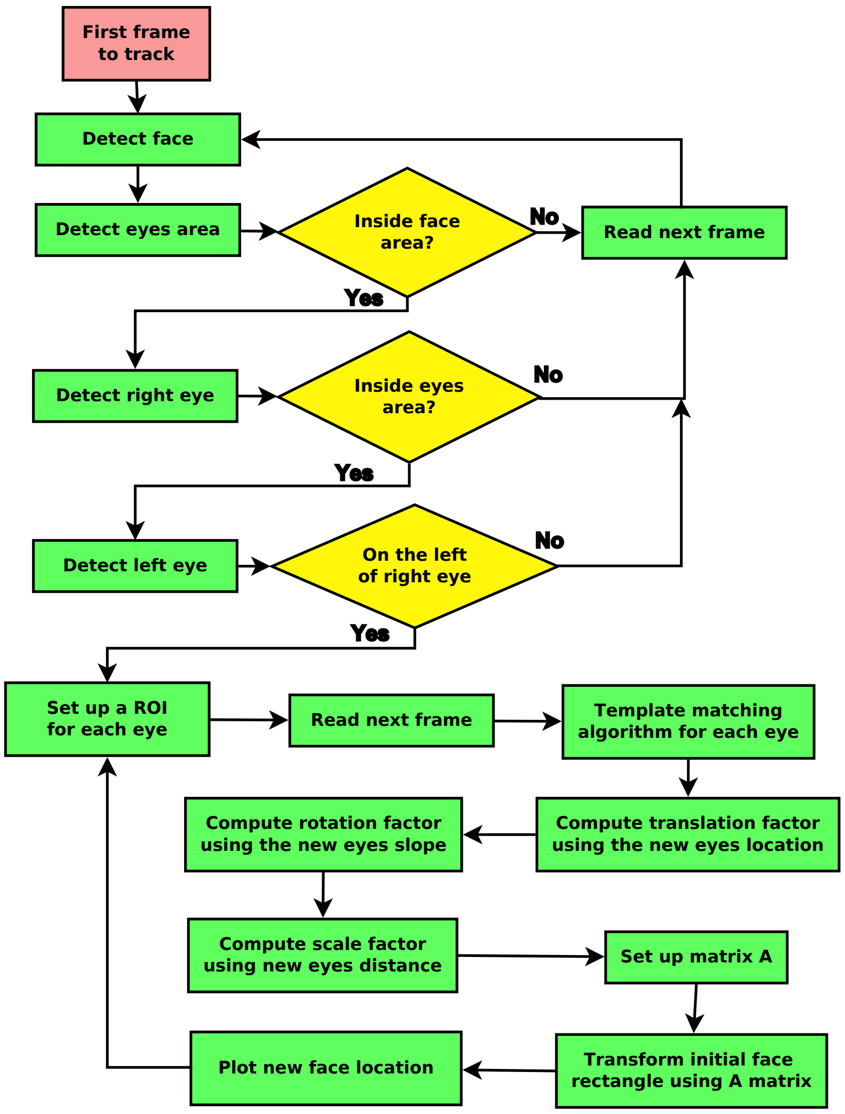

First frame to track
Detect face
Detect eyes area
Inside face area?
No
Read next frame
Yes
Detect right eye
Inside eyes area?
No
Yes
Detect left eye
On the left of right eye
No
Yes
Set up a ROI for each eye
Read next frame
Template matching algorithm for each eye
Compute rotation factor using the new eyes slope
Compute translation factor using the new eyes location
Compute scale factor using new eyes distance
Set up matrix A
Plot new face location
Transform initial face rectangle using A matrix

Figure 5.6: Flowchart of the affine tracker based on the position of the eyes

affine transformation is made from the movements of both eyes, the most important constraint in this tracker is that both eyes have to be visible in every single frame.

5.3 Affine face tracker based on the optical flow

All trackers showed so far were based on the template matching algorithm and, even though the previous tracker was able to detect the rotation and scale factors using the movements of the eyes, there are other more robust methods to detect the motion between two given frames.

Computing the optical flow, method that was showed in Section 3.5, is another way to find the motion between two frames and, although it requires a more complex computation than the template matching algorithm, it gives a more robust and reliable performance.

5.3.1 Algorithm

Idea

The main idea is to compute how the *most important* pixels from the present frame move to the following one. If this motion can be found, then the affine transformation that gives the relation between these two frames can be also found. As in the affine tracker based on the eyes position, once the affine matrix between any two given frames is known, it will be possible to bring the initial square where to face was, to the present location.

In other words, the key idea is to run the face detector just in the initial frame, where the face is in a straight frontal position and can be easily detected, and bring the square around this initial location to wherever the face is in the present frame, in which maybe the face detector could not find the face due to, for example, its rotation.

Before giving further details about how this tracker works, there are a few considerations that have to be taken into account. As there is only interest in tracking the face, this area is the only part of the whole frame that will be used to compute the optical flow. Otherwise, if something in the background is also moving, this motion would be also detected and used to compute the *average flow*.

Initialization

As said, the face region is the only part of the whole frame that will be used and the first step consists thus in its detection. As usual, a face detector based on the Viola-Jones algorithm will be employed, and a bounding box and a region of interest will be set around it.

Once the initial face has been detected, the only thing that lasts, before the tracking starts, is to detect the good features to track inside the face. This is going to be done either by the Harris edge detector commented in Section 3.4.1 or by the Shi and Tomassi method shown in Section 3.4.2. Both methods give a very similar performance, though the Shi and Tomassi one supposed to be a bit faster. An important detail when running such methods is to set up correctly the parameters they have, especially the one that refers to how many good features are going to be selected.

An important improvement with respect to the previous tracking method, based on the eyes position and discussed in Section 5.2, is that in this case when selecting the good features to track there are not false positives, because prior to be tracked by the optical flow algorithm it is no possible to know which ones have been well tracked and which ones have not.

Figure 5.7 shows a real example with the initial face detected and its good features to track selected by the Shi and Tomassi method.

