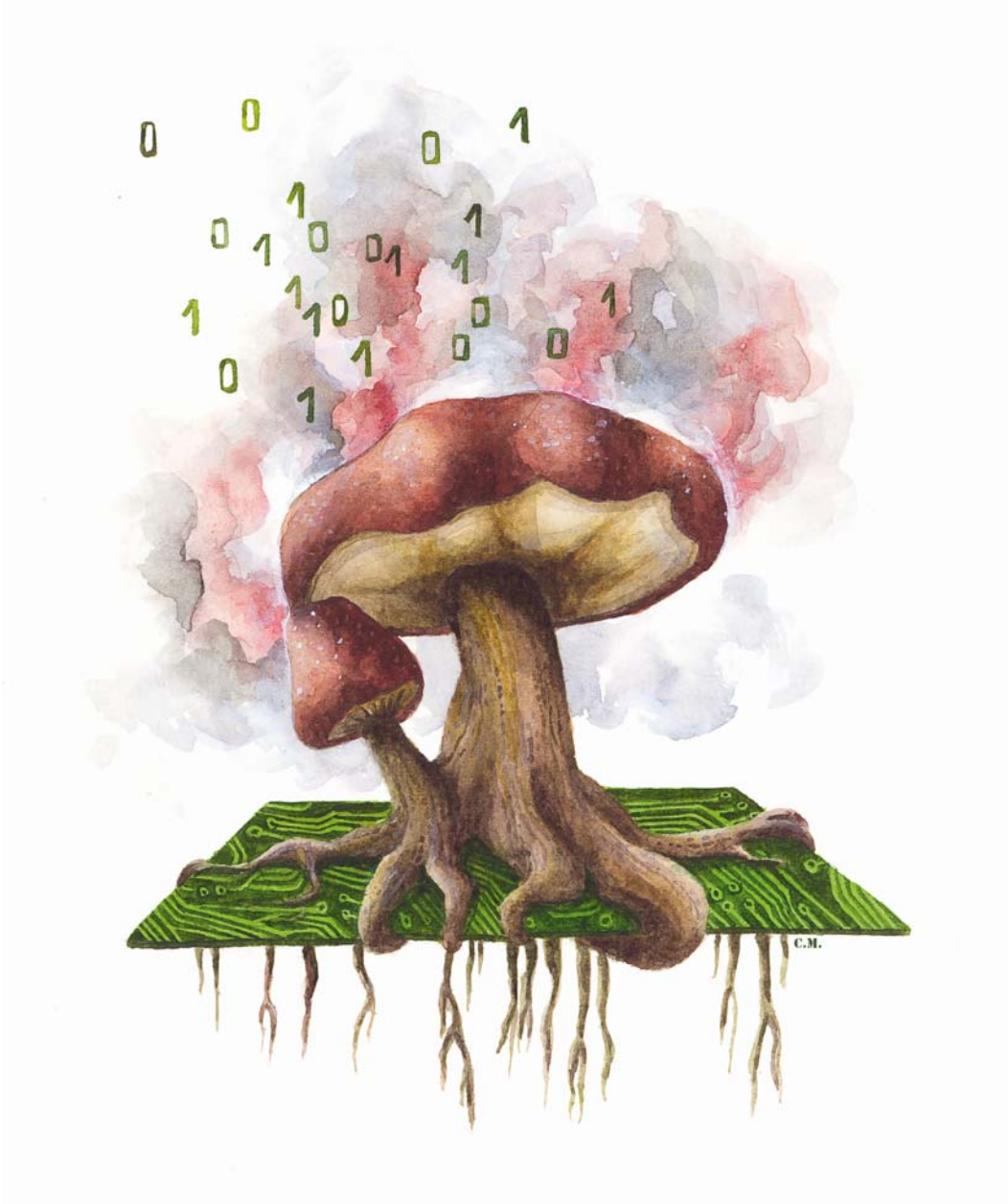
Mushroom Recognition

Master Semester Project, for Multimedia Signal Processing Group
February to June 2010

Under the supervision of Péter Vajda, and
Prof. Touradj Ebrahimi



EPFL ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE
MM SPG

Abstract

The purpose of this project is to implement a program that enables visual mushroom recognition on a mobile phone. It uses the possibilities of such a tool to take a picture and to send it to a server through internet, which will do the image recognition.

