〔非公開〕

ＴＲ－Ｍ－００３１

Handling of Human Motion

フィリップ　フェベ
Philippe FEBVAY

井上　誠喜
Seiki INOUE

１９９８．１．３０

ＡＴＲ知能映像通信研究所

Handling of Human Motion

Philippe FEBVAY

Contents

Chapter 1

Introduction

Coming in ATR, I knew that my studies here would be around motion tracking. I did not know a lot about it, and I had tried to get some information about it before coming to Japan. But I just had a look at some results in some books and that was all, because this domain is a very large one.

During the first month here, I learned to use my computer, to program in C some simple image processing routines and read about algorithms that could be of use in order to perform motion tracking. I understood after a short time in ATR that such motion tracking was part of a more larger area which was computer vision. Attending various talks about the subject, I could see how important such a domain was.

As my study was about handling of human motion, I decided first to try some methods based on color detection. Because the interesting parts to be tracked on human body are especially the face and the hands. It seemed that recognizing skin color would not be too heavy at a computational level. Finally, I was able to get some zones in the image, these zones being either the face, either the hands or some noise.

At this moment, I had to make some choices for the recognition part of the processing. And it showed up that this part was the crucial one. Depending highly on the conditions of use. I had no idea if I could make it real time or not.

That's why I went to video programming. I needed to get used to video in order to make some tests on it. And that's what gave birth to the template matching algorithm. I tried this algorithm because it was quite a simple one which was based on difference between frame and would bring me to video. As this method was performing better than expected, I spent time testing it and trying to improve it.

Then I came back to color and video. And I applied the color detection on video and observed the frame rate. It seemed clear that the processing was not so quick and that therefore, there would be a problem for real time, because the recognition requires further treatment. And also how to perform recognition remained unclear to me. It appeared that the recognition was linked with the conditions of use of this color detection, because reliable recognition would mean lots of calculations.

It is at this moment that I began to go in the studio. Because the tracking algorithms were performed to run there. So they had to run there. Consequently, I had to make some changes on the programs, because the machines were differents. It is during that time that I had the idea of making a boundary extraction of a person in the studio so that I could get high level information about the shape. As it will be discussed more in the following pages, I gave up this idea after a while.

From that time, I began working on possible applications for template matching: one in the studio, and the other being a motion database making.

Consequently, I will speak about the three methods I had the idea to apply during my stay at ATR: the color detection, the boundary extraction and the template matching. I finish with template matching because the further applications use this method.

Chapter 2

Some precisions on graphics formats

The image formats I used during my 0 were various, especially to deal with video flows.

2.1 The PPM file format

To save images on the hard disk, I chose the PPM format, because it is very simple do deal with. There is no compression, and the structure of the data is consequently very simple.

- File organization

  The PPM format is designed to be as simple as possible. Each starts out with a header, and the bitmap data follows immediately after. The header is always written in ASCII, and data items are separated by white space (blanks, tabs, carriage returns or 0). The data portion of each file type can be written 0 ASCII or binary form.

- File details

  There are 2 versions of the header. Although the header are in ASCII format, one is used for the ASCII version of the format, and the other is used for the binary version.

  Magic Value : Literally P3 for ASCII version, P6 for binary. Image Width : Width of image in pixels (ASCII decimal value) Image Height: Height of image in pixels (ASCII decimal value) MaxGrey : Maximum color value (ASCII decimal value)

- Image Data

  After the header is a series of lines describing width * height pixels. For PPM, each pixel contains three ASCII decimal values between 0 and the specified maximum value, starting at the top-left corner of the pixmap, proceeding in normal English reading order. The three values for each pixel represent red, green, and blue, respectively; a value of 0 means that color is turned off, and the maximum value means that color is "maxxed out".

  You can also include comments in the PBM file. Characters from a character to the next end-of-line are ignored. There is a suggested maximum of 70 characters per line, but this is not an actual restriction.

2.2 Video formats

Many times I had to cope with a video flow. Most of the time, the representation of every frame changes slightly according to what kind of function you use. mainly, I had to manipulate those two types :

- RGB
- Hue Y

Even if the data are coded in RGB, following the C-library you are using, the organization of the components change slightly, which can lead to stupid errors. In every program manipulating a video data flow, a function was written in order to, given the (x,y) coordinates of the image, access the R,G or B component.

Chapter 3

Head tracking by color detection

Tracking deformable objects such as face or hands can be done in a first time very quickly using color information. The main issue is to find a good representation of the human skin, so that it is after easy to decide for every pixel of the image 8 it belongs to a patch of skin or not. Some color spaces are more adapted than others for this recognition.