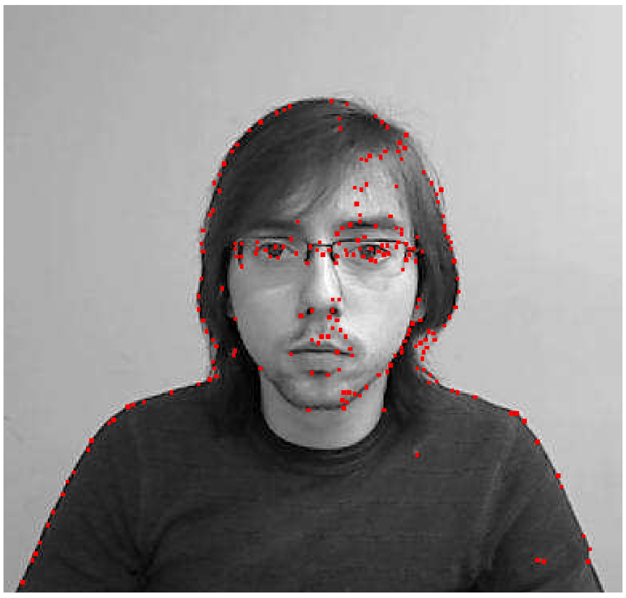


Figure 5.7: Initialization of the algorithm with the good features to track.

Main loop

The main goal in this step consists in tracking the good features from the previous frame to the present one. As it has been already said, this is going to be done using the optical flow algorithm proposed by Lucas-Kanade and discussed in Section 3.5.

Supposing that A_T is initially defined as the identity matrix, these are the steps the main loop follows:

- **Warp the present frame using** ${A_T}^{-1}$. Initially the current frame will be *warped* using the inverse of the matrix A_T, which contains the affine transformation needed for going from the initial frame to the previous one. Once this step is done, the face in the present frame would be almost in vertical position.

- **Locate the face in the warped present frame**. This is an optional step. Sometimes, to simplify the computation, the face area is the only region taken into account when computing the optical flow, hence it is necessary to *approximately* detect the face. This can be done either running the template matching method using the previous face as a template, or simply taking the coordinates where the face was in the previous frame and enlarge its region. Hopefully the face will be still in there.

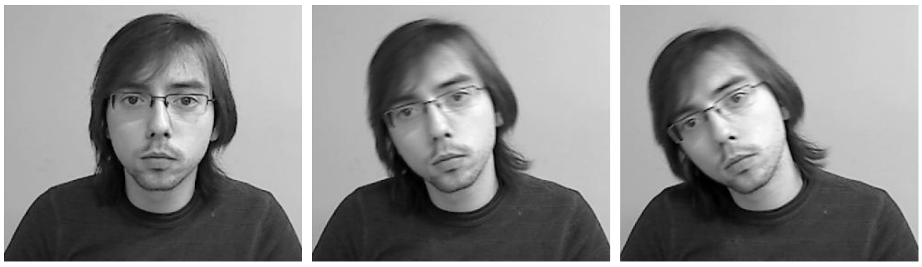

(a) Frame 1 (b) Frame 16 (c) Frame 20

Figure 5.8: Example of a sequence that will be tracked by this method

- **Recompute good features**. The good features to track inside the present frame face that has been warped are computed. They should look similar to the original good features to track, but due to illumination changes they may vary a bit.

- **Compute optical flow**. At this point of the algorithm, with the previous face warped, its selected good features to track and with a guess of where the face is in the present frame, the Lucas-Kanade optical flow approach is executed. This approach has as an input parameter the coordinates of all the good features to track from the previous frame and gives back a matrix with the coordinates of these features tracked in the present frame. Figure 5.9 shows the result, where for each of the features a line joins its previous location with its current one.

- **Mean motion**. The idea now is to compute the average motion of these tracked features, and take it as the motion that the face have experimented from the warped previous frame to the present one. In other words, the face was initially warped ${A_T}^{-1}$, which was the transformation found so far, but there was still a missing transformation between the previous frame and the current one; this is exactly what has been found. With this motion, a matrix

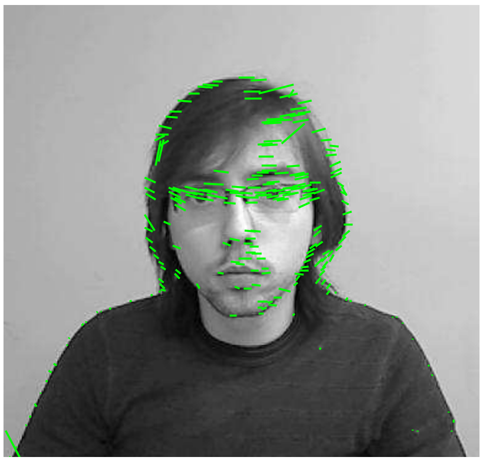


Figure 5.9: Optical flow of the good features to track between the first and the tenth frame of the sequence shown in Figure 5.8. In this sequence the subject is moving his face to the left, as the optical flow vectors shows.

A_n that gives the relation between the previous warped frame and the present one can be set. The way that this matrix is built was discussed in Section 3.1.

- **Recompute** A_T. Now that the motion between the previous frame and the current one as been computed and expressed in matrix A_n, it is possible to add this transform to the A_T affine transformation matrix using the expression that follows

$$A_{T new} = A_n \cdot A_{T old} \tag{5.9}$$

- **Transform initial square**. Finally, after all the previous steps, the square found by the face detector in the initial frame can be transformed and brought to the current face position.

5.3.2 Pros and cons

This is a much more complicated technique, compared with the other trackers perviously discussed. One of the more complex steps to do, which that requires a lot of computational effort, is warping the current frame back to almost the original position, using the ${A_T}^{-1}$ matrix. This is done using an interpolation which takes too long in high resolution images. Furthermore, as this tracker works with affine transformations, it is also only able to track frontal faces.

Regarding the good points of this tracker, the more important one is its ability to keep tracking a face even when small occlusions occur. Such situation can be handled because it is possible to compute the standard deviation of the length of all the vectors that draw the optical flow. If one of these vectors is *too* long with respect to all the others, it is possible to affirm that its corresponding feature has not been properly tracked, which can be caused for example by an occlusion. Thus, if most of the good features can be correctly tracked, but some of them can not due to an occlusion, it will be still possible to track the face.

5.4 Face tracker based on the mean face

The performance of this tracker exceeds the one of the template matching based tracker reviewed in Section 5.1, where the template used was just the last face found. That method had an important drawback: when an occlusion occurs the template to match gets corrupted with such occlusion, thus, in the following frame, if the occlusion is gone, the algorithm will not be able to match the object.

A simple way to sort out this handicap is to compute the mean of the object to track, for example, in the last 100 frames. In other words, to learn how the object looks like most of the time. Then, when a occlusion occurs, perhaps the matching template algorithm will not find any good match using the mean face, but when such occlusion disappears the tracker will automatically recover because the mean of the last 100 frames will still look good.