The algorithm will extract some of the already known features, and then find the best match in a provided database.

Some specific features are also proposed and tested, but not used for the final program. These features are aiming another kind of classification, using a biologist's knowledge about mushroom morphology.

In the first part of this project, a biologist's point of view is presented, which give essential information about mushroom in general. Then, an overview and a description of the program are done, and the results are presented. Finally, some important information are given for further work.

CONTENT

INTRODUCTION

Let's imagine you are walking in the woods, and see a mushroom on the ground. You would like to taste it, but have no idea about what kind of mushroom it could be. With a good digital camera and a recognition program, you could get some useful information, such as whether a mushroom is edible or not.

The aim of the project is to develop a mushroom recognition program based on specific characteristics extracted from photography. The main purpose of this program is to use mobile phone resources. Indeed, there are several advantages of combining a mobile phone with the mushroom recognition program. For instance, mobile phones are of everyday use and rather easy to understand for anyone, moreover they are powerful and rather simple to exploit. In addition, with the internet connectivity, one can benefit of an internet server which takes care of the computation part.

The state-of-the-art of mushroom recognition is nowadays only used for robot and automated sorting machine in food industry [1, 2]. However, the program developed in that case is specific for one task and is of no use in more general applications. Actually, these simple techniques focus only on a few features (like colour), and are not efficient in a more general purpose.

Consequently, more general image classification method are used, because it is a widespread topic, and there is a lot of well known features (such as colour histogram, sift, hog [3], shape descriptors [4], mser [5]). Thanks to the many studies, it is possible to find source code on internet. In order to focus on the main structure of the program, the mobile phone implementation, the database retrieving and specific feature creation, we will benefit of these source codes.

The final program should first provide a segmentation algorithm. In addition, like in most of the image recognition programs, a database of mushroom pictures has to be done, as well as a learning method to extract the features for the database, and a matching method to retrieve the best match from the database.

Several additional small programs have been implemented to gather information for results.

A BIOLOGIST'S POINT OF VIEW

In the following section, three persons are introduced. All of them provided important information, nevertheless, all of them told that it will not be an easy task. The image classification focuses only on visual components, therefore the outcomes deal only with the subject.

Mr. Jean-Yves Férréol

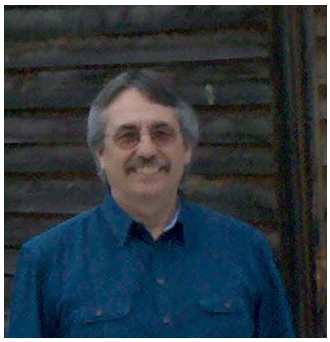


FIGURE 1 JEAN-YVES FÉRRÉOL

Jean-Yves Férréol is an expert of visual mushroom recognition, and a member of an association dedicated to the control of people's gathering, in order to avoid intoxication. On the contrary, biologists are not doing this kind of visual mushroom recognition anymore, because of its lack of efficiency. So for our application, his help and knowledge were essential.

He warned us against doing classification with only one image. In order to recognize the mushrooms, he usually holds it in his hands. He can see, smell and cut it if necessary. His judgement would not be reliable if he had only one image. In case he is not sure about a mushroom, he uses lens to analyse its structure. However, he usually does not deal with microscopy.

Then, he explained us what are the usual morphological relevant features that he controls. It is easy to find the same information in books about mushrooms and on internet as well.

However, the following point must be highlighted: the mushroom attributes (colour, cap ornament, etc...) is not always reliable information. Indeed, it can change, depending on the meteo and growth state. An amanita can lose his dots during strikes, and its colour can change from red to orange. The rain has also visual effects on several mushrooms. The shape also depends on the age or the room available. The smell can be relevant, but we have to pay attention to the objectiveness of this attribute.