3.1 Elements of color theory, choice of a color space

Human skin can be caracterized by its hue and saturation. The intensity, however, will vary as a fonction of the relative direction to the illumination. Of course, the perceived hue and saturation will be the product of the color of the ambient light and the skin color.

So, the most important thing will be to try to diminish the influence of the illumination as much as possible. That is why the RGB components returned by the camera must be somehow computed to change of color space.

The first simple idea is to normalize the color for changes in intensity. This can primarily be made by dividing the values RGB by the luminance. It is also to notice that the blue component is of small interest. Therefore, we obtain a color space (r,g) normalized which is already of interest for the detection.

I also tried the color space Hue-Saturation to check if the detection was increasing significantly, but the computational loss of time in the calculations was not worth as we will see in the results.

3.2 How to recognize skin once we have a quite good caracterisation of it

The candidate pixels for faces and hands can be detected very rapidly using a normalized color histogram. The histogram is initialized by observing a patch of skin and counting the number of time each normalized color value occurs. The histogram contains the number of pixels which exhibit a particular vector (r,g). Dividing by the total number of pixels in the histogram gives the probability of obtaining a particular vector given taht the pixel observes skin.

Bayes rule can be used to determine the probability of skin given the vector (r,g):

$$P(\text{skin/vector}) = P(\text{vector/skin}) * P(\text{skin}) / P(\text{vector})$$

- P(vector/skin) is found in the histogram.
- P(vector) : probability to find this vector on the image. We can suppose that each vector has the same probability. In that case, P(vector) is constant. This is not completely true but the approximation is quite good. P(skin) is constant also on the image so that the processing on evry pixel is only a multiplication by a constant value.

Finally, we are given a quite good approximation of P(skin/vector).

3.3 Implementation

See the source files for that.

However, the basic idea for the histogram is easy: I chose a 2 dimensionnal array. Given H[12][24] for example, you would get the probability of (12,24). For initialization, when you encounter (x,y) (r,g) value for a given pixel, you just increase H[x][y] by one. At the end, you divide everything by the number of pixels in the image.

3.4 Results on still images; Comparison

Figure 3.1: Using RGB

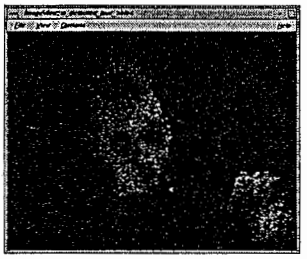

Figure 3.2: Using H-S

In these images, you can see that some points of the background may interfere in a very disturbing way. This is the case because these images were taken from my desk, where conditions are not controlled. On the contrary, in the studio, the background is blue, so there is no problem of this kind any more.

3.5 Now dealing with camera and real-time video

Well, the change is not very big. In fact, when you use video, you, finally, after others many lines of code to open the video flow, get a pointer to the first pixel of the image. Increasing this pointer, you can get wherever you want on the image, on any pixel and any Red, Green or Blue component. The only concern is sometimes to find a good way to access this data easily, and to discover the structure used to describe the image, which is changing very often. Otherwise, everything works the same. You just have to initialize the histogram once at the beginning and then perform the color detection on every frame just as if it was an image.

Running the (r,g) color detection system on video, here are some frame rate results depending on the window size:

- 320x200 : 10 fps
- 200x120 : 28 fps

3.6 Face recognition

At this stage, we have performed the detection of what is supposed to be skin. But the computer has still no idea of where the head could be. The image is still only a serie of pixels.

3.6.1 Bluring the image

So, first, it makes the color detection better to blur the image resulting from the color detection. On the one hand, this will erase some pixels which were found as skin color but are isolated, and on the hand, this may link some other pixels which were part of the same patch of skin but were separated during the color detection. It seemed that applying a local median filter with a window size of 3x3 is the best way to make the bluring. Calculating the mean would have again cost too much computer power, and in the case of video, too much time.

On Fig 3.3, you can see the result after this step:

Figure 3.3: Image after bluring

3.6.2 Connecting the pixels

Once this is done, we have to find zones in the image which could be the face. The first step is to define zones of connected pixels. For that, we used a connecting algorithm which returns the patches of pixels in the image. We deliberately keep only the 3 biggest zones of the image, because the smallest one are necessarily (or so we hope) noise. We kept only 3 because we assume that the face must be in the 3 biggest patches, and that one hand or two hands are also likely to appear on the image, maybe larger than the face. In order to perform the connectivity algorithm, we have first to choose which kind of connectivity we are going to use. We finally chosed the Right, Left, Up and Down connectivity definition (4 pixels are neighbours of the pixel on focus), which was enough.

