Facial expression recognition

Delft University of Technology (TU Delft)
-
Electrical Engineering, Mathematics and Computer Science
Knowledge Based Systems (EEMCS – KBS)

Project Report, February 2nd to July 16th 2004

Frédéric GEORGES

Supervised by Professor Leon Rothkrantz

0. TABLE OF CONTENTS

1. Acknowledgements & Summary

Above all, I would like to thank all researchers and students of KBS department (Knowledge Based Systems) of TU Delft for their reception and their sympathy.

I specially would like to thank our Master: Leon Rothkrantz for its reception, its helpfully and its kindness, and Dactu Dragos for his availability and his helpfully.

I also thank all my Erasmus friends I met in Netherlands (In particular the ones who lived in the same residence than me : Helena Sà Marqués, Silvia Lupini, Isabel Beck, Irina Plesnila, Daniel Teixeira, Stephano Ganassin, Gianluca Tanda, Daniel Weiss, Bogdan Dolinski, Tebelet, Radek Sovjak, etc. and the others I cannot name because they are too much) for their affection and kindness, and especially Romain Fissette from the same school than me, for his helpfully and his friendship during this project and our stay in the Netherlands.

Thank you finally to ENSEIRB administration, which helped us for various steps: Mr. Paul Y. Gloess and Ms. Evelyne Blanchard who made possible this stay in Netherlands.

Abstract :

Automatic recognition of facial expression is rapidly becoming an area of intense interest in the research field of machine vision. In this report, I present an algorithm I developed to recognize a facial expression in profile-view colored face images. From these images I extract the profile contour and from it, I extract 10 profile-contour feature points. Based on those points, the facial expression can be found.

Key words:

Facial expression recognition, image processing, features points, contour of profile-view face or silhouette.

2. Introduction

I carried out our final project at TU Delft, in the Netherlands, within the framework of the exchange program Socrates-Erasmus. I worked from February 1 to July 16, 2004 within the research laboratory of Knowledge Based System in EEMCS faculty of TU Delft.

My mission was to find and implement an algorithm to recognize a facial expression in a face profile image.

Leon Rothkrantz, our professor at TU Delft and Datcu Dragos a student at the KBS, have supervised me. This training course had for main objectives:

- To train theoretical and practical knowledge acquired during 3 years at ENSEIRB … in particular skills related to technologies of multimedia

- To work in an international context: it means to improve my English by using it both for work and everyday life. But also to discover a country and its culture: the Netherlands … but still to meet students coming from the whole world and particularly from the other countries partners of the European Erasmus program (Germany, Spain, Portugal, Italy, Romania, etc.)

These goals were largely reached during these 6 months. This experiment was very instructive and enriching for my future work. This project still accentuated my desire to work in a multicultural and international context … while allowing me to conclude a stimulating project at the technical level.

3. Purpose

3.1 ISFER: Global presentation

3.2 Original Subject from Mr Leon ROTHKRANTZ

In my part, the goal was to find and implement an algorithm witch would recognize a facial expression in face profile image. This work should complete the research of facial expression on an image, knowing that a module already exists for the frontal view face. Combining both researches, the algorithm should recognize a facial expression with a low level of error.
This algorithm can be added to the ISFER program with or without manual verification, depending on its quality.

There are 2 different steps:

- Find some characteristics points in the face profile image.
This approach is very simple. We start with an image of a face profile. Then we find the contour of this silhouette. Finally we deduce the profile-contour fiducial points that correspond to the extremities points of the contour as we can see in Fig. 1.

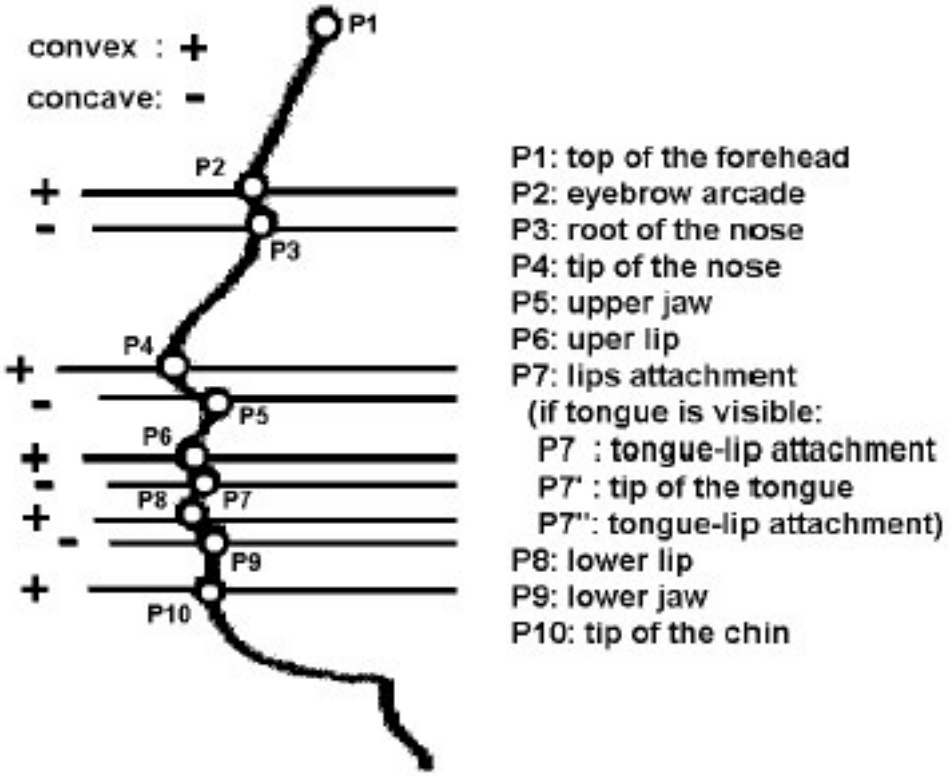


Figure 1 : Features points

- Analyze these points and deduce the facial expression corresponding among neutral, fear, anger, disgust, happiness, surprise and sadness.
Here we have to find some characteristics values deduced by the precedent research like distance between points or angles formed. Then, the facial expression correspondent is given by an analyze of this values and a comparison with the same values in known expressions.
We also have to compute the different values for known images.

3.3 Understanding the problem

That was a subject proposed by Mr Rothkrantz before my arrival in Delft. I chose it first because of my specialization in the multimedia technologies in my third year, but also because it sounded very interesting and was like a challenge for me.

3.4 Specifications of the project

3.4.1 Hardware needed

- Computer with the appropriate softwares

3.4.2 Software specifications

With our supervisor, we outlined the requirements of the algorithm I wrote.

3.4.2.1 Functional Requirements

- Find the characteristic points in the profile-view face image
- Find the facial expression in the profile-view face image

3.4.2.2 Non-Functional Requirements

- Display this points on the image
- Add the algorithm as a module for ISFER

3.5 Constraints

- The algorithm has to work with both Linux and Windows.
- The language of programming is C/C++
- The images are in the BMP 24 bits format

3.6 Anticipated Project Schedule

- Week 1-2: Study the ISFER programs, read some manuals, read some application reports
- Week 3-8: Write the algorithm for the simple type of image
- Week 9: Test the algorithm
- Week 10: Modify the algorithm to make it work with the other type of image (general case)
- Week 11-16: Improve the algorithm with the new type of image
- Week 17-18: Find and compute the characteristics values of the profile-view face for the 200 pictures of the database.
- Week 19-20: Find and implement the Bayesian Network for the expression recognition
- Week 21-22: Improvet the algorithm and recognition
- Week 23: Test the algorithm
- Week 24: Write the report

4. Conception

The first step in the research of the facial expression is to extract the contour of the face profile silhouette. This depends a lot on the images I have to work with. So we need to do some preprocessing on the images to standardize them and make the contour research easier.

4.1 The images

On the beginning of the project, the images I had to work with were very simple. It was black and white images with no noise (I mean than each pixel can take just 2 values: black or white) and perfectly cut on the neck and on the head.

Figure 2 : the 1st type of images

With this kind of image the extraction of the contour is very simple. You just have to find for each line the first pixel not white.

Then I had to pass to more complex images like this:

Figure 3 : the 2nd type

On those ones, the face profile is localized in a little part of a colored image and there is lot of noise. Furthermore there are different kinds of skin with different colors and the algorithm has to work with all of them.

Figure 4 : Another example of the 2nd type

Then, we have to apply some filters on the 2nd type of images to make them be more or less similar to the 1st type.

First we have to cut the image in order to keep only the face profile. This is useful to earn some time and not to do useless processing on some parts where there is nothing interesting. It is also helpfully to find some points, like for example the chin. You can use the fact than the chin is localized in the bottom part of the image to localize it more easily.

Then maybe a rotation can be carried out, because the subjects took different postures on different images. This is also important because some features points correspond to extremities in the face profile contour. Well then the choice on the x-axis for the image engenders the determination of those points. So they depend a lot in the rotation carried out.