In addition to those statements, he also said that we should never trust an image for mushroom recognition. But on a French internet forum [6], they try to find the name of the mushroom by watching an image, whereas it is not nonsense to do it. Then Jean-Yves redirected me to another person, Prof. Heinz Clémençon, a retired professor from the University of Lausanne.

Prof. Ian Sanders

FIGURE 2 PROF. IAN SANDERS

He is a Professor at the University of Lausanne, successor of Prof. Heinz Clémençon.

For the database, he explained that morphological analysis is anterior to image digitization. Consequently, it would not be easy to find a good one. As for the recognition, a trained brain is far more efficient than informatics program; this is why some people are able to do it. Nowadays, biologists use DNA for a reliable recognition. He also told to speak with Heinz Clémençon.

Prof. Heinz Clémençon

FIGURE 3 PROF. HEINZ CLÉMENÇON

He sent a mail, in which he says that if the project has mycological purpose, it will be a failure for many reasons. First of all, if it is for image analysis purpose, he advised to restrict the number of objects to recognize. According to macroscopic attributes, the objects to classify should be easily differentiable. He also helped to contact someone for the database.

Secondly, the mycological purpose will fail because the macroscopic classification of mushroom is old-fashioned. Nowadays, techniques such as photonic microscopy and wall chemistry of spores and cytoplasm are used. Indeed if we can easily recognize some species (such as amanita), those are exceptions. He added that it's amateurship to think that we can do recognition with a photo. Although considering good quality pictures, human will in most of the case be wrong.

For image analysis purpose, he advised to choose one species in a specific region which is easy to recognize visually such as Amanita from the North of the Alpes.

Books, internet, database

As regards for the program, the idea is to find a mushroom tree, on which we can find the mushroom name according to some biologic criteria such as:

For the **cap**: shape, surface, structure, colour, ornament and border shape
For the **gills**: attachment, spacing, size, colour, structure and disposition
For the **stalk:** shape, root, surface and colour above/below ring, ornament (valve, ring), colour, structure
And some **others**: growth surface (habitat), population, smelling, spore-print colour, ring number and ring type

There is a database of these criteria available on internet (Schlimmer's mushroom database [7]), but it's only to investigate the edibility component by data mining, and don't contain any images nor mushroom names.

The climate, the chemical nature of ground, physical nature and also biotic factors can influence the state of a mushroom. For clarity, the list of criteria is not exhaustive. There are so many elements to take care, and for each type of mushroom, there are different attributes to analyze [8].

The actual set of images is coming from Miss. Béatrice Senn, which after several mails sent pictures of mushrooms from the region. The pictures have been taken by the mushroom amateur Max Danz (from Attiswil, Solothurn, Switzerland). Due to the small database, there are only two images (sometimes three) for each mushroom type. It is not really a good set to train the program. On internet, we can find a lot of photography, but they have to be collected one by one, and it may not be reliable for the mushroom names.

FIGURE 4 DATABASE SAMPLES

PROGRAM

The program is divided in several parts and takes advantage of the OpenCV c++ library. On one side we have the server and mobile client, and on the other side the segmentation algorithm and the database learning and matching.

When the client sends the correspondent command, the server will call the segmentation program or the matching program. A small program overview is annexed.

Segmentation

When the client on the phone is started, you can take a picture, and then select the foreground and the background for the segmentation. This will create a mask. Then you send it to the server, which will do the segmentation and return the segmented image. If the image is well segmented, it will afterwards run the matching sequence on the server, and send you back the information about the best match.

The segmentation algorithm is called GraphCut and is based on Graph Theory. It is grouping the pixel image in two: the object, and the background using the min-cuts/max-flow algorithm. This

method was chosen to let the user participate, and use the phone touch screen capability. The GraphCut technique lets you put a hard constraint for the segmentation, and is relatively efficient.