5.4.1 Algorithm

The initialization process is very similar to the one of the tracker showed in Section 5.1, running the template matching algorithm until finding a face, and setting up a region of interest around it.

A buffer of the last 100 faces found is created to compute a mean face in every iteration. Of course, during the first 100 frames the mean face is computed only by the total faces detected so far.

To check how good is the best match between the mean face and the present frame, the difference between them is computed and compared to a certain threshold.

- **If the difference is higher than the threshold**. It means either that an occlusion is happening or that the face was not properly tracked and that best match does not thus correspond to any face. In this particular case, the region of interest around that *best match* is set up bigger than the normal one.

- **If the difference is lower than the threshold**. It means that the face has been properly tracked and the best match corresponds to a face. The region of interest is set up around the new face location.

In both cases the best match is taken as a face and added to the mean face. Thanks to the fact that the mean face is computed using the last 100 best matches, even when some of them are not correct, it will still look as a face. Figure 5.10 shows how the mean face evolves along a sequence: even though it gets blurry along the time, it always looks like a face.

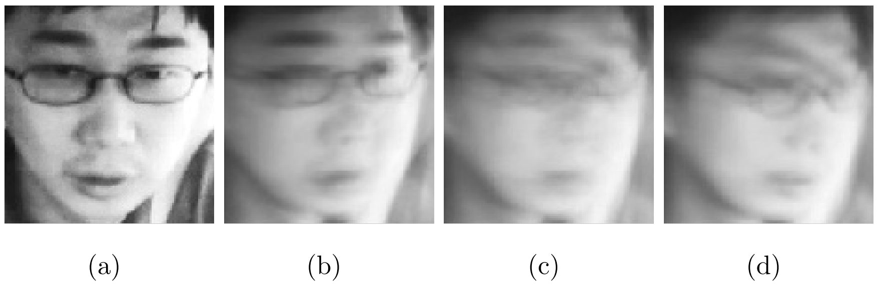


Figure 5.10: Evolution of the mean face along the first 100 frames of a sequence.

The key idea of this method is taking the mean face as a template. When the subject does dramatic movements the tracker will get perhaps get lost but:

- The tracker will know that the face is lost.
- The tracker will automatically recover the face as soon as it becomes visible again.

5.4.2 Flowchart

Figure 5.11 shows the flowchart of this algorithm.

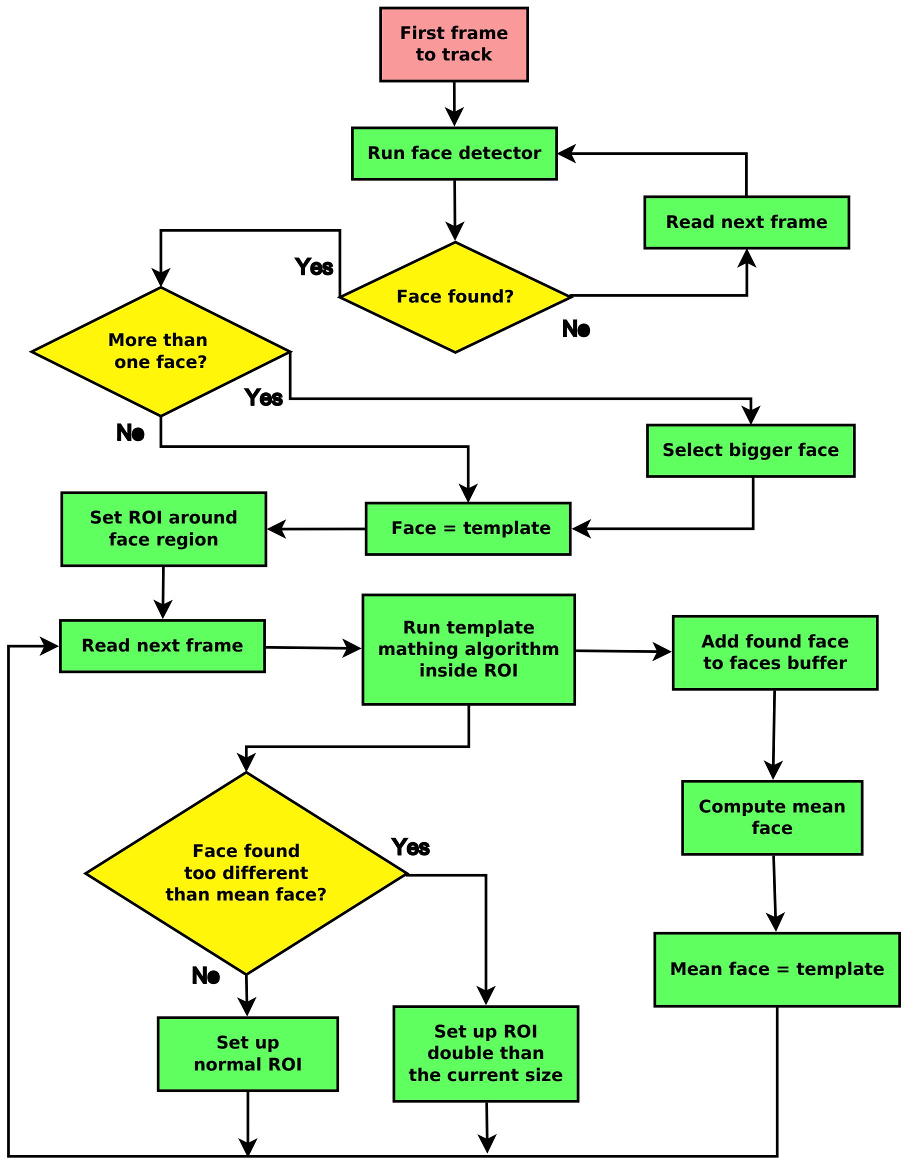


Figure 5.11: Flowchart of the template matching face tracker based on the mean face.

5.4.3 Pros and cons

The main advantage of this tracker is its ability to automatically recover when it loses the face due to dramatic movements or occlusions. This makes the tracker a very reliable and robust system because, at some point, it will automatically find again the face.

About the drawbacks, the more important one is the lack of information about the scale and rotation factors.

5.5 Non-affine eye tracker

Since most of the trackers discussed so far are based in the use of affine transformation, they are not able to deal with pose changes.

This tracker shows a very simple approach of how to deal with objects that disappear from the image in a certain frame, as the eyes do when the subject turns his face to one side.