In order to perform this color detection, we have to use a grid representing the image and memorizing for every pixel if it has been assigned a zone or not. Then, we begin by the top left of the image and begin to scan. When we find one pixel supposed to be skin, we lunch a recursive function on all the neighbours that checks if they are skin pixels and, if they still don't belong to any other skin patch (thanks to the grid), add them to the current one (and thus put them as belonging to a patch in the grid). When the whole patch is finished, we memorize a

pointer to the tree generated, and also the number of pixels in the tree. Once the whole image has been scaned, thanks to that size information, we only keep the 3 largest areas.

You can see a result after this step on Fig 3.4

Figure 3.4: definition of zones

3.6.3 Recognizing the head

Finally, we had to find a way to recognize among the 3 candidates which one was the face. For that, as we just have a pixel based information, the most obvious way is to make a shape analysis using statistic methods such as moments and so on.

The main problem with moments is that they are quite long to calculate once you go to a high order. Also, all of them have a particular physical meaning, but not specifically for face shape of course. Consequently, the best way to use them is by feeding a neural network with them, and after this neural network was trained with various faces, it should be able of recognition with a good probability. But this is not really possible to do it real time.

I went to another, simpler, solution. Just by calculating the (x,y) coordinates of the center of gravity of each zone and then calculating variance along x and y axises, you can compute quickly the y_variance / x_variance value caracterizing roughly the shape of the object. This can provide you with a basis recognition.

For my face, the value appear to be around 0.6 depending of the image of course. Among the 3 zones you have, you would then choose the one which is the nearest of 0.6

I am aware that this recognition is somehow hazardous and should be improved. That's why I didn'y try to run it real time, but only on screenshots.

3.7 limitations

- Conditions of Use.

  1. Simple Background

     As we already pointed it out, the color recognition is reliable only if the background is not too complicated with colors approaching skin color. Otherwise, this step will completely fail. Some people in

ATR also work on color recognition with no real time needs. They can then sacrifice time to apply heavier calculations to get in an another color space, but they face the same problem though.

2. Histogram initialisation

   The initialisation of the histogram changes with every person which must be tracked. Every time you want to track somebody, you have to take a picture of the person's skin in order to initialise the histogram. Even if it is always the same person, it is better to do so because the lighting conditions may change a lot between 2 different experiments (although I tried to diminish the effects of lightings conditions)

3. Good lightings conditions

   Which is: not too dark because everything would be black and not too bright because everything would be white.

- Speed limitations

  The whole process is too slow if applied to a whole image.

  Because making the color detection needs a path on every pixel of the image. Then bluring would require roughly 8 paths on every pixel. So, as bluring is not essential, I discarded it without a too big loss in results. The system looses robustness though. Anyway, you cannot escape the connection of the pixels. It requires at least another path in the entire image. And then you have to calculate information about the shapes, which requires you again to access memory (but not on the entire image fortunately) and to make scientific calculations. Consequently, this processing, even if it is very light, cannot be done real-time on a 640x480 image.

  The alternative is to make it locally. When you have done it at the begining at a high time cost and have then found the face, it is possible to repeat it only in a local area where the face will certainly be the frame after. This, so, was not implemented yet because I had to focus on another method, using template matching.

Chapter 4

Motion tracking using template matching

In order to get used to video programming, I decided to make a template matching algorithm, because it seemed quite simple. This kind of algorithm, contrary to what I began to do with the color detection, is deeply based on motion because it uses the difference between two images, the current one and the previous one. Surprisingly, this method gave unsuspected good results for certain tracking tasks.

4.1 Basic idea of template matching

In template matching, the intensity profile of the entity we seek to match the image forms a template. The process consists in searching for regions in the image where the image grey-level (for example) and the template grey-level regionally coincide. The template is defined in a template-specific coordinate system, as g(x1,x2), and the image function, again in terms of a chosen coordinate system, is denoted f(x1,x2). Assume that g(x1,x2) is nonzero only over a region, in the template system. The determination of a good measure of match, or metric, is important. The value of the metric should be large when the template and image region coincide in intensity levels over R, and small otherwise. For analysis purposes, we may wish to use an alternate measure of mismatch. The template g is shifted over all possible locations in f, and a measure of match is computed at each of these locations. The following are two candidates metrics to indicate mismatch (indices are omitted for simplicity):

```
m1 = SS|f-g|
m2 = SS(f-g)^2
```

4.2 What was used

To use the template matching algorithm for realtime tracking purpose, we had to make some choices. We chose the m2 metric. Given the first frame of video, we initialize the template. Once this is done, template matching can begin with the next frame. Of course, it is not possible to search the whole image for the new template position, because they would be as many comparisons and metric calculations as they are pixels on the video image. So we have to decide first of a searching range around the current position. Then every frame, the new assumed position of the template is chosen by minimizing the metric value over the searching area. Once the choice is done, the template is of course re-actualised with the new current position elected. And the process goes on forever.