Finally a filter used for the contour research is applied. It has to work with different types of skin and has to be as insensible to the noise as possible.
For the research of the contour, it exists several algorithm based on convolution products between the images and different operators. The most famous of them are the operators of Roberts, Prewitt, Sobel and Kirsch. The most used is the Sobel one, which is enough in the majority of the cases.
But these filters are very sensitive to noise. So we have to apply before, an algorithm to reduce it. In this goal, the median filter can be useful. It filters the image with median of given size. Depending of the images you work with, the size of the filter can be modified to be perfectly adapted.

After these treatments, the images should be standardized and look like the images of the 1st type except for the white and the black that should be inversed. Anyway, the research of the face profile contour can be managed.

4.2 Research of the contour

The contour of the face profile corresponds to an important variation in the gray level. The variation might be distorted by some noise, especially with the images I worked with. So maybe this research will not be so simple as we will see in the next chapter.

4.3 Find the points

According to some documentation, the features points are supposed to be extremities in the face profile contour.
But how those extremities can be found? Different approaches were possible.

The simplest way was to calculate the first derivative of the contour using a formula to approximate it in a non-continuous space. Then calculate the extremities as null values of this first derivative. This should give us all the extremities points of the contour and particularly the researched points.

Another one was to approach the contour by a function like a polynomial or a template. The problem with the approximation by a function is precisely that you do some approximation and this destroys completely the structure of the contour and then the extremities too. The template seems more interesting but this approach is first complex, introducing some probabilities, but mainly, very

expensive in time. This approach has already been tried for the research in a front view face. This is more logical because of the complexity of this research. There are so many variations from a face to another even from the same person but with different expressions that an algorithm that not use a probabilistic way to find some characteristics parts like eyes or mouth, has few chances to work.
For the face profile, these variations are quite less noticeable. Everybody have more or less the same profile. There are of course, lots of variations in the shape of the nose for example, but this doesn't matter with the presence or not of the extremities, which are more or less in the same places.
Furthermore this approach works well with the front view face, but it's very costly in time. So this method has been abandoned.

Others methods exist and had been developed by different research laboratories but they are also very complex and according to me useless.

I choose finally the first method to find my features points on the images, namely calculate the 1st order derivative and find the points which value is null for this function.

I calculate the pseudo 1st order derivative using the formula:

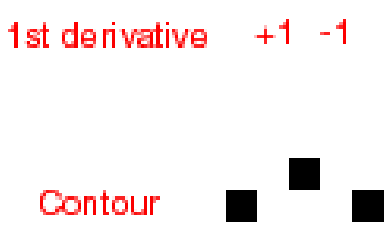
$$\frac{\partial f}{\partial x}(x_i) = \frac{(y_i - y_{i-1})}{(x_i - x_{i-1})}$$

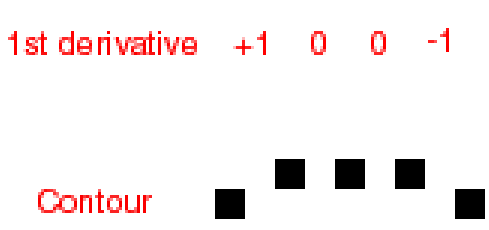
On this case, the value of $x_i - x_{i-1}$ is 1.

This is also the simplest one. I could use a more complex one using 2 points around the pixel involved but this one seems enough for this work.
We can remark as well that this is the calculation of the 1st order left derivative, which can be different than the right one knowing than the space is not continuous. So we have to analyze both 1st order derivatives before to say if a point is an extremity or not.
Furthermore as we work in non-continuous space, the value of the 1st derivative is not necessarily zero on the extremities. This even appears in just few cases, often when the derivative is null, the point don't have to be taken.

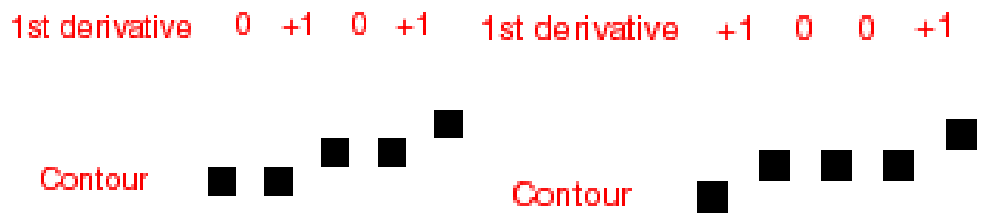
Let's see some examples.

Here are some examples of contours where we can see an extremity:





Here are examples of contours without extremity:



The first case is very simple; the scope is positive before the point and negative after. As it could be the opposite for the minimums, we have to check the sign of the product of the two 1st order derivatives for each pixel. When this product is negative we have found an extremity.
In the 3rd case we can see that 2 points have one derivative null but are not extremities. In the 2nd case, for the point of the middle, both derivatives are null, and it's also the case in the 4th picture. One has to be taken, but not the other. Then find one or 2 zeros is not a guarantee of an extremity point.
To look at the variation on the scope gives better result in this research. When you find a zero, you keep the value of the scope of the precedent not-null one. Then when the scope passes by a non-null value you have to look at the sign of the product between this value and the precedent not-null value of the derivative. When this product is negative you have to take a point between the 2 pixels involved (3 choices in the 2nd case). This choice of the point should not very important; we could take it randomly. But I chose to take the point in the middle, because I think it's the best choice to represent an extremity.

However after having work a bit with the images, I could see that all the points do not correspond to extremities in all the cases. For some points like the nose, the eyes and the eyebrow, the criterion is reached, but for the points around the mouth and for the chin, it can happen that they do not correspond to maximum or minima in the contour. This is due to the posture taken by the persons and the characterization of their faces.

This is a big problem that can be solved by a rotation of the image. But which rotation do you have to apply? This algorithm has to work alone with lot of different images and a 'magic rotation' that could work with all of them, doesn't exist.

An idea was to calculate the 2nd order derivative using the same formula than for the 1st one but applied to the 1st derivative function. Indeed, when some features points are not extremities in the contour, they are often (or a point very closed to it) inflexion points, and then the 2nd order derivative should be null on those points. However, due to the non-continuous nature of the function, the calculation of those inflexion points generates lot of candidates to the features points. See Fig 5 for an example:

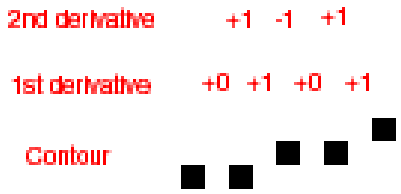


Figure 5 : Example of problem

This contour is an example of the representation of a continuous line in a non-continuous space. It means than for us it seems like a line and then does not possess inflexion points. But if you calculate the 2nd order derivative you can see that it pass by 0 (the function pass from 1 to –1 and back to 1, so there are 2 inflexions points between them). So at least one point is found by the research (you can take arbitrarily the one before or after the 0) but has nothing to deal with a feature point. This case appears a lot in the contours I worked with, so this method is not very nice because it brings to many points.

But after some research, I saw that the points of the mouth correspond to extremities if you consider the line linking the nose and the chin as the x-axis. Have a look to Fig 6 and 7 where it's perfectly visible.

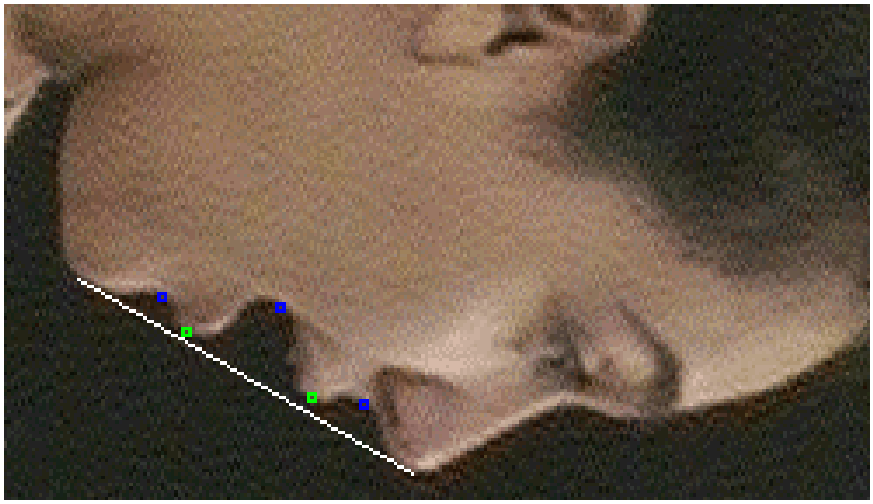


Figure 6 : The features points as extremities of the contour

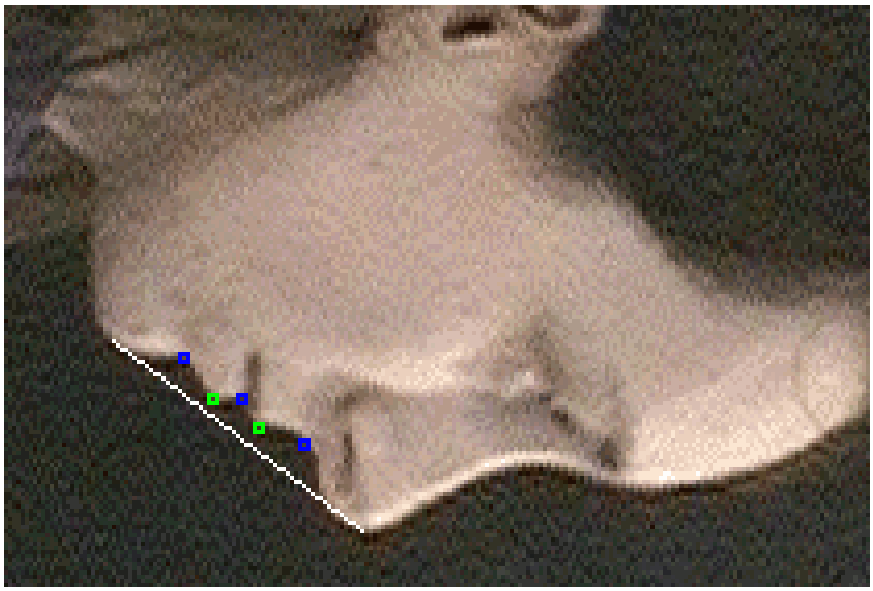


Figure 7 : Another example

We can see than in the 2nd case, some points doesn't correspond to extremities if you take the normal x-axis, but are well there if you change this axis for the line linking the nose and the chin.