The min-cut/max-flow algorithm comes from the duality theorem that states, if max-cut has a solution, min-cut has the same solution. The implementation of the algorithm has been done in c++ by Yuri Boykov and Vladimir Kolmogorov [9, 10, 11]. The segmentation structure comes from a matlab toolbox implemented by Mohit Gupta and Krishnan Ramnath from the Robotics Institute in Pittsburgh.

Database learning

This is a simple standalone program that is called manually. It has to be ran every time the database changes. This will simply extract the features for each image contained in the database folder.

Image matching

It will take the segmented image, and extract the features from it. Then it will compute the Euclidian distance between this image feature and features from the images in the database. The SIFT feature doesn't use the Euclidian distance for matching, but it counts how many point of interest are the same between two images. Afterward, it is possible to define the image from the database which has the smaller Euclidian distance to be the best match (or the bigger number of corresponding points for the SIFT).

The last thing to do is to get the real name from the best match in the database. In consequence, a file has to be prepared, in which the database pictures are linked to a mushroom name.

Additional programs

There are also useful standalone programs implemented: one to test a feature, a second to segment one image after another to prepare the database and a third to save each Euclidian distance in a file. This last program should have helped for a potential mixture feature.

To test a feature, the idea is to take one by one the images of mushroom, and to compare it with the database. Of course the best match will be the same image, but only the second best match is considered. As we have at least two elements for one object, it should allow counting how many times the first and second best match are from the same mushroom. For the blob and the ring detection, another procedure is used. A file stores the true information for each object to recognize, and then we compare the feature result with this truth to get the score.

Android

The mobile client is coded for the android operating system, in order to work on a lot of devices. This implies to use java programming language. The structure is a basic client-server communication through internet. The only tricky part is to create the mask.

FEATURES

Here is presented all the features implemented and used in the main program. In order to do a good classification, we can use three types of data: the common features, the specific features and the contextual features.

COMMON FEATURES

1. Color histogram (LaB) [12, 13]

This is a basic and simple feature that group pixels according to the colour into bins. For our purpose, the Lab colour space has been used, but several other colour space exists. This choice was motivated by its efficiency, as it is representative of the human visual system.

2. Sift [14]

This scale-invariant feature uses a stage filtering approach. The aim of this is looking for key location by finding extrema of difference-of-Gaussian function, the only smoothing kernel for scale space analysis. This technique that consists of finding point of interest is efficient for linear transform invariance. From these points, some features (called SIFT keys) are extracted over local area. They are considering gradient magnitude and orientation (this is to reach rotation invariance), and also orientation histogram.

By counting the number of same point of interest (by identificating similar keys) corresponding between two images, it is possible to characterize the amount of likeliness.

3. Gabor Wavelet [15, 16, 17]

The code used for the Gabor Wavelet transform comes from the FELib [18]. The interest for this feature is that values are representative of the texture (repetitive patterns) of the image. As mushrooms are quite textured, it can be a good candidate.

4. Shape Feature [19]

The shape is also a good candidate for our application.

4.1 Moments [20]

First we just have to extract the contour, in order to compute statistical moments. To improve the feature, we can use the HU moments which are invariant to rotation, scale and translation.

4.2 Fourier Descriptor [21]

This is another basic shape feature. It consists of the creation, with the contour coordinates, of a one dimension complex signal.

$$(x, y) = x + j * y$$

Then, we compute the discrete Fourier transform, and keep only a certain number of low frequency components, as it is the main component in the shape.

SPECIFIC FEATURES

5. Ring detection

The idea is to do a one-way edge detection on the mushroom's stalk, to project the white pixels over the y axis, and to process this result. Furthermore, we need to rotate the stalk by using the covariance matrix to optimize the projection.

To detect a ring, we first separate the stalk from the cap, using the same segmentation technique as explained before. Then, we do the one-way edge detection (using one Sobel kernel). To get the angle for the rotation, we have to compute the arctan of the eigenvector of the covariance matrix which has the bigger eigenvalue [21]. After projecting the white pixels over the y axis, the output vector is normalized. Now, all values smaller than a specified threshold is set to zero. The last thing to do is to remove the values above the threshold if they are near the end or the beginning of the stalk.