A basic idea of the implemetation is given by the following "bma.c" C-function.

The real-time implemantation needed to change slightly this function. Because the images we dealt with was from video data which are organized in a single array. So it required first to write a function, quite simple though, which enable us to access directly to a given pixel in memory. As we also dealt with color image, we got color data in the video flow, which means that we could'nt handle directly with intensity of the image.

Several solutions were possible:

- either calculating the intensity after having read the R,G,B components with the formula :

$$I = 0.3R+0.59G+0.11B$$

- either selecting one of the components R,G or B and make the metric calculation only on this component.

After comparison of the two solutions, it appeared that calculating the intensity was not really interesting, because it was not more robust and took more time. Finally, after comparison between the three components, it appeared that the Green component was the one performing best.

4.3 Solving unexpected problems

That being done, it was possible to have a first idea of the results. It appeared at once that the algorithm was not useable like that, for two main reasons. Those problems we were able to solve.

1. Noise on the image

   As I wouldn't have suspected before, the video quality of the image is not perfect. Consequently, every pixel value slighlty varies over time, even if this is a still pixel. For example, if you record a movie of a still scene (my desk on that case) ,extract 2 different frames which in theory are exactly the same, and display the color histogramm of the 2 images, you will be able to see many differences between those 2. In order to solve the problem, the only obvious solution was to introduce a thresholding in the decision process. First, given the size of the template, we had to quantifize the amount of maximum disparity around a value V. This being done, on the final step of the decision for the new position, we apply the rule: if m2 minimum value ¡ V, we don't move and return a (0,0) motion vector. The fact that m2 doesn't equal zero is surely because of the noise on the image. oterwise, we accept the value and move.

   The only problem this kind of decision brings though is that the algorithm becomes less precise and less reactive to small variations on the template. It can happen that, if the template observes a patch of skin and there is a real movement of this patch, that the template needs time to react, because of the thresholding.

2. Speed problem

   Another annoying issue was the fact that the template could not follow too quick movements. Because if you move too fast, the frame after, the real new position of the template will be out of your searching range. Consequently, the new position decided will be completly wrong and the tracking will be lost. Of course, you can increase the searching range to solve that, but this would result of a drop in the framerate and makes the program useless.

   In order to improve the reactivity of the template, I decided to search for the new position, not around the present position, but around the suspected new probable position. I decided to use this very simple method to predict the new position:

   Given a particular frame, if the template is moving at a certain speed, that speed won't have changed a lot between the 2 frames (because of the frame rate of 30). Consequently, the framerate being constant, the change of the motion vector won't be important. So, knowing the previous motion vector, it is highly probable that the new position will be around the actual position + motion vector and we will search around this point. After having made that change, I realized the tracking was much better and was know able to follow good speed motions.

   The problem is that it still suppose that the motion is smooth enough. If the motion becomes too jerky, even with this slight improvement of the basic algorithm, the tracking will be lost. Consequently, as above method is based on speed considerations, and tried some acceleration based methods, but they lead to instable results. Furthermore, the tracking was not more reactive to rapid movements and finally it was clear that the speed based method was the best.

4.4 Conclusion on this algorithm and applications

You can see a definitive version of it in the annexes.

It appears finally that the tracking is especially good to follow edges, where the intensity has a very particular important variation along the edge. Also, the main problem of that method is that when a new position is decided, the template is updated and its contents changes. So, when time goes by, after a few seconds, the new tracked template is not at all the same as it was at the begining.

Nevertheless, we can find some applications to that method. These are the ones I worked on:

- virtual mouse.
- database tool.

Chapter 5

Boundary extraction

At one step of my studies, it appeared that getting the boundary of a person could be useful to get high level information about the person moving in the studio. I was thinking specifically of distance of the person to the camera, and, which would have been even more interesting, by correlation with some pre-known boundaries, being able to recognize some positions. Even if I had no precise idea of which correlation fonction choose, it seemed to me that it should be very possible to make it, even if not real time, at least at a reasonable speed, because boundaries are not made of too many points. Therefore, some local boundary extraction could also give interesting information about head position. So, I decided to go for it.

5.1 Algorithm

I designed my own algorithm, which afterwards showed up to be the one currently used, or very near to it. Fisrt, a step must be performed which has segmented the image in 2 areas: the background, and the person. This can be done by color considerations, as in the studio the background is blue.