Then the first step is to find the nose and the chin.

The nose should not be a problem; it's in all the case, an extremity in the face profile contour. It appears as well in the middle of the images and corresponds to the pixel with maximum value in y-axis, if you consider just a section of the image where the high part is excluded (because sometimes the hair causes problems).

For the chin, the research is more complex. It is not all the time an extremity of the face profile contour, depending of the physical characteristics of the person, his posture, the expression took, etc.

Because we don't know a priori if the chin will be or not an extremity, we have to find some criterions to valid or not the first research, which searches a candidate in the extremities found.
First we can use the physical character of the chin. This character is the big variation in the scope of the contour just before the chin. So we can check that such a variation exists.
When this criterion is reached, another one is used. It is the distance between the nose and the candidate. If this distance is too short, the candidate is invalidated.

On this case, the chin has lot of chance not to be an extremity in the contour. So another research that does not use the extremities found firstly, has to be carried out.

I found an idea for this research, it's to replace the normal x-axis by the line linking the nose with the up-left corner of the image; the chin appears this time as an extremity, see Fig. 8.

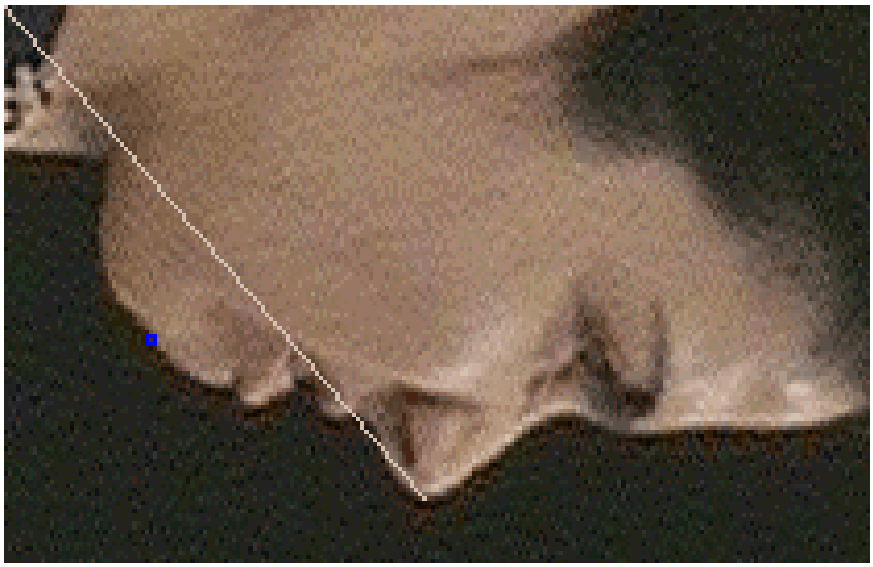


Figure 8 : The chin as an extremity for a special line

But instead of taking the nose, we can take as well the candidate invalidated by the precedent research. Indeed, this point is the first maximum after the chin (which is supposed not to be a maximum on such a case) and we can see that the chin is still a maximum for the contour if you take the line linking the invalidated candidate and the pixel on the corner of the image as x-axis. We also have a criterion very simple; the chin corresponds this time to the pixel with maximum y-value. This supposes a good cut on the beginning, not to have too much things in the neck that could interfere with the research.

After that, the research of the others points between the nose and the chin should be easier.

I noticed than the projection of the pixel-mouth on the line linking the nose and the chin is more or less placed in the middle of this line. This can be a criterion for finding the mouth that is furthermore a minimum in the contour.
Then, find the lips and jaws should be easy; they correspond to the firsts maximums and minimums around the mouth.

For the eyes, the first minimum with the normal x-axis should correspond to it. We can add a criterion as well, checking than the candidate is not too close to the nose.

Finally for the eyebrow, it corresponds to the first maximum after the eye.

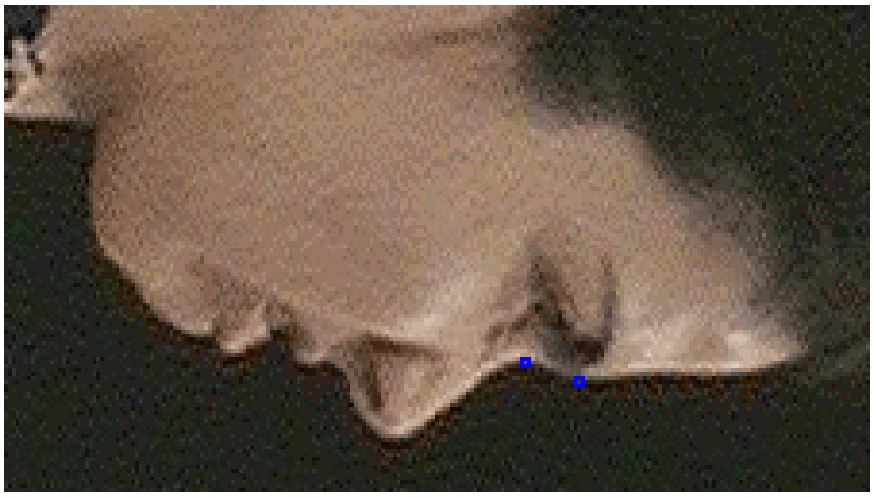
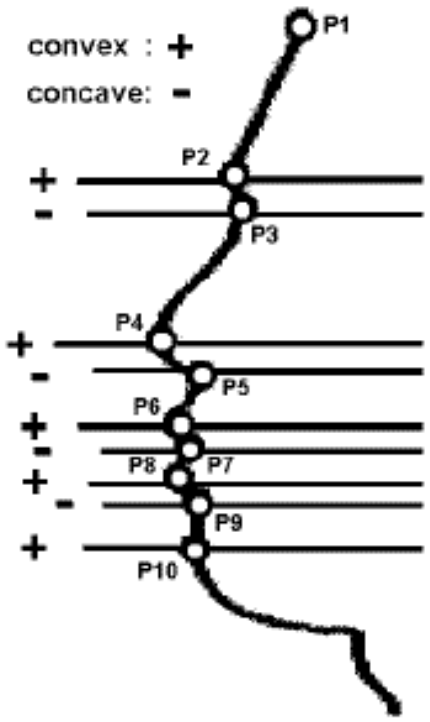


Figure 9 : Eye and eyebrow as extremities

4.4 Parameters of expression

In the goal to recognize an expression, we first have to find some characteristic values deducted from the features points found in the precedent part.



P1: top of the forehead
P2: eyebrow arcade
P3: root of the nose
P4: tip of the nose
P5: upper jaw
P6: uper lip
P7: lips attachment
(if tongue is visible:
P7 : tongue-lip attachment
P7' : tip of the tongue
P7'': tongue-lip attachment)
P8: lower lip
P9: lower jaw
P10: tip of the chin

I found some, but I don't think they're enough to recognize all the expressions:

- Distance between the eye and the eyebrow (p2 and p3). This characterizes the feature raised or waned of the eyebrow.
- Distance between the lips (p6 and p8). This characterizes the opening of the mouth.
- Angle of the mouth (angle between p6-p7 and p7-p8). This characterizes as well the opening of the mouth.
- Distance between the upper lip and the upper jaw (p5 and p6). This characterizes the feature fold or not of the upper lip.
- Distance between the bottom lip and the bottom jaw (p8 and p9). The same than before but for the bottom lip.

All this values have to be normalized to be independent of the physical characteristics of the subject. The factor of normalization found is the distance between the upper lip and the eye (p6 and p3). This factor was found experimentally.

Then we have to calculate those values for the maximum of pictures possible classified by expression. We could so deduce some characterization for each expression.

After this work we can recognize an expression by comparing the compute values and the registered values for each expressions. For that we can use a Bayesian network, which compute the probability for a given variable to be in a state knowing the value of the others variables and the conditional probability associated.