5.5.1 Algorithm

This algorithm is based on the template matching method showed in Section 3.3, with which the eyes are going to be individually tracked frame by frame. Initially, the eyes can be either manually selected by clicking its center with the mouse pointer, or automatically detected using cascade classifiers trained to detect eyes, method that has been already discussed in the initialization part of the affine tracker based on the eyes position explained in Section 5.2.1.

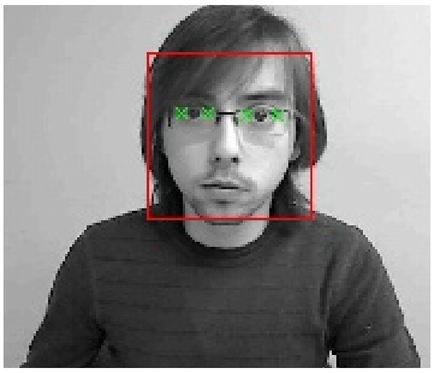
Once the eyes have been initialized, the algorithm creates a patch for each of them. This patches will be used to track the eyes along the sequence, basically using the template matching method showed in Section 3.3.

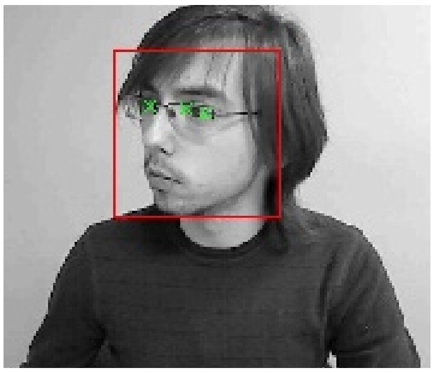
The main innovation of this tracking method relies on how the algorithm detects that a tracked area, any of the eyes in this particular case, disappears from the view of the camera. In every single frame, when the template matching algorithm has found the best match for each of the patches, the new coordinates have to fulfill a list of requirements. If they does not pass them, it means that the new position that has been found is not correct and the corresponding eye is considered lost.

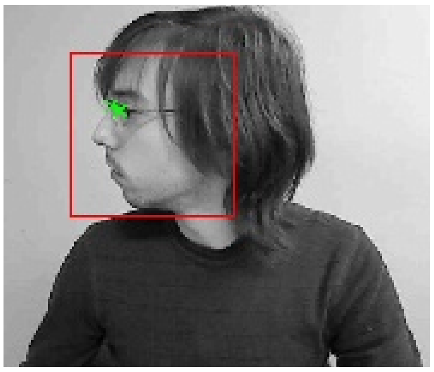
This requirements list consists of a set of situations that can never happen in a human face. For instance, if the distance between the corners of the eye gets smaller and smaller, until it is less than 0.5 cm, it means that the subject is turning his face to one side and therefore, one eye is lost. Other requirements are related with the slope between the eyes, as it has to be the same for any given pair of eyes' corners.

When any of the eyes gets lost, the tracker keeps looking for the last correct template found in both the last position where was detected and in its original position. If the eye is lost because the subject turned his face to one side, it is assumed that he will turn back to a frontal position again. When this happens, the tracker will detect again a good match between the last good eye patch found and the current eye appearance and the eye will be tracked again.

Figure 5.12 shows an example of a sequence where a subject turns his face to one side, and then turns back frontal again. In the first frame, showed in Figure 5.12(a), both corners of both eyes are correctly detected. A few frames later, in Figure 5.12(b), the face appears turned to the left and the outer corner of left eye is already hidden. The tracker correctly detects this and stops tracking it. The face keeps turning until it reaches a total profile position, showed in Figure 5.12(c), where almost non of the eyes are visible and they are all represented in one single point. Finally, in Figure 5.12(d), the subject turns frontal again and the tracker recovers the lost points.


(a) Initial position


(b) Turning to one side


(c) Profile shape

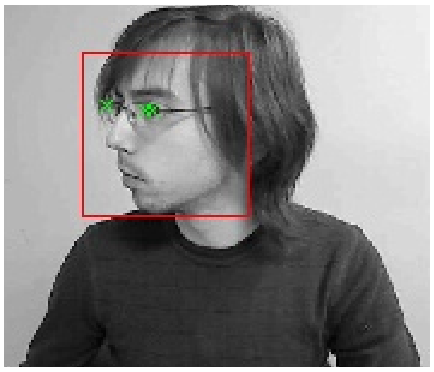

(d) Turning back to frontal position

Figure 5.12: Real example of the non-affine eye tracker. In this sequence the subject turns his face to one side and then turns back frontal again.

5.5.2 Flowchart

Figure 5.13 show the flowchart of this algorithm.