If there is a value different of zero in the final vector, we consider there is a ring.

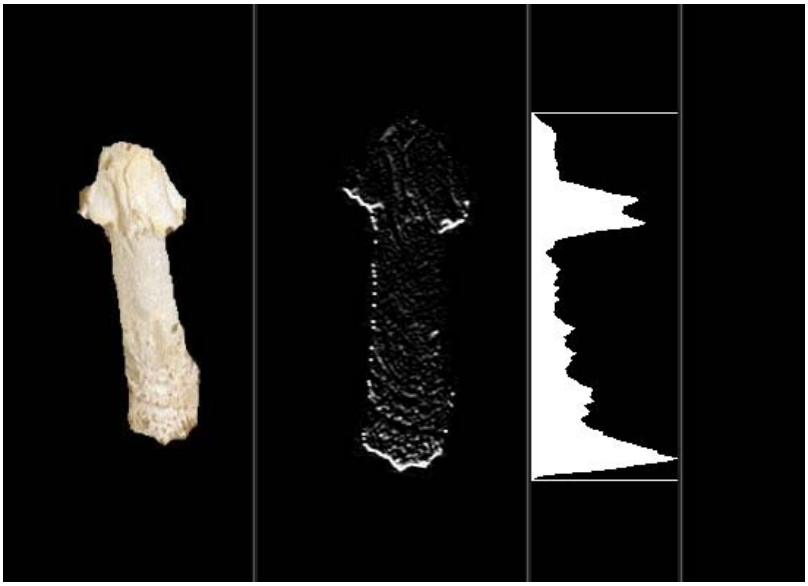


FIGURE 5 RING DETECTION

7. White Blob detection

This feature is based on morphological analysis, so we have to threshold the image in order to make it binary. For this application, we want to detect white dots, so the threshold is set to two hundred. Now we simply want to keep only the small white dots, and to remove the big white elements.

In order to make the dots disappear, we open the image with a big structuring element. We can now subtract this image to the original thresholded image. Thus we keep only the dots. After the opening, the image is quite deformed, so we have to restore it before subtraction.

We consider that the mushroom has dots only if the total sum of the white pixels is above a defined limit. This limit has been set to one thousand.

FIGURE 6 BLOB DETECTION

RESULTS

Each feature has been tested over a dataset of one hundred and four segmented images. After using the program to test each features, we can display the efficiency of the features.

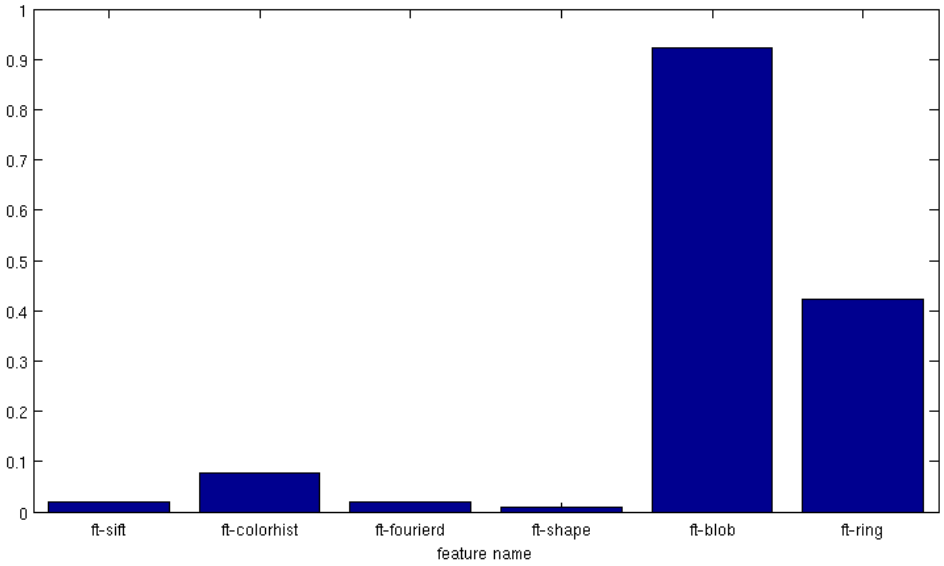


FIGURE 7 RESULTS

The common features, which are efficient for general image recognition, obviously perform very badly. The colour histogram has apparently the best score, and this means that the colour is an important component for the mushroom recognition. For every other feature, we only use black and white images, and considering only black and white images from different mushrooms, it's far more difficult to distinguish them.