5. Implementation

This part describes the implementation of the algorithm and the different structures used.
First let's look to the structures used in this program: TGrayImage, TImage, Tpixel, TintArray and TPixArray

5.1 Class TGrayImage - TImage

To work with the images, we need an interface enabling to grant directly to the pixel or permitting to do some treatment. We need, for example, to recover the value of a given pixel, to write a value in a given pixel, to do some treatment on the images like filters, etc. On this goal, an interface was given. I created or modified some functions but the major part was already programmed. Two types of images are present: gray images and colored ones. These class are defined in img.cpp / img.h

```
class TGrayImage
{
        UINT_ x,y;
                                BYTE_ *img;
        BYTE_ cg;
private:
public:
        TGrayImage():x(0),y(0),img(NULL){}
        void LoadFromGrayImage(TGrayImage*);
        ~TGrayImage();
        //--------
        INT_ AllocImage(ULONG_,ULONG_);
        void FreeImage();
        //--------
        void setbground(BYTE_);
        INT_ putpix(UINT_,UINT_,BYTE_);
        INT_ getpix(UINT_,UINT_,BYTE_ *);
        INT_ MedianImage(TGrayImage*,UINT_);
        INT_ ChangeRes(TGrayImage*,UINT_);
        void Roberts(TGrayImage *);
        void Prewitt(TGrayImage *);
        void Sobel(TGrayImage *);
        void Kirsch(TGrayImage *);
        INT_ ConvolImage(TGrayImage*,TGrayImage*);
        INT_ LoadMask(char*);

        void setPenGrayLevel(BYTE_);
        BYTE_ getPenGrayLevel(void);

        void DrawGrayLine(int,int,int,int);
        void DrawGrayBox(int,int,int,int);

        void ClipImage(TGrayImage*,int,int,int,int);

        //--------
```

```
        INT_ LoadBMPGrayImage(char*);
        INT_ SaveBMPGrayImage(char*);

        INT_ ToColorImage(TImage*);

        UINT_ getdx(){return x;}
        UINT_ getdy(){return y;}
};


class TImage
{
public:
        INT_ x,y;
        BYTE_ *r,*g,*b;
        BYTE_ cr,cg,cb;
private:
        void fputRGBPixel(UINT_,UINT_,FILE*);
private:
public:
        TImage():x(0),y(0),r(NULL),g(NULL),b(NULL){}
        void LoadFromImage(TImage*);
        ~TImage();

        INT_ AllocImage(ULONG_,ULONG_);
        void FreeImage();

        INT_ LoadBMPImage(char*);
        INT_ SaveBMPImage(char*);

        INT_ ToGrayImage(TGrayImage*,float rw,float gw,float bw);
        INT_ putpix(UINT_,UINT_,BYTE_,BYTE_,BYTE_);
        INT_ getpix(UINT_,UINT_,BYTE_*,BYTE_*,BYTE_*);

        void setColor(BYTE_,BYTE_,BYTE_);
        void getColor(BYTE_*,BYTE_*,BYTE_*);

        void DrawLine(int,int,int,int);
        void DrawBox(int,int,int,int);
        void ClipImage(TImage*,int,int,int,int);

        INT_ LoadFromGrayBuffer(UINT_,UINT_,BYTE_*);

        UINT_ getdx(){return x;}
        UINT_ getdy(){return y;}
        void getr(BYTE_ *p){memcpy(p,r,x*y);}
        void getg(BYTE_ *p){memcpy(p,g,x*y);}
        void getb(BYTE_ *p){memcpy(p,b,x*y);}
```

```
void setr(unsigned long k,BYTE_ p){r[k]=p;}
void setg(unsigned long k,BYTE_ p){g[k]=p;}
void setb(unsigned long k,BYTE_ p){b[k]=p;}
};
```

I especially used the gray images because the sources were in black and white.
But I used the colored images as well, to visualize some special points with different colors, which is better than with gray levels.
We can pass from one to the other by functions ToGrayImage(TGrayImage*,float rw,float gw,float bw) and ToColorImage(TImage*). rw, gw and bw are the multiply factors for the 3 different colors, red, green and blue.

To create an image, the function AllocImage(ULONG_,ULONG_) can be used.

To load and save BMP colored images it exists 2 functions LoadBMPImage(char*) and SaveBMPImage(char*). You just have to give the name of the image as the first parameter of the function.

putpix(UINT_,UINT_,BYTE_) and getpix(UINT_,UINT_,BYTE_*) gives access to the pixel define by the 2 first parameters. putpix put the gray level given by the 3rd parameter (for TImage there are 3 parameters for each color red, green and blue) in the given pixel. getpixel take the value and put it in the pointer given as the 3rd parameter (for TImage there are 3 pointers to give).

To draw some boxes or lines you can use the functions DrawGrayLine(int,int,int,int) and DrawGrayBox(int,int,int,int). The parameters of these functions are the coordinates of the pixels that define those boxes and lines. First you have to choose the value of the gray level of your box. This can be realized by the function setPenGrayLevel(BYTE_). It registers the value given as a variable of the image and need to be called just one time for each boxes or lines of the same color.
You have exactly the same functions for the class TImage using setColor(BYTE_,BYTE_,BYTE_) instead of setPenGrayLevel(BYTE_).

To clip an image, the function ClipImage(TGrayImage*,int,int,int,int) can be used. The image-result is register in the first parameter. The other parameters define the coordinates of the corner pixels in the new image.

You can apply as well some filters in the images:
- MedianImage(TGrayImage*,UINT_) filters the image with median of given size (2nd parameter). For each pixel, the gray level in the image-result is the median value of the gray level of the pixels contained in a square around it. This function is specially used to reduce the noise in the images.

- ChangeRes(TGrayImage*,UINT_) reduces the resolution of an image. The 2nd parameter is the factor of reduction (for example a factor of reduction equal to 2 means that the image-result will have her size divided by 2)

- ConvolImage(TGrayImage*,TGrayImage*) applies the convolution product between the image on which the method is applied and the image given in first parameter (kernel). The result is saved in the third parameter.

- Roberts(TGrayImage *) applies the Roberts operator to the image. The result is saved in the 2nd parameter. The Roberts operator is an approximation of the first derivative; it corresponds to a pair of 2×2 convolution kernels:

+1	0
0	-1

Gx

0	+1
-1	0

Gy

- Prewitt(TGrayImage *) applies the Prewitt operator to the image. This one correspond to a pair of 3×3 convolution kernels:

-1	0	+1
-1	0	+1
-1	0	+1

Gx

+1	+1	+1
0	0	0
-1	-1	-1

Gy

- Sobel(TGrayImage *) applies the Sobel operator to the image.

-1	0	+1
-1	0	+1
-1	0	+1

Gx

+1	+1	+1
0	0	0
-1	-1	-1

Gy

- Kirsch(TGrayImage *) applies the Kirsch operator to the image. This operator corresponds to a 8×8 convolution kernels.

5.2 Class TPixel

Manipulating the images, it could be interesting to have a structure for the pixels. This is defined in the module pixel.h/pixel.cpp

```
class TPixel
{
  float x, y;

 protected:

 public:
  TPixel(float a=0, float b=0):x(a),y(b){};
    ~TPixel(){};
    //--------------------------------
    float getx(){return x;}
    float gety(){return y;}
    void set(float a, float b){x=a; y=b;}
    float distanceEucl(TPixel *p);
    float projection(TPixel *p, TPixel *p2, TPixel *res);
};
```

A pixel is just an array of two floats. Maybe it's seems strange than I chose float instead of int for pixels of images, but I did that because I also used this structure for temporary pixels which correspond to normal ones in an image, but after a rotation. So for these images, the values of pixels are no more integer and then I need a float structure.

The constructor Tpixel initializes the values of the pixel with the parameters given or in default by 0.

getx() and gety() give the different values of the pixel.

set(float a, float b) is used to change the values of the pixel with the values given in parameter.

distanceEucl(TPixel *p) is a function that calculates the Euclidean distance between this pixel and the pixel p. I just applied the formula of the Euclidean distance in dimension 2:

$$D((x_1, y_1),(x_2, y_2)) = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

The function projection(TPixel *p, TPixel *p2, TPixel *res) is used to calculate the projection of the pixel to the line passing by p and p2. The result is given in the pixel res. This time I applied the formula:

$$x_{res} = \frac{a}{a^2 + 1} \cdot (y + \frac{x}{a} - b)$$

$$y_{res} = a \times x_{res} + b$$

a and b are the parameters of the line equation passing by p and p2 : $y = a \times x + b$

So $a = \frac{(y_2 - y_1)}{(x_2 - x_1)}$ and $b = y_1 - a \times x_1 = y_2 - a \times x_2$

5.3 Class TintArray

This class defines an array of int. It is used in the MedianImage(TGrayImage*,UINT_) function in the TGrayImage class to filter the image with median of given size.
It's defined by 3 variables, his size, his maxSize and the classical array of C (pointer on int)

```
class TIntArray
{
        unsigned maxSize,size;
        int *data;

protected:
        int AllocArray(unsigned);
        void FreeArray();

public:
        TIntArray();
        ~TIntArray();
        //--------------------------------
        void reset();
        int  add(int);
        //--------------------------------
        int  MedianArray();
};
```

Each time you add an integer to the array, the value of size is incremented. When Size and maxSize are equal, the maxSize of the array is increased by a value specified in the beginning of the file:

```
#define BLOCK_ALLOC 128
```

On this case this value is 128. It means that, when the number of elements in the array reaches 128, the size of the array is increased to 256. Then when it reaches 256, the size is incremented to 384, etc.