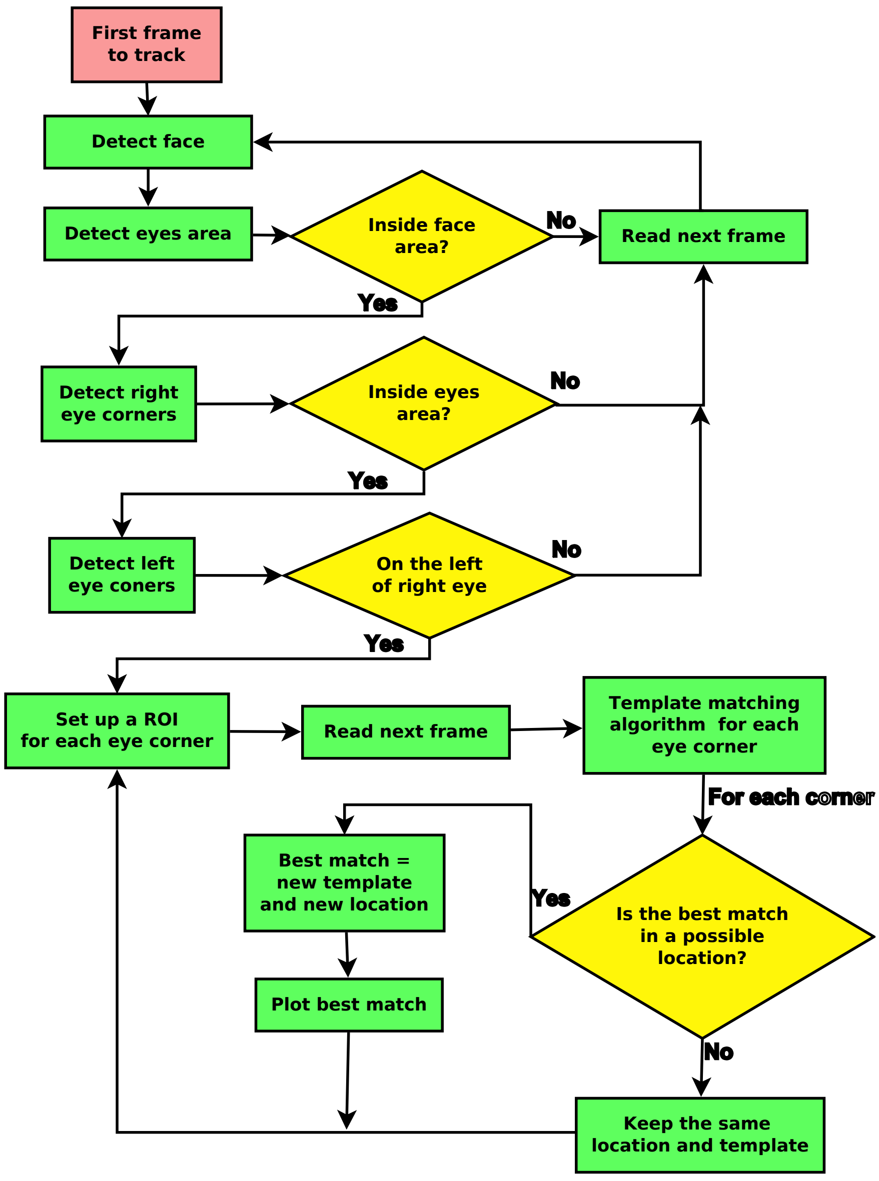


Figure 5.13: Flowchart of the Non-affine eye tracker algorithm.

5.5.3 Pros and cons

This tracker is just a simple approach to show an easy way to keep tracking parts of the face that at some point get hidden along the sequence, but then show up again. As it does not work using affine transformations, it does not compute neither the scale or the rotation factors, but they can be easily implemented in future improvements.

Chapter 6

Results

This chapter summarizes the results obtained once all the tracker techniques explained in Chapter 5 have been implemented. Although in order to check the real performance of each tracker it is necessary to use video format, key frames where it is possible to see the performance in specific moments along the tracking sequence will be shown.

Each of the following sections shows the performance of an specific feature, such as frame per second rate, scale precision, pose robustness, etc., for each of the implemented methods.

6.1 Translation precision

The translation precision is very similar in every implemented method, but it is a bit better in the ones based on the template matching algorithm than in the ones based on the optical flow.

The reason of this difference is the small error that occurs when the optical flow from the good features to track is computed. Since some of these features cannot be correctly tracked, the computed average movement might be a little corrupted by such bad tracked points.

Table 6.1 shows, among other results, the translation precision performance for each of the trackers.

6.2 Scale precision

The scale factor gives information about how close is the subject with respect to the camera. When the subject that is being tracked is closer to the camera, his face looks bigger in the video and the opposite occurs when he moves away from the camera. These changes must be reflected in the square around the face: it must get bigger or smaller depending on the movements of the subject. Otherwise, it will either track the face just partially or track other non-relevant objects present in the background of the image.

Figure 6.1 shows an example of the tracker based on the mean-face template matching, discussed in Section 5.4, where the subject comes closer to the camera. Since such tracker does not take into account the scale factor, in Figure 6.1(b) the face does not fit into the square.

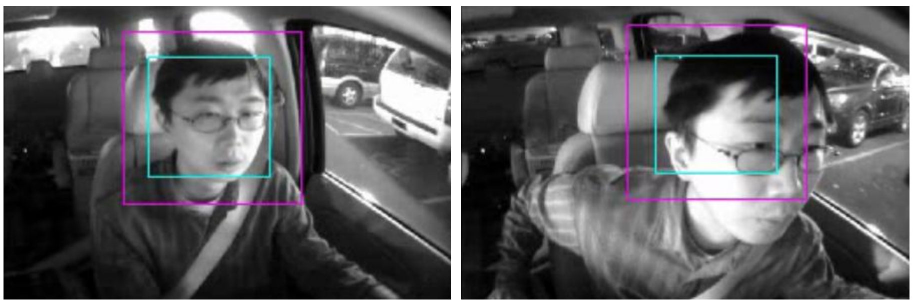

(a) (b)

Figure 6.1: Example showing the importance of the scale factor. In this sequence the tracked face comes closer to the camera, but the square that shows where the face is keeps the same size. A tracker method that contemplates the scale factor, would increase the size of the square.

Such problem is fixed in the affine tracker based on the eyes position where, when the subject comes closer to the cam, the square gets bigger. Figure 6.2 shows the performance of this tracker.

6.3 Rotation precision

The rotation factor is only computed in the tracker based on the optical flow, showed in Section 5.3, and in the affine tracked based on the position of the eyes, showed in Section 5.2. The other methods cannot compute this feature.

Because of the small error caused by the optical flow computation, explained in Section 6.1, the rotation precision computed by the eye tracker is a bit better.

Figure 6.2 shows a few frames where the eye tracking based tracker is running, and it is possible to see how precise the rotation factor is.

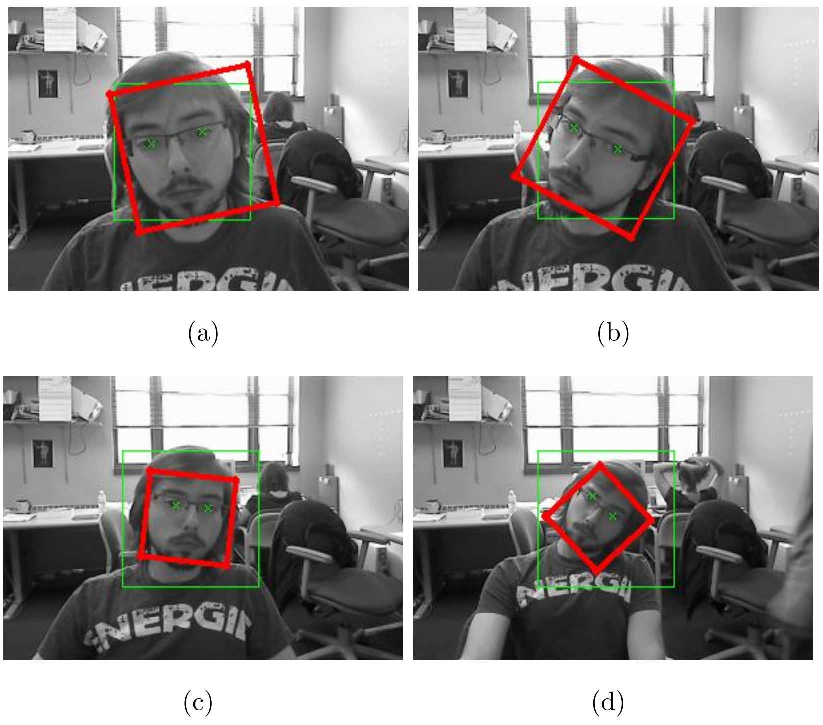


Figure 6.2: Performance of the affine tracker based on the eyes tracking. Translation, rotation and scale factors work perfectly.