Demo

The demo consists of the client android program running on the mobile phone, and the server, running on the computer. We have to take care of putting the good IP address and to connect both mobile phone and server to internet. We can first run the client, take the picture, make the selection of the background and foreground, and send it. After a while, we can see the segmented picture showed on the phone. If we are satisfied with the segmentation, we can click a button to begin the matching. Then, it will show us the name of the corresponding mushroom according to each feature.

FIGURE 8 DEMO ANDROID CLIENT

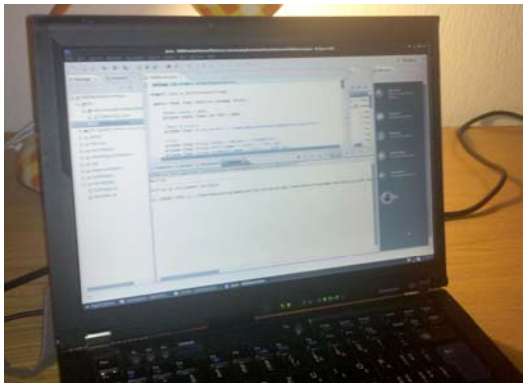


FIGURE 9 DEMO SERVER

CONCLUSION

Personal conclusion

This has been a hard project which brought me a lot of knowledge. Now, I would have done things differently, as I am going to explain below in further work. As it was already said, we cannot use the common features as they are really not efficient. A better approach is to implement several specific features. Although using different specific features could be the solution, the results are surprisingly poor.

The database used for this project is not exhaustive and the results could be really improved

Further work

To improve this program, several changes have to be made. The first one refers to the database: we must find a better one, with higher resolution, and better mushroom position. Indeed, sometimes, the position implies occlusion, or some attributes are not visible and potentially not recognizable. Furthermore we should do more specific features, in order to do a more efficient discriminating classification based on biologic tree. We can also use more advanced matching techniques such as in machine learning to achieve good results.

An important improvement could be to use the contextual information, as we have seen the influence of the external local factors on mushroom determination. We can get them by the mobile phone sensors (GPS, time, camera focal data), and even asking the user, like for the mushroom smell, to finalize the matching. This is supposing we have to prepare with attention the database, which should not anymore contain only images. Some useful contextual information would be (if possibly retrievable) the weather forecast, on which ground it has grown, the age, etc...

We can deduce from all this work that this application, if it worked well, can't be used as a real reliable tool. It can only help to begin in this domain, for the interested people, but we should not trust it if the life of someone is at risk.

Acknowledgment

I want to thank all the people indirectly implied in this project, such as Max Danz, Béatrice Senn, the Biologists, and the people who did some code. I wish to thank specially my assistant Péter Vajda, who helped so much for the mobile phone and server program, Veronica Sergi, who had a lot to do for the fast English correction of the report, and my sister Capucine Matti for the beautiful title illustration.

REFERENCES

[1] R.D. Tillett and B.G. Batchelor, "An algorithm for locating mushrooms in a growing bed", Computer and Electronics in Agriculture, 6 (1991) 191-200, Elsevier Science Publishers B.B, Amsterdam

[2] Raji A. O. and A. O. Alamutu "Prospects of Computer Vision Automated Sorting Systems in Agricultural Process Operations in Nigeria", Department of Agricultural Engineering, University of Ibadan, Nigeria

[3] Navneet Dalal and Bill Triggs, "Histograms of Oriented Gradients for Human Detection", INRIA Rhône-Alpes,655 avenue de l'Europe, Montbonnot, France