The function AllocArray(unsigned n) allocates n boxes in the array.

The constructor TIntArray() calls AllocArray with the variable BLOCK_ALLOC as parameter.

The function add(int) is used to add a integer at the end of the array.

reset() is used to reinitialize the array. It changes the value of the size to 0.

Finally MedianArray() returns the median value of all the value contained in the array.

5.4 Class TPixArray

This class defines an array of pixel. The structure is similar to the array of integer defined in the precedent section

```
class TPixArray
{
        unsigned maxSize,size;
        TPixel *data;

protected:
        int AllocArray(unsigned);
        void FreeArray();

public:
        TPixArray();
        ~TPixArray();
        //-------------------------------
        void reset();
        int  add(TPixel);
        void print();
        void getPixel(int, TPixel *);
        unsigned getSize(){return size;}
};
```

Like for the array of integer, each time you add a pixel to the array, the value of size is incremented. When Size and maxSize are equal, the maxSize of the array is increased by a value specified in the beginning of the file:

```
#define BLOCK_ALLOC_PIX 25
```

On this case this value is 25. It means that when the number of elements in the array reaches 25, the size of the array is increased to 50.

The function add(TPixel) is used to add a pixel at the end of the array.

getPixel(int i, TPixel *p) gives the pixel indexed by i in the array. The result is saved in the pixel p.

print() displays the values of the pixels contained in the array. It was used for some testing and is not used in the final version.

reset() is used to reinitialize the array. It changes the value of the size to 0.

Finally, getSize() returns the size of the array (the index of the last pixel in the array).

5.5 Algorithm

After the definition of the different class used, we can start to implement the algorithm for the research of the feature points.

5.5.1 Working with images

Figure 10 : Example of picture

The first thing to do is cut the image to get only the face profile like the first images. The user, who has to give the coordinates of the pixels of the box, does this operation manually. This choice has been done because of the complexity of such algorithm. This part is very important for the following because the research of the chin depending a lot from the cut chosen.

A call to the function ClipImage of the class TgrayImage with the correct parameters give the good part of the image.

For our example, the pixels of the border of the box are (600,200) and (730, 470). So we get:

Figure 11 : After the cut

After that I choose to rotate the image with an angle of -Pi/2 radians. There is not special reason for this choice; it's just a question of comfort because on such case the contour of the silhouette is found by a research on the x-axis. For some images where the person is not perfectly right, we also need to rotate the image with a different angle. This angle is calculated manually and has to be given by the user. The rotation is done by the call of the function rotateImage, rescaleImage and fillImage.

- void rotateImage(TGrayImage *g, TGrayImage *g2, float Theta)

This function creates an image witch size is $2\times\sqrt{d_x^{\,2}+d_y^{\,2}}$ with (d_x,d_y) as sizes of the first image. The origin of the first image is placed on the middle of the second one and the rotation is calculated.

Example with an angle of –Pi/3 instead of –Pi/2

Figure 12 : After the call of rotateImage

- void rescaleImage(TGrayImage *g, TGrayImage *g2)

This function clips the image, deleting the lines or columns where there is no interesting pixel (they are all white). It calls others functions : findPointBlackColumn and findPointBlackLine. These function research the first point under a gray level given in a column or line given. These functions take another parameter; it's the line or the column where the research has to start, because it exists some errors at the border on some pictures.

Figure 13 14 : After the call of rescaleImage

- void fillImage(TGrayImage *g)

This function fills the image by putting black pixels in the top of the image and pixels of the same color than the last pixel not white in each column for the pixels in the bottom.

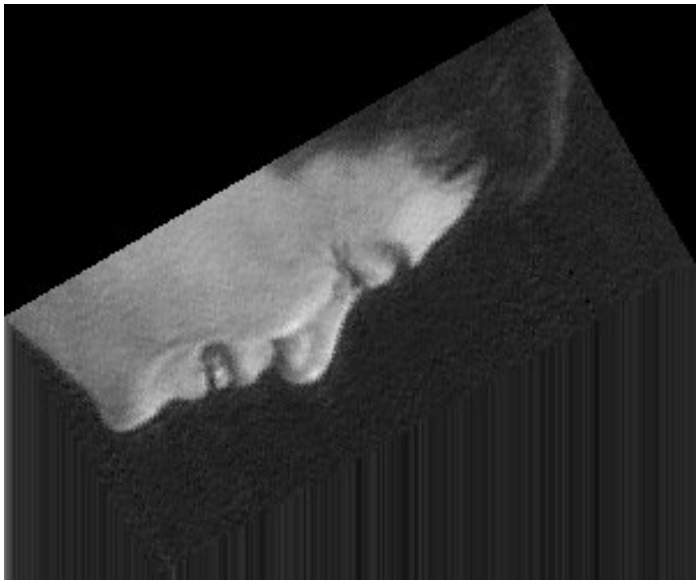


Figure 15 : After the call of fillImage

This choice was made because it does not create noise after the application of the median filter.

Now we have to apply a filter which result doesn't depend on the type of the skin of the person. It exists different ones based on an approximation of the first derivative. Roberts, Prewiit, Sobel and Kirsch operators are usually used for this kind of research of contour with a preference of the Sobel one which results are often good.

But these filters are very sensitive on noise. Let's try to apply the Sobel filter without any reduction of the noise:

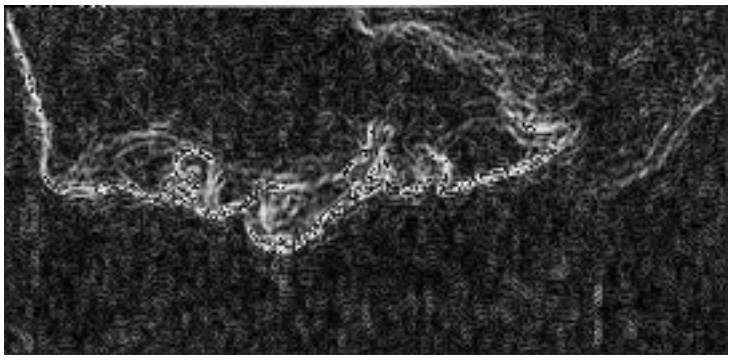


Figure 16 : After a Sobel filtration

As you can see, the research of the contour is now almost impossible. The noise has been increased by the application of the filter. Then we have to apply first a filter to eliminate the noise.
I applied a Median filter that filters the image with median of given size. The size has been chosen by different tests, and the result start to be good with a value of 6.

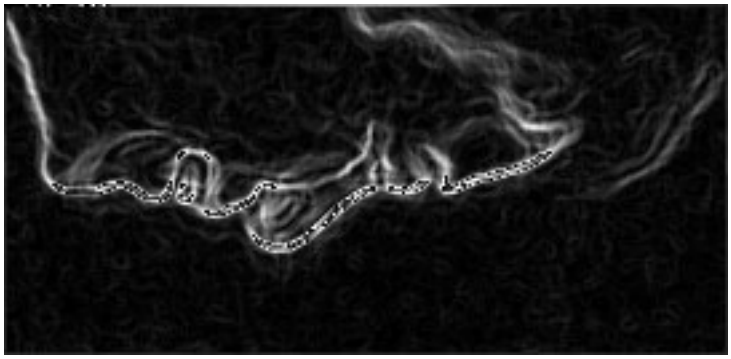


Figure 17 : Median filter with a size of 5

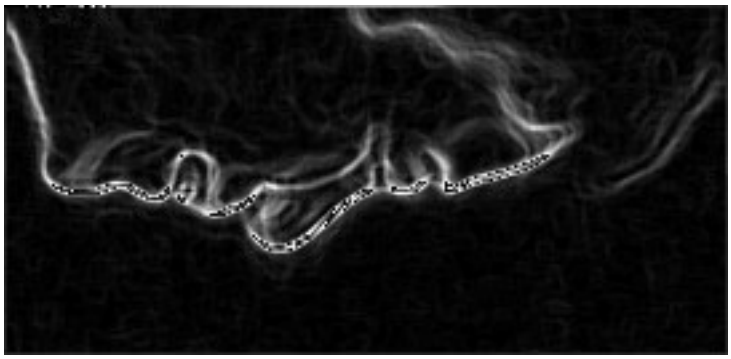


Figure 18 : Median filter with a size of 6

Let's try the algorithm with a picture of a black person:

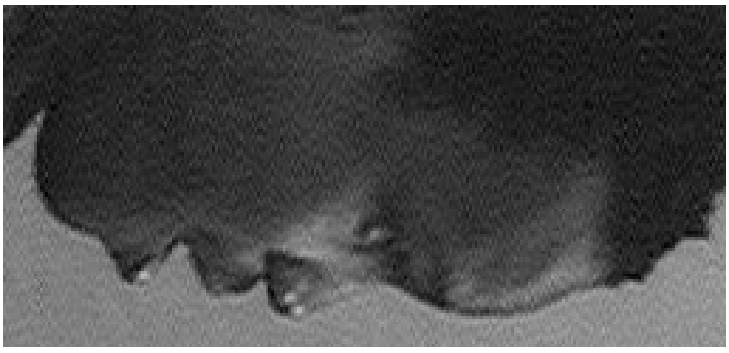


Figure 19 : Another example

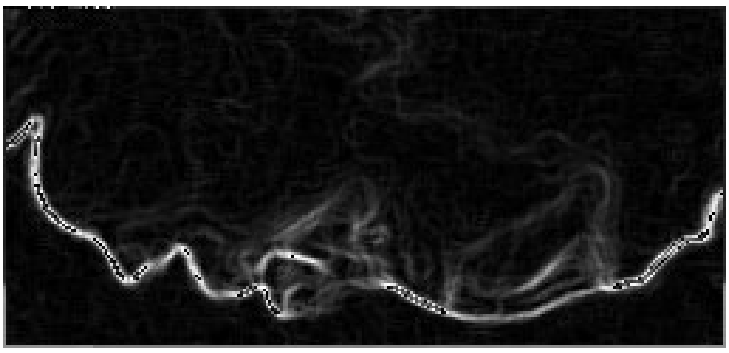


Figure 20 : After the Median and Sobel filters

We can observe than the result is good and can be exploited for the following.