On the other hand, Figure 6.3 shows the performance of the rotation factor in the tracker based on the optical flow. As it is easy to notice, for example in Figure 6.6(b), the green square should be more rotated to the right.

Finally, Figure 6.4 shows an example where the simplest of all the trackers, based on template matching and described in Section 5.1, is used to track a sequence where the subject turns his face to one side. Since in this technique the scale and rotation factors are not computed, the green square is always in an vertical position and has always the same size.

6.4 Illumination robustness

The illumination factor has to be taken into account when tracking an object in a video, because depending on the amount of light that a given frame receives, the object will look darker or lighter.

An easy way to deal with this problem is applying the histogram equalization method before processing each of the frames to track. This technique distributes the intensities of any images in a better way, substantially increasing the contrast

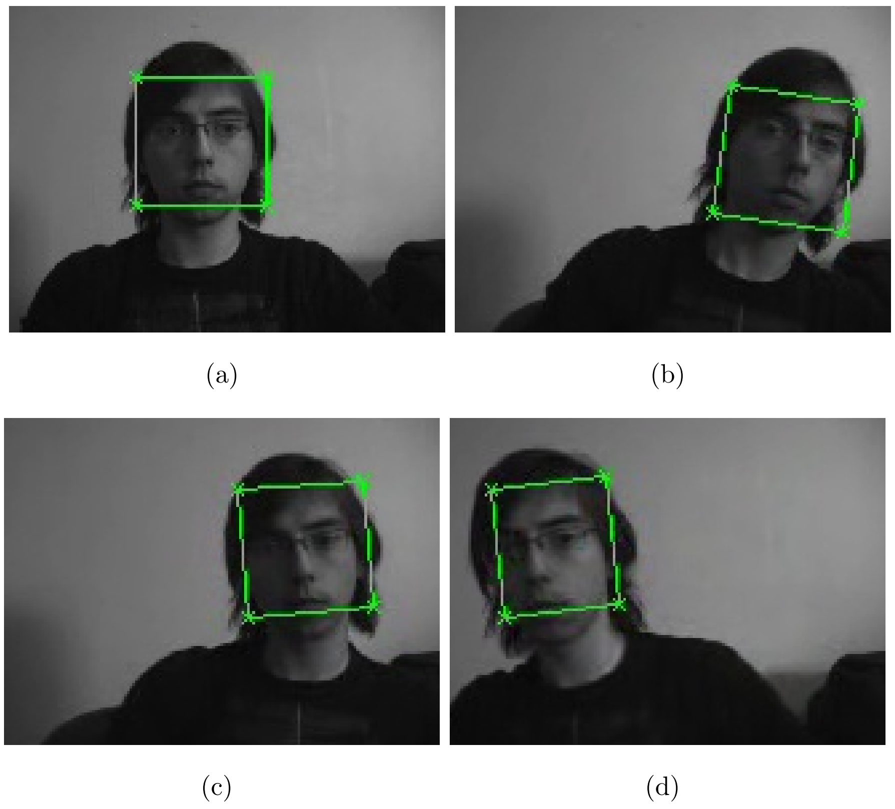


Figure 6.3: Performance of the optical flow based tracker. The rotation factor is almost perfect, but not as good as the results showed in Figure 6.2.

of the images.

Figure 6.5(a) shows an example where the illumination is not good enough to clearly distinguish a face. Before processing it, the region of interest where the face is supposed to be, highlighted with a pink square, was enhanced by the histogram equalization method. The enhanced found face is showed in Figure 6.5(b).

Since this can be applied to all the implemented methods, they are all illumination invariant.

6.5 Pose robustness

The methods based on affine transformations are only able to track faces that do frontal movements, and this is a huge constraint. On the other hand, the template matching based trackers are pose invariant, but they can compute neither the scale nor the rotation factor.

Figure 6.6 shows an example of the template matching based tracker where the

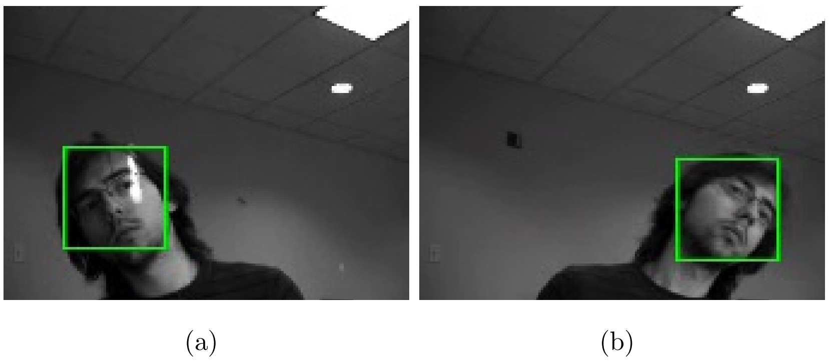


Figure 6.4: The rotation factor cannot be detected using the template matching algorithm. Notice how the green square is always in a vertical position.

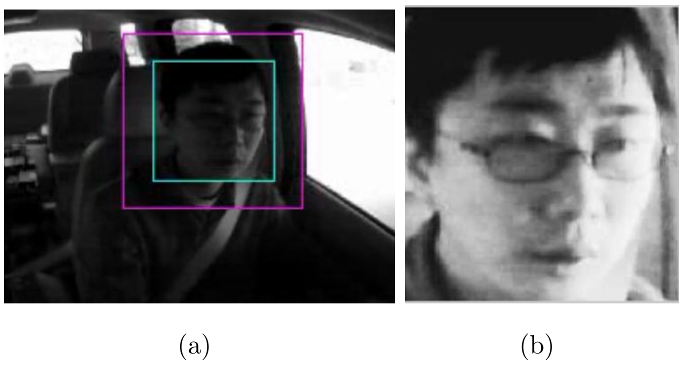


Figure 6.5: Example of the enhancement achieved using the histogram equalization technique before tracking the face.

face can be tracked even if it is in a profile position. Although this result looks good, this is not a very robust tracking method because it does not give any information but the translation, and the template to track gets corrupted very easily.