[4] Yang Mingqiang, Kpalma Kidiyo and Ronsin Joseph, "A Survey of Shape Feature Extraction Techniques", in Pattern Recognition, Peng-Yeng Yin (Ed.)(2008) 43-90

[5] Per-Erik Forssén and David G. Lowe, "Shape Descriptors for Maximally Stable Extremal Regions", Department of Computer Science, University of British Columbia

[6] http://www.champis.net

[7] http://archive.ics.uci.edu/ml/datasets/Mushroom

[8] Roger Heim, "Champignons d'Europe", editions N. Boubée et Cie, Place Saint-André-des-Arts 3, Paris, France

[9] Yuri Boykov and Vladimir Kolmogorov, "An Experimental Comparison of Min-Cut/max-Flow Algorithms for Energy Minimization in Vision", in IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI), September 2004

[10] Yuri Y. Boykov and Marie-Pierre Jolly, "Interactive Graph Cuts for Optimal Boundary & Region Segmentation of Objects in N-D Images", Siemens Corporate Research, 755 College Road East, Princeton, USA

[11] Vladimir Kolmogorov and Ramin Zabih, "What Energy Functions Can Be Minimized via Graph Cuts?", IEEE Transactions on pattern analysis and machine intelligence, vol. 26, no. 2, February 2004

[12] Michael J. Swain and Dana H. Ballard, "Color Indexing", Iternational Journal of Computer Vision, 7:1, 11-32 (1991), Kluwer Academic Publishers, Netherlands

[13] Gernot Hoffmann, "CIELab Color Space", September 15, 2009

[14] David G. Lowe, "Object Recognition from Local Scale-Invariant Features", Computer Science Department, University of British Columbia, Vancouver, Canada, 1999

[15] Jianke Zhu, Mang I Vai and Peng Un Mak, "A New Enhenced Nearest Feature Space (ENFS) Classifier for Gabor Wavelets Features-Based Face Recognition", University of Macau, Macau, China

[16] P. Kruizinga, N. Petkov and S.E. Grigorescu, "Comparison of texture features based on Gabor Filters", Institute of Mathematics and Computing Science, University of Groningen, Netherlands

[17] Anil K. Jain and Farshid Farrokhnia, "Unsupervised Texture Segmentation Using Gabor Filters", Michigan State University, East Lansing