5.5.2 Extraction of face profile contour

Starting with the Sobel-filtered images the research of the contour is much easier. So the background of the images is almost black and the contour of the profile is almost white depending on which part we are. A simple research of the first pixel the gray level value of which is under a certain value should give us the contour sought.

With this image:

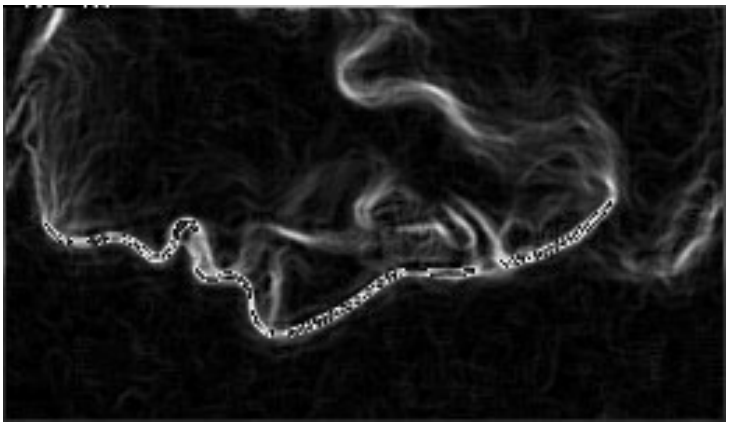


Figure 21

The research of the first pixel under the following values gives these results:

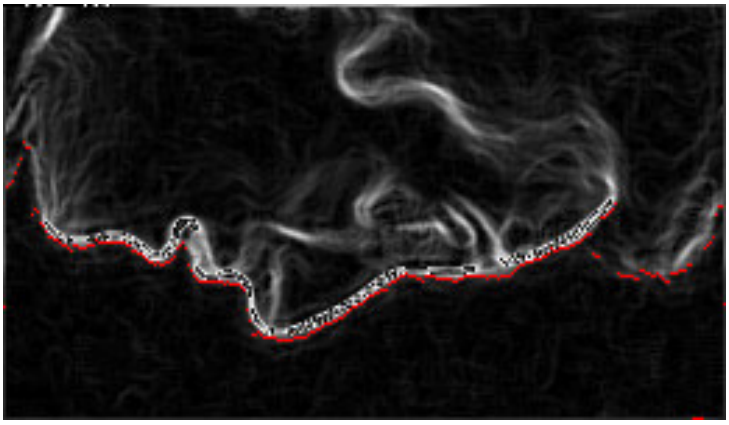


Figure 22 : First pixel for each column the gray level of which is under 45

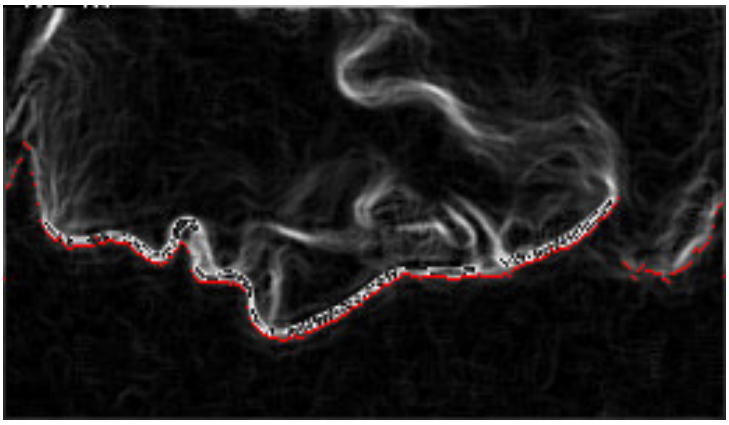


Figure 23 : First pixel for each column the gray level of which is under 55

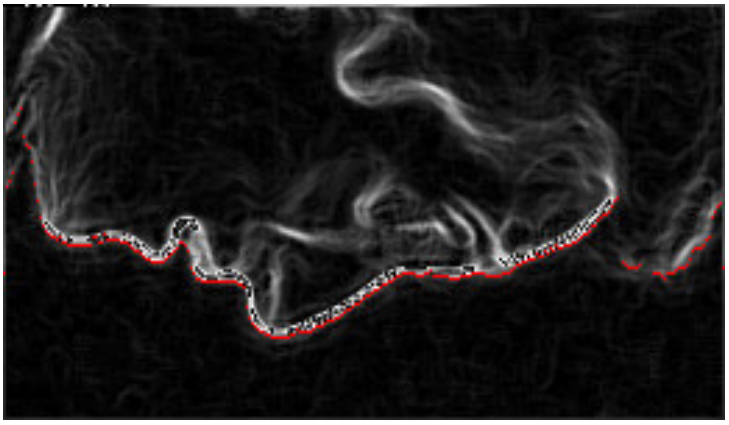


Figure 24 : First pixel for each column the gray level of which is under 65

I finally decided to choose 55 as the minimum value of gray level. This value wasn't chosen haphazardly, lot of test with different values enables to find the good one.

This value is defined as a global variable in algo.h

```
#define GRAY_LEVEL_MIN 55
```

The research of the contour is done by the function findPointsColumn(TGrayImage *g, INT_ xmin, INT_ xmax, INT_ border_error, int gray_level, TPixArray *res). The parameters xmin and xmax are the coordinates of the columns between which the research is done. border_error is used to jump the first lines where there is some noise. gray_level is the value of the minimum gray level for the research of the points (so in this algorithm we pass the variable GRAY_LEVEL_MIN). Finally, res contains the results of the research and is so an array of pixels.

This function is using findPointWhiteColumn(TGrayImage *g, INT_ x, TPixel *p, INT_ border_error, int gray_level) which researches the first pixel which gray level is under the value of the parameter gray_level for a given column (the 2nd parameter : x).

Now than we have the contour of the face profile, the extraction of the features point can start. The 2 first points to extract are the nose and the chin. On this goal, we have to find the extremities points of the contour. Then we have to calculate the pseudo 1st order derivative of the contour.

5.5.3 Find the extremities

This work is done by the function findExtremum(TPixArray *data, TPixArray *res_max, TPixArray *res_min). The parameter data contains the list of pixel representing the contour; res_max and res_min are the list of the pixels results, the 1st one for the maximums and the 2nd one for the minimums.
The function calculate the pseudo 1st order derivative using the formula:

$$\frac{\partial f}{\partial x}(x_i) = \frac{(y_i - y_{i-1})}{(x_i - x_{i-1})}$$

On the case we deal with, the value of $x_i - x_{i-1}$ is 1.

Then the algorithm looks at the variations of this function and takes the points when the function passes from negative value to positive one or the opposite. If on this case the function is null for more than one point, the point taken is the one of the middle. Moreover the function looks after a serious variation in the scope, certainly due to a pixel of noise and doesn't take care of it. Finally if one extremity is too close to another one (1 pixel), it is not registered.

Have a look on Fig. 25, that represents the points found by this part of the algorithm.