6.6 Occlusion robustness

The affine tracker based on the optical flow, and the template matching based on the mean face are the two methods that offer a higher partial occlusion robustness performance.

In the case of the first one, based on the optical flow, this robustness is achieved thanks to the optical flow from the good features to track, with which is possible to distinguish the points that have been well tracked by computing the standard deviation of all of the tracked points. If an occlusion happens, the good features

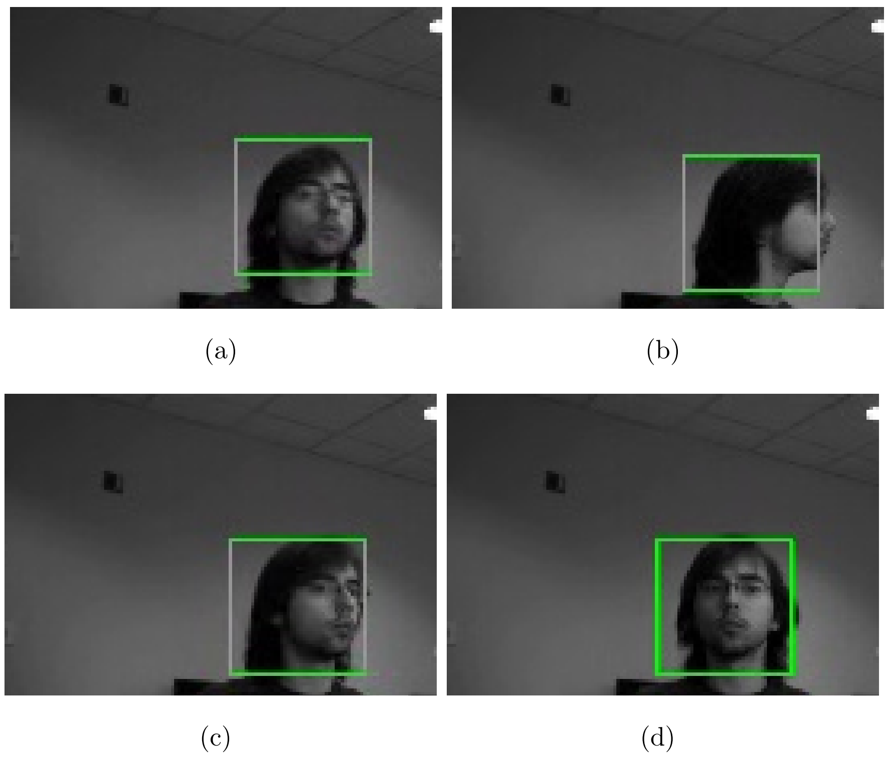

(a) (b) (c) (d)

Figure 6.6: Performance of the template matching based tracker, showing its pose robustness.

to track of the occluded area will have an optical flow much different that the well tracked ones and they can be discarded.

The second method that is invariant to partial occlusions is the one based on the mean face template matching algorithm. With this algorithm, the template to match is computed using the last 100 found faces. Therefore, after a period of occlusion of the face, it can be tracked again because the mean face would look still like the face being tracked. However, during the occlusion time the tracker will probably get lost.

6.7 Execution speed

One of the goals of this project was to develop real-time algorithms, being the execution speed an important feature to analyze. As a conclusion, and taking into account that the frame size of all the tested sequences is 640x480 pixels, it is possible to confirm that all the implemented methods are able to run in a frame per second rate high enough to work in real-time systems, except the affine from optical flow showed in Section 5.3.

The reason of the slow execution in the case of the affine from optical flow tracker is the need to warp each of the frames using the accumulated affine transformation matrix when trying to bring each of the faces to the original location. As it was already commented in Section 5.3, this process is very time-consuming. A few solutions have been tested in order to make this step faster. One of the most satisfactory consists in warping just the face area instead of the whole frame, because the size of the image to warp is much smaller. However, the speed up that this solution brings is not enough to achieve a real-time execution of this algorithm.

The frame per second rates achieved for each of the algorithms are summarized in Table 6.2.

6.8 Summary tables

	Translation precision	**Scale precision**	**Rotation precision**
Template matching	Excellent	None	None
Eyes position based	Excellent	Excellent	Excellent
Affine from optical flow	Very good	Very good	Very good
Based on mean face	Excellent	None	None
Non-affine eye tracker	Excellent	None	None

Table 6.1: Performance of each of the methods, regarding the translation, scale and rotation factors.

	Illumination robustness	**Pose robustness**	**Occlusion robustness**	**Speed execution**
Template matching	Excellent	Good	Poor	~15 fps
Eyes position based	Excellent	Poor	None	~12 fps
Affine from optical flow	Excellent	Poor	Very good	~3 fps
Based on mean face	Excellent	Very good	Very good	~12 fps
Non-affine eye tracker	Excellent	Good	None	~13 fps

Table 6.2: Performance of each of the methods, regarding the robustness to illumination, pose and occlusion.

Chapter 7

Conclusions and future work

Taking into account the initial goals of the project and once the performance of the implemented methods has been discussed in Chapter 6, this chapter aims to extract some general conclusions.

Moreover, it summarizes the future work that can be done in order to improve the performance of the algorithms, with the final goal of having a totally robust face tracker, capable of tracking a face in any given scenario.

7.1 Conclusions

Face tracking is a hard problem to solve due to the large amount of variables that appear when trying to *teach* a computer how a face looks like and how it moves. Recent advances in machine learning and computer vision techniques have made this work increasingly possible, but there is still a long way to go to create a totally robust face tracker.

Face tracking would not be possible without a proper face detection algorithm which, thanks to Viola and Jones [15], is now an easy task that can be done in real-time when working with straight frontal faces, but does not work in any other pose or rotation. Thus, a face cannot be tracked just detecting it in a frame-by-frame basis, because it might be not always detectable by the Viola-Jones algorithm.

These restrictions lead to the development of face tracking techniques that, for any given frame and having the face detected in the previous one, should be able to find its location in the present one. In other words, the goal is now to find the motion between any two given frames, task that can be done by template matching or optical flow algorithms, among other techniques.

Each one of these techniques have a different performance and, depending on what is being tracked and how it moves, one of them will be more appropriate to use than the others. Without entering into details about their performance, it is important to say that none of both is good enough to build a totally robust system, basically due to the large amount of variables to deal with.

Adding more information to the algorithms, for example, how the mean-face looks like or where the face is supposed to be located in the present frame, is a way to help them to achieve its goal. But again, even though it is possible to build robust methods for specific contexts, there is not a universal robust method yet.