The basic principle is the following :

Given a point of the boundary, you have to choose next one. If we decided to turn the trigonometric way for example, given the current point of the boundary and the previous one, turn around the current one, starting from the previous one, as shown in this image (Fig 5.1):

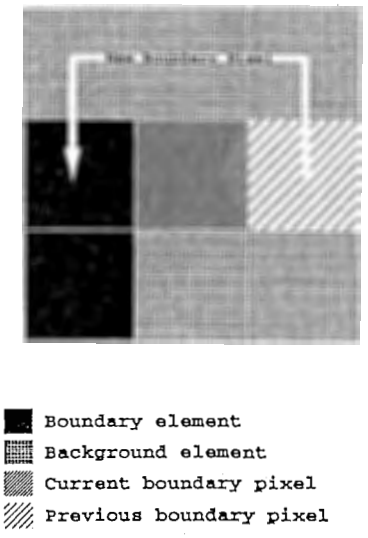

Figure 5.1: Selection of next boundary pixel

Consequently, the initialisation of the process requires us to choose the 2 first elements. First will be the first boundary element we can get. Starting from the middle of the image, we can get it easily following the principle shown on the next image. (Fig 5.2)

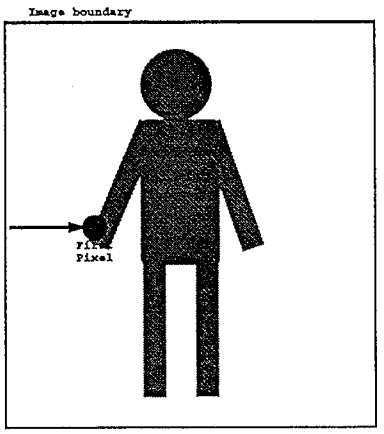

Figure 5.2: Selection of next boundary pixel

Then, we have to choose the second point. It is computed following this principle (Fig 5.3) :

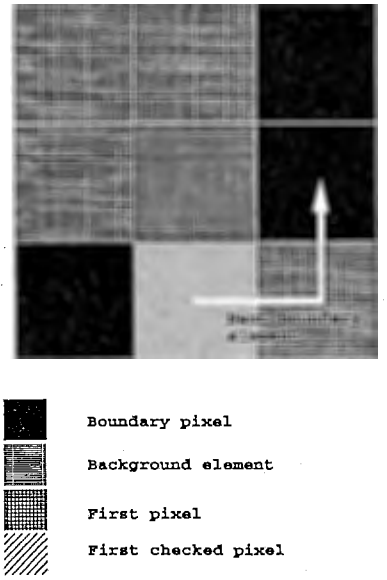

Figure 5.3: Selection of next boundary pixel

5.2 Modification due to camera image

The previous algorithm run on an image taken from a movie gives the following result (Fig 5.4):

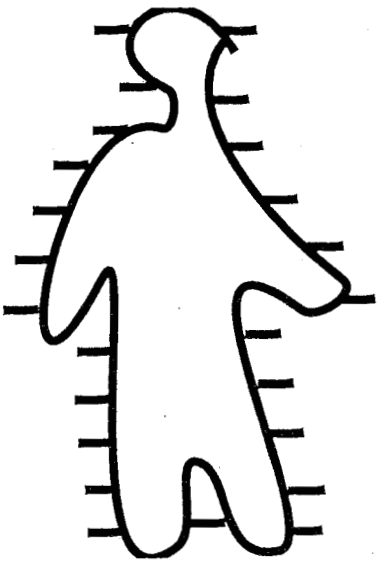

Figure 5.4: Speed effect

There is "a speed effect". That's because every video image captured by the camera is in fact made of 2 different interlaced fields. It would be possible to run the program every field, but as the standart for movies is 30fps made of 2 fields, I decided to modify slightly the algoritm so that it can deal with the problem.

In order to solve this, I have to add a "second image" in the program during the processing, so that for every pixel of the image already saved as a boundary pixel, it is marked on that image. Now, every time you add a new pixel in the boundary chain, you check these data and if this points already belongs to the boundary, it is that all you have saved in the boundary from last time you went by that point is a line due to "speed effect". So, you discard these last points from the boundary. The result is a smoothing of the boudary, and the "speed effect" is not visible any more.

Here is an image given by this method :(Fig 5.5 and)(Fig 5.6)

Figure 5.5: Image before boundary extraction

As for the result (Fig 5.6), it appears that it is not perfect because of the difficulty of blue color detection, especially around the body. That's why the boundary detected is always "fatter" as the real boundary, as you can clearly see thanks to the next image :(Fig ??)

5.3 Further use

This boundary extraction could be made real-time. I tried to do it but it did not well, and crashed after a few seconds because the blue color detection was not good enough (some blue pixels are still present from time to time and lead to a crash). But I learned recently that Dvs6000 was able to perform the blue color detection itself realtime. So, using this ability, it should solve the crash problem due to a bad color detection.