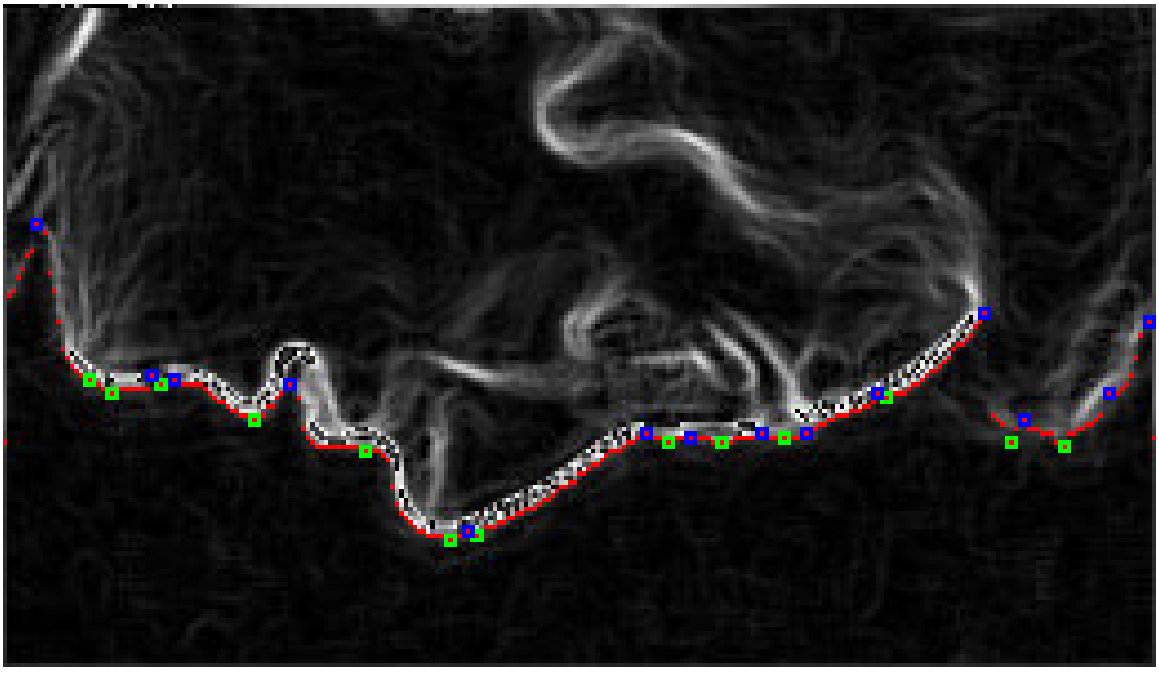


Figure 25 : Extremities found, green for maximum, and blue for minimum

5.5.4 Extraction of the nose

For the nose, I choose an easy criterion; it's the maximum with the highest value of y-axis excluding the 2 exterior quarters of the image (if the x-size of the image is 100, the nose is localized between the columns 25 and 75).
In this goal I wrote a function findNose(TPixArray *data, TPixel *res, int min, int max) which the two last parameters are the minimum and maximum value of x for the pixel-nose (in our case it's *size/4* and *3*size/4*, if *size* is equal to the x-size of the image).

5.5.5 Extraction of the chin

For the chin the criterion is not so simple as we will see. For the research there are different functions:

- findChin(TGrayImage *g, TPixArray *data, TPixel *res, int min_y, int max_y, int min_var)
data contains a list of maximum (found by the precedent algorithm in our case), res is the pixel result, min_y and max_y are the minimum and maximum value of the y-axis than can take the result, min_var represents the minimum variation in the y-axis in the research, it's explained in the following.
The algorithm looks at the variation of the y-axis of the points of the contour between 2 maximums, and sums it when it's positive. If the variation is above the value of the variable min_var, the point is found and the research stops. This criterion was chosen because of the character sketch of the chin, there is a big variation of the y-coordinates just before it. This characteristic is used as a criterion.
The parameters chosen for the call of the function are 40 for min_y to exclude the points of the neck, the y-value of the nose for max_y to exclude eventually some pixel of noise, and 25 for min_var that is always reached for the images I worked with.

This algorithm could work well if the chin was always detected as a maximum in the face profile contour, but actually, it's not often the case. A criterion is used to see if the research with the function findChin is a success or not, it's the distance on the x-axis between the nose and the pixel found by this function. This value has to be above 63 for the images I deal with. It's not a very good criterion knowing that it depends on the images I work with, but I didn't find a better one.

Example:

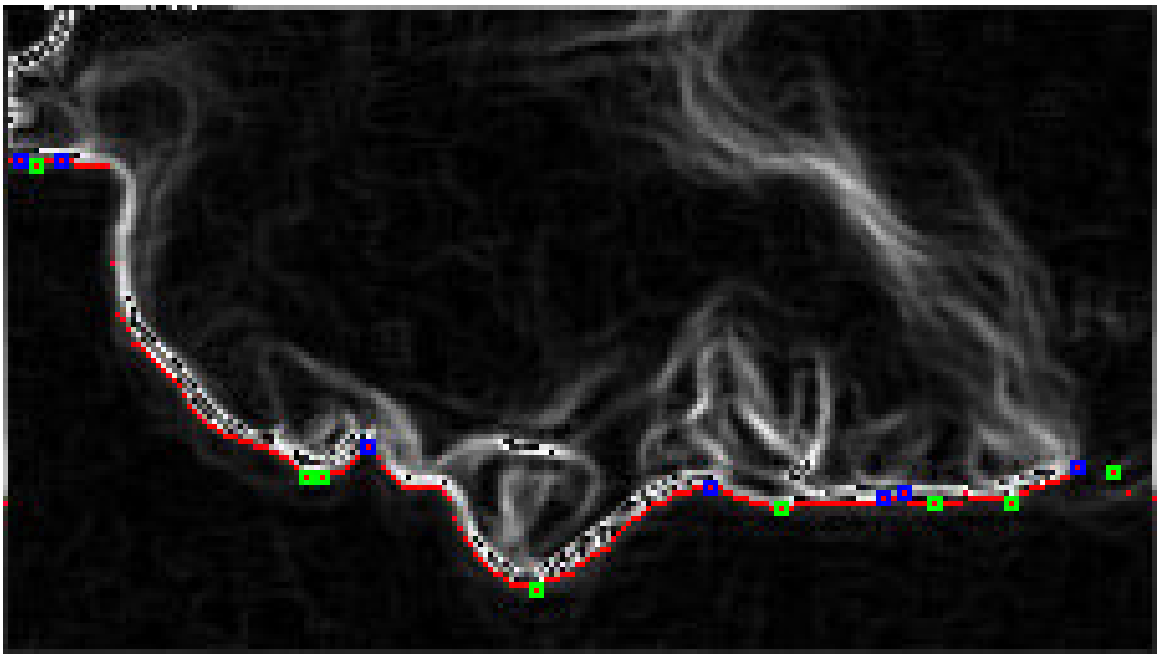


Figure 26 : Extremities found

In this image we can see than the chin is not found as a maximum for the contour. And we can also say than even for somebody (eq. not a computer) it's not very easy to place a point for the chin in this kind of case.

Then, if the criterion is not reached, the algorithm calls another function:

- findChinUsingSlopingLine(TGrayImage *g, TPixel *p1, TPixel *p2, TPixArray *res_max, TPixArray *res_min, TPixel *res)
This function calculates the extremities of the contour, replacing the x-axis of the image by the line linking p1 and p2, the pixels given as parameters to the function. The extremities found are saved in res_max and res_min, and the candidate for the chin is saved in the variable res. This function saves all these results because it's also used in the following to extract the other points and not just the chin.

First this function rotates all the points of the image in the goal to have the line (p1, p2) parallel to the x-axis. Then it finds again the contour of the silhouette using findPointsColumn with this new x-axis. After that, some points are excluded, when the variation in the y-axis is too much important from a pixel to another (above 10 pixels) – i.e. it's a pixel of noise. The function findExtremum is next called and the extremities found are then rotated back. The chin is localized as the pixel with maximum value in the y-axis after the 1st rotation.

The 2 pixels given as parameters to the function are the pixel found by the call of findChin and the pixel origin of the image (0,0). This choice is not all the time perfect, because it depends a lot with the cut done on the first part of the algorithm. But I had no details for the choice of the cut, so I chose this.

On the example, findChin return the 2nd green point on the image. So the function findChinUsingSlopingLine is called with this point and the point origin (0,0). With these parameters the chin is found as the following:

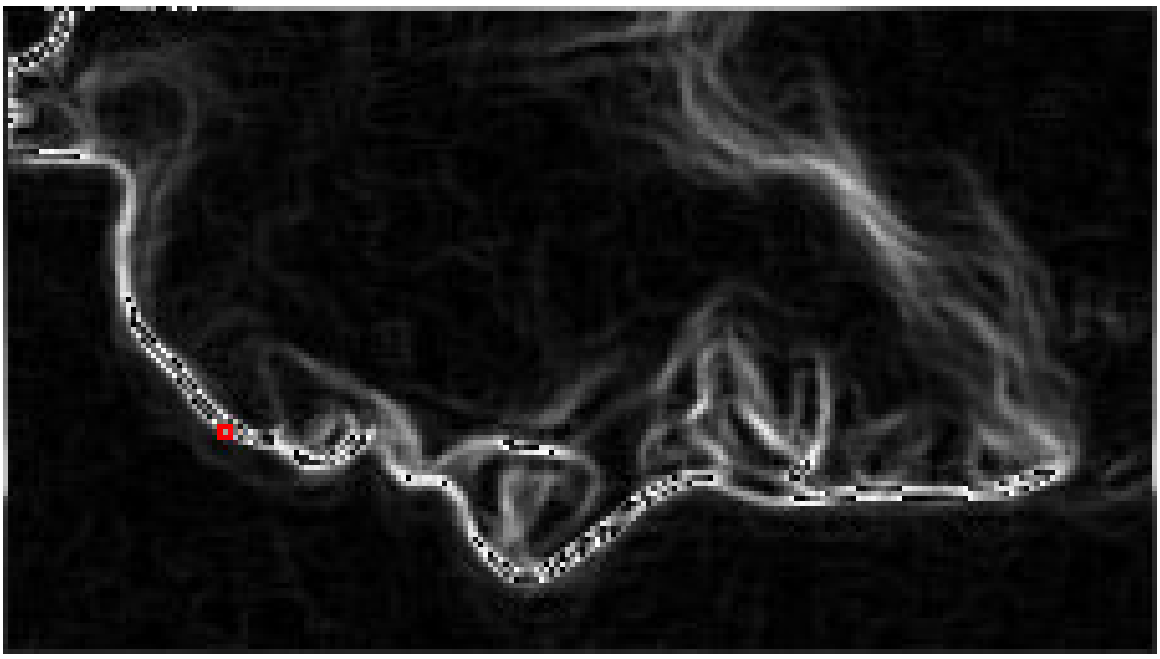


Figure 27 : Chin found

We can see than the point found corresponds more or less to the chin.