In this project a few methods have been developed and tested in different scenarios. The goal was to develop a face tracker, and it has been achieved with the constrain that, for each scenario, a particular method has to be used.

7.2 Future work

There is still a long way to go until a robust method to track faces that works in any given context can be built. To complete the work done in this project, some other recent published methods can be studied and implemented, such as Active Appearance Models [4], Adaptive PCA Face Tracking or Mean-Shift[3].

Active Appearance Model is a method published by Cootes, Edwards and Taylor in 2001 [4] that matches a statistical model of object shape and appearance, built in a training phase, to a new image. Applying this method and having a proper previous training step, it could be possible to track faces not only in frontal position but also in any other pose or expression. The only drawback of this method is its computational cost, which is usually too high to run in a real-time system.

Adaptive PCA Face Tracking is another technique that would bring more robustness to the tracker. It consists in applying the Principal Component Analysis, PCA, algorithm to all the faces founds so far, in order to find a simpler way to represent them. As the video sequence goes on, a new PCA basis is computed in every iteration, process that will lead to a very robust representation of the face, since it takes into account the different poses and expressions observed so far.

Applying these methods and other ones that are currently being researched would help to create a more robust face tracker. Moreover, it would be also interesting to investigate the development of an algorithm that uses the best part of each of the methods discussed along this project, resulting in a both pose and expression invariant tracker.

Bibliography

[1] M. J. Black and P. Anandan. A framework for the robust estimation of optical flow. pages 231–236, 1993.

[2] Gary Bradski and Adrian Kaehler. *Learning OpenCV: Computer Vision with the OpenCV Library.* O'Reilly Media, Inc., 1st edition, October 2008.

[3] D. Comaniciu and P. Meer. Mean shift: A robust approach toward feature space analysis. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24(5):603–619, 2002.

[4] T. F. Cootes, G. J. Edwards, and C. J. Taylor. Active appearance models. *Lecture Notes in Computer Science*, 1407:484–??, 1998.

[5] Bernhard Fröba and Christian Küblbeck. Robust face detection at video frame rate based on edge orientation features. In *FGR '02: Proceedings of the Fifth IEEE International Conference on Automatic Face and Gesture Recognition*, page 342, Washington, DC, USA, 2002. IEEE Computer Society.

[6] G.D. Hager and P. Belhumeur. Efficient region trackingwith parametric models of geometry and illumination. *IEEE Trans. on Pattern Analysis and Machine Intelligence 20*, 20:1025–1039, 1998.

[7] Chris Harris and Mike Stephens. A combined corner and edge detector. In *The Fourth Alvey Vision Conference*, pages 147–151, 1988.

[8] Desmond J. Higham and Nicholas J. Higham. *MATLAB Guide.* Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2000.

[9] Berthold K.P. Horn and Brian G. Schunck. Determining optical flow. Technical report, Cambridge, MA, USA, 1980.

[10] Kazuhiro Hotta. A robust object tracking method under pose variation and partial occlusion. *IEICE - Trans. Inf. Syst.*, E89-D(7):2132–2141, 2006.

[11] Bruce D. Lucas and Takeo Kanade. An iterative image registration technique with an application to stereo vision (ijcai). In *Proceedings of the 7th International Joint Conference on Artificial Intelligence (IJCAI '81)*, pages 674–679, April 1981.

[12] C. P. Papageorgiou, M. Oren, and T. Poggio. 1998, a general framework for object detection. *Computer Vision, 1998. Sixth International Conference on*, pages 555–562, 1998.

[13] Jianbo Shi and Carlo Tomasi. Good features to track. In *1994 IEEE Conference on Computer Vision and Pattern Recognition (CVPR'94)*, pages 593 – 600, 1994.

[14] J. Sung, T. Kanade, and D. Kim. Pose robust face tracking by combining active appearance models and cylinder head models. *International Journal of Computer Vision*, 80:260–274, 2008.

[15] Paul Viola and Michael Jones. Robust real-time object detection. In *International Journal of Computer Vision*, 2001.

List of Figures

List of Tables

Resum

Les tècniques de seguiment facial s'han convertit durant els darrers anys en un dels temes de recerca més populars en el camp de la visió per computador, principalment gràcies a la gran quantitat de situacions on poden ser aplicades en el món real. Tot i que la definició del problema és molt fàcil d'entendre, trobar una solució robusta resulta ser molt difícil, en part per culpa de, per exemple, els canvis d'il·luminació, posició, aparença, etc., que se succeeixen al llarg d'una sequència de vídeo. Inicialment, aquest projecte fa una ràpida introducció a l'estat del art tant de les tècniques de seguiment facial com de les de detecció de cares. A continuació, es mostren los publicacions més rellevants dels últims anys en aquesta àrea, com per exemple el mètode de reconeixement d'objectes de Viola i Jones, o l'algoritme de Lucas i Kanade per calcular el flux òptic. Finalment, s'implementen cinc diferents algoritmes de seguiment facial, cada un dels quals ofereix una solució robusta per a un determinat context.

Abstract

Face tracking has become an increasingly important research topic in the computer vision field, mainly due to its large amount of real-world situations where such methods can be applied. Although the definition of the problem to be solved is very easy to understand, it is very difficult to come up with a robust solution due to variations in illumination, pose, appearance, etc. Initially, this project gives a brief introduction to the current state-of-the-art of both face detection and face tracking techniques. Moreover, the most important publications in this area, such as the Viola-Jones method for object recognition or the Lucas-Kanade algorithm for the optical flow, are presented. Finally, five different face tracking algorithms are implemented, each of them giving a robust solution for a specific context.

Resumen

Las técnicas de seguimiento facial se han convertido en los últimos años en uno de los temas de investigación más populares en el campo de la visión por computador, principalmente gracias a la gran cantidad de situaciones en el mundo real donde pueden ser aplicadas. Aunque la definición del problema es muy fácil de entender, encontrar una solución robusta resulta ser muy difícil, en parte por culpa de los cambios de ilumunación, pose, apariencia, etc. que se suceden durante una secuencia de vídeo. Inicialmente, este proyecto hace una rápida introducción al estado del arte tanto de las técnicas de seguimiento facial como de las de detección de caras. A continuación, se muestran las publicaciones más relevantes de los últimos años en este campo, como por ejemplo el método para reconocimiento de objectos de Viola y Jones, o el algoritmo de Lucas y Kanade para calcular el flujo óptico. Finalmente, se implementan cinco diferentes algoritmos de seguimiento facial, cada uno de los cuales aporta una solución robusta para un determinado contexto.