Figure 5.6: Image after boundary extraction

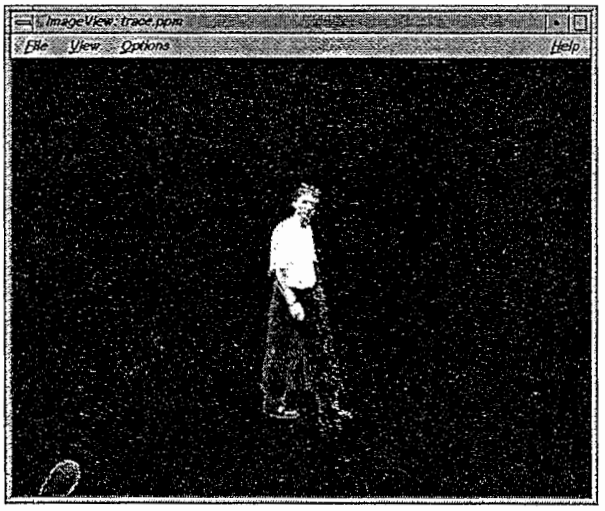

Figure 5.7: Image after blue color extraction

Chapter 6

Applications

These are the applications which were finally developped with the tools I tried to produce. Both of them are based on the template matching algorithm. The first one could be improved if linked with other treatment, such as color detection or boundary extraction. But this has not been done yet.

6.1 Virtual mouse

6.1.1 Basic idea

The basic principle is shown on the video. If the person tracked can get an idea of where the template is actually, it is very easy for that person to move it with hands or fingers. Therefore, the square can act as the pointer of a mouse. If there were a mean to emulated the mouseclick, we would have a perfect virtual mouse. In fact, this would be possible to make it by performing a local color detection around the template position. The main issue would then be to know for example wether the hand is closed or opened. It is not sure however that color information would be a sufficient information to make the decision. But other treatments, heavier to a computational level, could be imagined: since we would already know where the hand is, the treatment could be only local.

6.1.2 In MIC studio

To have the square's position feedbach, the simplest way is to sit in front of a computer, but in that case the interest is only limited, because the classical mouse is far much accurate. On the other hand, it could be useful in MIC studio, where of course no mouse can be used. The advantage of MIC studio is that you can see yourself moving on the large screen facing you. In order to display the square's feedback, we use the connecting shown in Fig 6.1.

Note: it is important that the video signal from the camera goes through the DVS before going into the SGI input, because otherwise the video signal and the computer are not synchronized, and the program crashes very soon.

6.1.3 How to use the programs

The SGI performs the treatment, that is gets the (x,y) position of the template and then, thanks to the "devctl" program directly controls the DVS by software. In that case, it also performs calculations on (x,y) and formats the datas so that the switcher understands them. The switcher will then display a black disk around the tracked position, so that the person can have the desired feedback.

To make the system work, the call is made by the command line :

tracking -arguments — conv -arguments — devctl

Tracking is the tracking program, which arguments are:

- -g Width x Height to make a cut of Width x Height in the normal size

- -z d/d where d/d is a fraction¡1, which sets the zoom ratio

Conv is the conversion program

- -g Width Height if you called tracking -g Width x Height
- -z d/d if you called tracking -z d/d

Devctl is the switcher control program

6.2 Database tool

6.2.1 Basic principle

As already underlined, the main fault of this method is that the tracking can most of the time be reliable for only a few seconds. After that, you begin to lose the original target little by little.

But this can be use to make a motion database. Suppose you have a movie recorded, on a VCR on my case, and you read this movie and input it in the computer. It requires somebody to click with the mouse on the point that should be tracked. Then the program prints in a file the coordinates. When the tracking is lost, the person stops the process. Then, re-playing the movie and the datafile synchronisely, that person checks until when the data can be kept. Then, correcting the problem by giving the good position at the frame just after the last valid frame and runing the program with this initialization, you get another set of valid data. Making this process 1 or 2 times, you can have data which are correct for a very long time.

The advantage is that you can specify any detail of the movie that you want to track. Though, this requires 2 mains features:

- in order to make edition of the data file, you need to adopt a structure where you know which data (x,y) corresponds to which frame. This can be easily done anyway.
- The principal hardware need is to be able to synchronize the movie and the program. The start of the movie must be commanded by software. Otherwise it is impossible to make the tracking begin at a specified frame.

6.2.2 How to use what was done

I didn't have a VCR that can be commanded this way. Nevertheless, it was possible to synchronize data and movie, but only once, handling like following :

- First you run the movie (push play on the VCR)
- Then run the tracking

Once the tracking is lost, you stop the process. Looking for a specific point of the movie (maximum or minimum of x or y component, easy to find if you use a graphic display of the data), you can synchronize the data. Once it is done, you read them synchronisely and discard what is bad information at the end. Then you have a movie and the data with it. Of course, to check that the synchronization is good, you must be able to re-play the movie with the data. This is possible thanks to dmplay, which enable us to play a movie recorded to the hard disk with the mpeg compression algorithm. I worked on dmplay to modify it so that we can now display the position of the template thanks to a data file. See the Annexes for use of the modified program.