This part is very important in the algorithm because the nose and the chin are used in the following to find all the points between them (lips, mouth, and jaws).

5.5.6 Extraction of the mouth, lips and jaws

Assuming than the extraction of the nose and the chin passed well, the extraction of the points around the chin can start. The first step in this extraction is to calculate the extremities points after a rotation of the image intending to have the line linking the nose and the chin parallel to the x-axis.
The operation is done by the function findChinUsingSlopingLine, we have already seen in the precedent chapter. The pixels given as parameters are this time, the nose and the chin found firstly.

In our example the contour after rotation is:

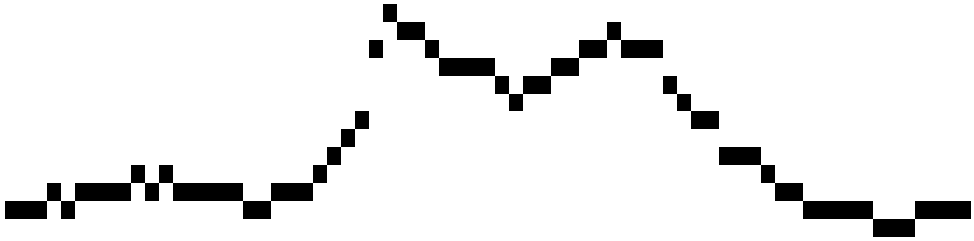


Figure 28

And the extremities found by the algorithm are:

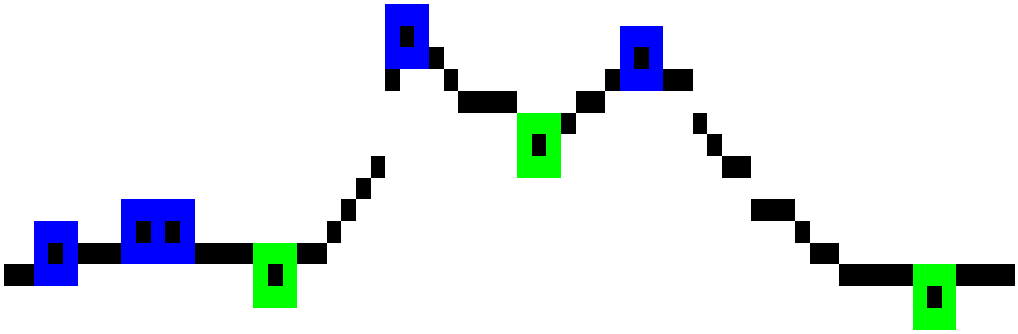


Figure 29 : Extremities found, maximum in green, minimum in blue

Once this research done, another function is called to determined which extremities correspond to the features points. This function is findPointsUsingMouth:

void findPointsUsingMouth(TPixArray *data_max, TPixArray *data_min, TPixArray *res)
The parameters are data_max and data_min, which contain a list of extremities, and res, which will contain the result of the research.
The function finds the minimum pixel (in blue on Fig. 28) which distance between its projection on the x-axis and the pixel placed on the middle of the nose and the chin, is minimum. This corresponds to the mouth. After that the algorithm search for the first pixels maximum and minimum before and after the mouth. Each of them corresponds to a feature point of the profile face.

5.5.7 Extraction of the eye and eyebrow

To find those points, we first call 2 functions:
- findPointsColumn with the x-value of nose as the first parameter defining the column of the search and the x-size of the image as the 2nd one. So the result is the contour of the face profile after the nose.

- findExtremum with the contour found in the precedent call as the 1st parameter.

Then the eye is found as the first minimum in the result given by the call of the 2nd function. But sometimes some noise can create an error in this research. To by-pass this, the algorithm calculates the distance between the nose and the point find using the function distanceEucl of the class pixel. If this value is under a certain value, the algorithm goes ahead, and tries to find another one. The experimental value found for this minimum distance is 25.

The eyebrow is next found as the first maximum after the eye.

Then all the features are found, except P1, the top of the forehead that is not important to find the expression, and it's also difficult to place it.

5.6 Results

I test the algorithm in several images from the database of the KBS. It was colored images with different persons and different expressions. Furthermore each expression (except neutral) was shown in different ways for each person.

For some expressions like neutral, surprise, fear, happiness the algorithm works well and doesn't need often a manual correction. But for the other ones like disgus, anger and sadness where the mouth is closed and can have a strange shape, the algorithm is this time not so good, and there is often some mistakes in the lips. This is due to some pixels of noise that appears on those situations and the shape of the lips whose the extremity character is not very well marked.

But it's very difficult to evaluate such an algorithm.
First because it's already difficult to place the points manually on a face profil. When you have to place them, you don't know very well if the point is there or in another pixel after or before.
Then because the efficient of the algorithm depends a lot on the cut done on the beginning to define the region where the face profil is localised. If the cut is bad, the algorithm can fail to find the chin, and then all the points found after are wrong. For 2 different cuts on the same image, you can have really different results. As I do this cut manually, and as this cut changes on each image, it's then difficult to evaluate the effectiveness of the algorithm.
Another raison is that the algorithm can be good to find 8 points but do a mistake on the 9th. It doesn't means that the algorithm is bad, but it needs sometimes manual correction.
The last raison is that the goal of the algorithm is to find the expression. So the research of the points is important, but the expression can be found even if all the points are not found or not very well placed.

Finaly the points are found without mistake on 50% of the images. It corresponds often to one of these expressions : neutral, surpise, fear and happiness. On 30% of them, the points found need some manual correction. It can be the case on just one point or two. For the rest of the images, the result are bad, often due to noise pixel close to the mouth.

6. Improvement

Some improvement to the algorithm can be carry out, first directly on the algorithm, by reducing the noise or find better the points, then by adding some parts like the expression recognition and finally include this in the ISFER program.

6.1 Java Native Interface (JNI)

It's possible to create a module to ISFER by using the JNI. It's very simple; with this tool, you can use C++ functions in Java programs. You can pass as well values or pointers from C++ programs to Java programs.
So my algorithm can be used for the research of the points and give them back to the ISFER program that is coded in Java. Then you can display the points found by the algorithm and ask for manual verification and modifications.
I tried to implement this but I finally abandoned because of the complexity of the ISFER program and the time which I still had on the project and which was not sufficient.

6.2 Find the expression

I started a little bit this part, but I didn't have enough time to finish it.
Anyway I compute the different features values from 200 different images, using my algorithm and manual correction in case of placement error.

The next step is to compute the different probabilities for an expression to be present knowing the different parameters. For example if the 2nd parameter (distance between the lips) is big, there is a large probability than the expression took is surprise. These probabilities are computed by studying the known values on the 200 images. You have to find some connection between the different values to deduce the important ones for each expression, and an interval of values for each parameter where the probability for an expression to appear, knowing that the parameter belong to this interval, is maximum. This asks lot of work and lot of correction before to find the good interval for each parameter.

Once this work done, the recognition of the facial expression could work. It depends if the parameters I chose are enough and well chosen. I didn't have enough time to check this, but I think there are not enough to recognize all the different types of expression.

6.3 Elimination of the noise

I'm not completely satisfied with the reduction of the noise made by the Median filter. This destroys a bit the contour of the profile introducing new extremities where they weren't before. This is of course very bad for the algorithm.
So I think this part could be improved and then having a better approximation of the contour.

6.4 Find the points

Of course all the algorithm can be improved by finding better criterions for each point found and then check better if the points found are good or not. This work can be particularly carried out for the research of the chin that depends too much on the manual cut done on the beginning of the algorithm. But this remark is also valid for all the features points in particular the mouth.

7. Conclusion

This project was really wonderful. It was an enrichment both from the point of view of the work and on the social plan. This training period abroad will remain certainly unforgettable.

The work in TU Delft was really interesting (the most I have ever done for the moment), I learned lot of things in different subjects (treatment of image, Bayesian network, programming tips) and a different work approach.

I met as well lot of people from all Europe with different knowledge and culture, and I enjoyed a lot to live in a multi-cultural place such as that one. It was really the best moments of my life.

Finally I want to thank again M. Leon Rothkrantz who allowed this dream to become reality.

8. Glossary

9. Bibliography

10. Annexes

The Source Code can be downloaded at the following address:
http://www.enseirb.fr/~georgesf/thirdyear/project/source.zip

The compute of the different features values from the 200 different images are there:
http://www.enseirb.fr/~georgesf/thirdyear/project/stat.xls