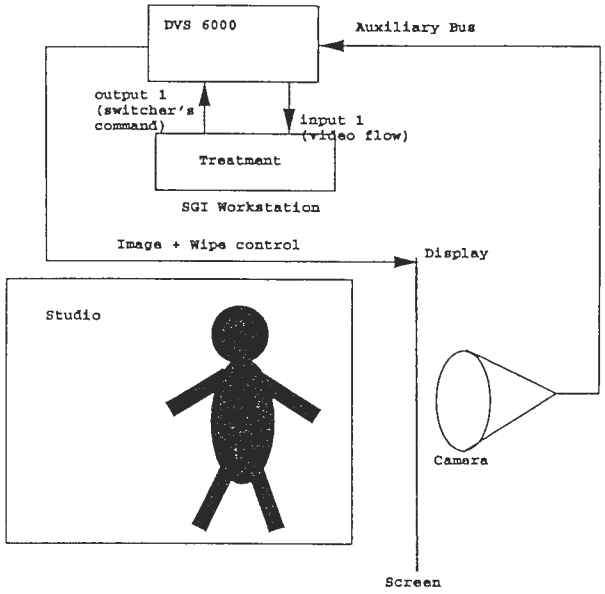

Figure 6.1: Connecting in the studio

Chapter 7

Conclusion

7.1 Conclusion about my results

As it appears in this presentation, I tried a lot of differents methods which I believed could lead to efficient motion tracking. Somehow, I am also aware that there are still remaining problems for each method.

However, I feel that an application could be made from what was currently done. The solely template matching method applied in the studio has got one only fault, which is to lose the tracking after some time. But linking this simple method with color information would definetly improve the system resistance very much. The already made color detection system (without color detection) can be used for it.

What I think about is for example hand tracking. First, you have to specify which area has to be tracked (mouse click). Then, template matching is running (current version). But you also make a local color detection around the template. If it happens that there is less than 50 percent of color (for example) in the area where skin is detected, then it is that the template is beginning to lose its initial target (problem of this method), and in that case you replace it were there is skin in the detected patch. I have the feeling that such a modification would give good results for hand tracking.

And then, you could link the resulting data of the tracking to some audio and video effects in the studio. For the future or such a future application, I hope my study here will be helpful to anybody who would like to do something about motion tracking in the studio.

7.2 An interesting overview over vision

ATR being a famous research institute, various talks and conferences were held here during my stay. I had the opportunity to attend some of them, and I am very glad of it. Because it gave me a very good overview of what people are able to do with image processing. I saw what is done, but also what remains to be done; which are the promising techniques and what their application could lead to. I really understand much more better how important is computer vision.

7.3 An interesting overview over research

Also my stay in ATR gave me the opportunity to see how a research work is performed.

This is something which was completely new to me, and the thing I had the most difficulties to get acquainted to. Being a student, I was not used to the process of creating a new application, and for me, good job was somehow synonym of quick job. In other words, I was not used to long term project. My research project here being for 7 months, I did not really know how to organize it to make it in 7 months.

Research process also means that you can be led to change slightly your goal or your methods when you get through unexpected difficulties. That's why I tried many different methods. Discovering advantages and drawbacks of each one, I could then see more clearly what use could be done of them to get to my goal. Understanding that took me some time however.

I also discovered, beside the research project activity, the importance of other points, such as publishing papers for the scientific community, explaining ones research activities in conferences and reading such papers as well as attending such conferences, because it can give you ideas about your own research project.

7.4 Final word

To finish this report, I would like to thank Inoue-san for having let me a lot of freedom about my research during my stay in ATR, and for having spent a lot of time with me in the studio to make things work. I do believe that ATR environment was a great one for a first job opportunity and I will remind my stay in ATR as a very important step in my life.

Appendix A

color detection

The following are in miris37/home/febvay/color

To deal with images (read from disk and write to disk for example), I used Parksan routines wich are available in img_utl.c and nr_utl.c. I included these files first. Consequently, almost all the time, you will need "img_utl.h" and "nr_utl.h" include files.

All the routines used to perform the differents steps are included in the file my_utl.c

- (r,g) color detection is performed by : essai.c

  Just run essai.c on a given image. To do that, change the ppm file name in the source and recompile following makefile.

- Hue-sat color detection is performed by : essai2.c

  Just run essai2.c on a given image. To do that, change the ppm file name in the source and recompile following makefile2.

A real-time (r,g) color detection is performed by second.c (based on contcapt.c Silicon Graphics source program (no OpneGL version). Second.c also performs template matching (no OpenGL version, old version which performs not very well). You can call color detection by "second -w"

If you call "second" alone, it is template matching. To recompile this program, you need the following :

img_utl.c, nr_utl.c, my_utl.c as well as new.c and second.c. New.c will be discussed in template matching section. To recompile, use makefile4. You require include file "mesfonction.h".

Note: every time, it is very important to have images read which initialise the histogram. These images depend on the person you wish to detect. In my case, it is always : calibre_visage.ppm (face skin patch) and calibre_main.ppm (hand skin patch). These files must be specified in the source programm, and placed in same directory where the program are run.

Once color detection is performed (you can also apply blur if you want, using the blur function in img_utl.c), you run test2.c, which finds the 3 biggest patches of skin in the image. Once again, the name of the image must be specified in the program. You can again recompile it using makefile3.

I put all these source programs in miris37/home/febvay/color. I also put calibre_visage.ppm and calibre_main.ppm files and executables versions of these programs discussed above as well.

Appendix B

template matching

The following (plus executables and examples) are in miris37/home/febvay/template

The basic algorithm is in bma.c

The final real-time version is in miris37/home/febvay/template/real-time/

- in new.c, there are the functions that make the template matching
- these functions are called by the program video.c

It also requires include file "mesfonctions.h" to work.

To re-compile the program, use makefile.

Note: the video.c program comes from a modification of contcapt.c, OpenGl version. It adds mouse to click on the point we wish to track at any moment. That's the main modification. Otherwise, after each captured frame, I added my routines contained in new.c

See the commented files for further information.

Appendix C

boundary extraction

The following (plus executables and examples) are in miris37/home/febvay/boundary

In contour.c, you will find the algorithm described before. It deals with "speed effect". All the routines are in that file. To use these functions, you should create a program like "test.c". To substract blue background in the studio, in did not find any simple solution. The program "essai.c" tries to do that, but it not very successful. If you run it on an image however, you can get rid of a good proportion of blue. It is not reliable enough though to be used in real time. I learned recently that DVS6000 could extract the blue background in the studio. Using this ability, it should be possible to make the boundary detection in real-time.

However, to compile "test.c" you should use "makefile" and to compile "essai.c", you should use "makefile2".

I made an attempt to run the boundary extraction real-time, but I had crashes due to bad blue color detection. However, I put it also in miris37/home/febvay/boundary in case somebody would like to have a look at it. To compile it, you should use "makefile3". It uses "rtime.c", which is just another modification of "contcapt.c" from SGI and "contour2.c", in which all the functions were written to work with real-time. But as I already said, some things may be changed as this was not tried with good blue color extraction.

Appendix D

dmplay modification

The dmplay program commented is placed in miris37/home/febvay/dmplay/ It is made of:

- alInit.c (not modified)
- gfx.c (modified)
- mvInit.c (not modified)
- dmICDecInit.c (not modified)
- mvInit.c (not modified)
- parseargs.c (modified : added -f command for synchronisation of data movie)
- setDefaults.c (not modified)
- vlInit.c (not modified)
- streamDecompress.c (modified)
- dmplay.c (modified : basicaly changed the name of the main() function to be able to give the control to my interface interface.c)

The files which were not modified are not included in the annexes.
There were two different version of dmplay available as SGI source program. However,

- the one under /usr/share/src/dmedia/dmplay/dmplay.cosmo
- the other under /usr/share/src/dmedia/dmplay/dmplay.dmic

The cosmo version, once recompiled is not able of using the ice jpeg decompression build-in engine of the O2. Then, the decompression is made by software and is very slow. Fortunately, the dmic version works fine on the O2. The source codes are differents in both directories.
To the files above, I added "interface.c", which makes an interface (based on Athena widget) so that you can :

- play the movie as many times as you want
- select a particular frame
- go to next/previous frame

Also, if you call dmplay by attaching to the movie a data file as follows:

dmplay -f FrameNbr moviefile < datafile

The program will run the movie and the datafile you have specified synchronisely, following the first data in datafile (which is the frame number of the movie which corresponds to the first data).

There are 2 examples of movie and data synchronized under the directory miris37/home/febvay/dmplay/examples/ Movie1 and data1 are synchronised; Movie2 and data2 are synchronised. The values for synchronisation were saved in "start" file. So, if you want to play Movie1 and data1, you should use the following command:

dmplay -f 24 mviel < data1

If you like to modify the program or recompile it, you should use Makefile.

Also, to display a graphic of the data file, you can use "plot" file by calling gnuplot : "gnuplot plot". In plot file, you should specify the 'data' you want to plot. It helps very much to have such a graphic visualisation of the data to be able to find a particular point for synchronization.