[18] http://appsrv.cse.cuhk.edu.hk/~jkzhu/felib.html

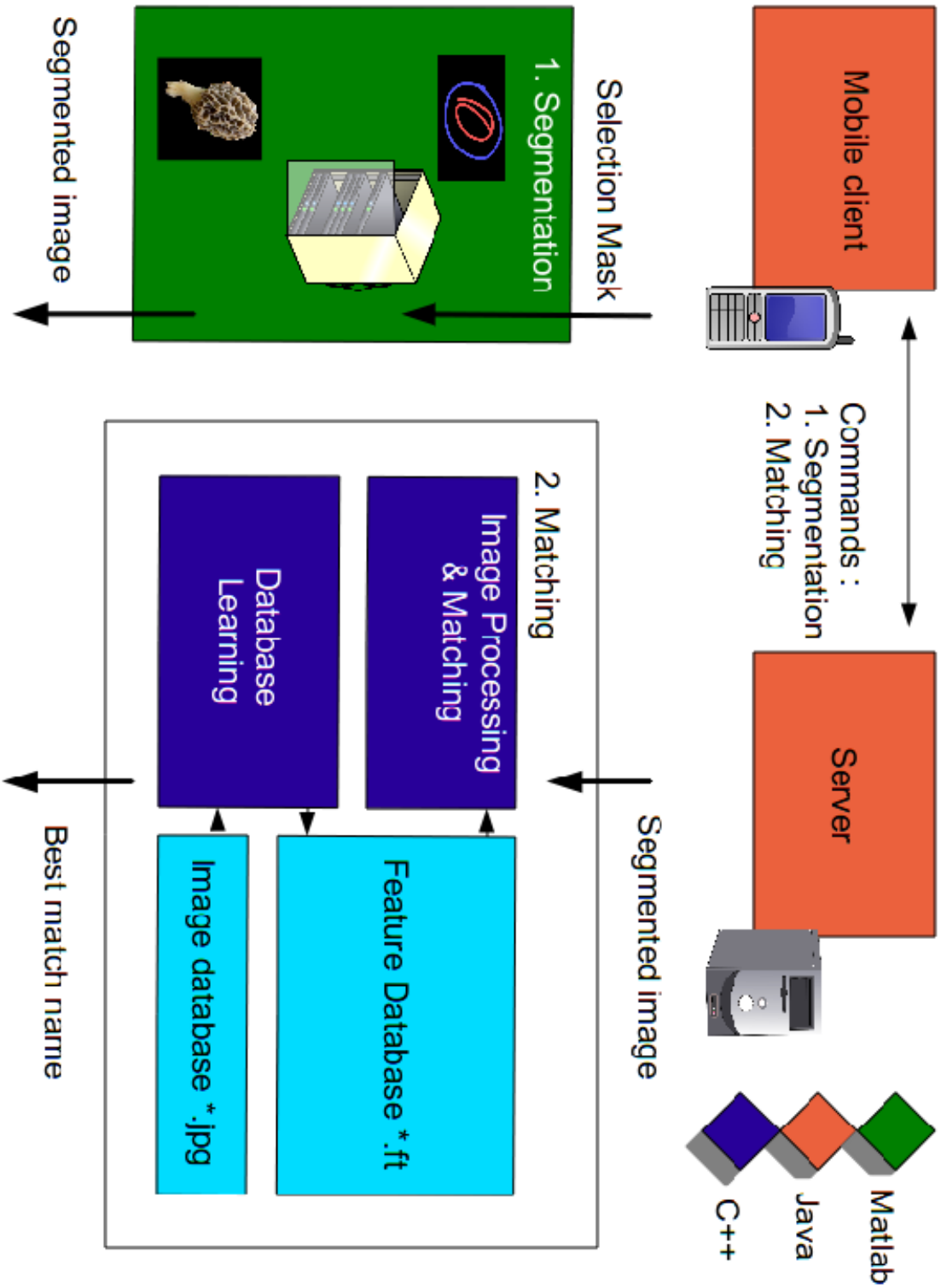
[19] Remco C. Veltkamp and Michiel Hagedoorn, "State-of-the-art in Shape Matching", Utrecht University, Department of Computing Science, Netherlands

[20] M. K. Hu, "Visual Pattern Recognition by Moment Invariants", IRE Trans. Info. Theory, vol. IT-8, pp.179-187, 1962

[21] Prof. Jean-Philippe Thiran, "Image analysis and pattern recognition", Master lecture at EPFL

ANNEXES

1. Program overview



2. Source code of blob feature

```
int main(int argc, char** argv)
{
	//==============
	//Read the image
	//==============

	IplImage* src_image = cvLoadImage( argv[1] ,0);
	if(src_image==0){printf("Image was not loaded.\n");}
	CvSize src_size=cvGetSize(src_image);

	IplImage* thres_image = cvCloneImage(src_image);
	IplImage* temp_image = cvCloneImage(src_image);
	IplImage* morph_image = cvCloneImage(src_image);
	IplImage* rest_image = cvCloneImage(src_image);
	IplImage* out_image = 0;


	//============
	//Thresholding
	//============

	cvThreshold(src_image,thres_image,200,255,CV_THRESH_BINARY);

	//=====================
	//Morphology
	//=====================

	//CLOSING
	IplConvKernel* strel_close = 0;
	strel_close = cvCreateStructuringElementEx(3,3,1,1,CV_SHAPE_ELLIPSE);//ELLIPSE,RECT,CROSS
	cvMorphologyEx(thres_image, morph_image, temp_image,strel_close,CV_MOP_CLOSE,3);

	//OPEN
	IplConvKernel* strel_open = 0;
	strel_open = cvCreateStructuringElementEx(3,3,1,1,CV_SHAPE_ELLIPSE);//ELLIPSE,RECT,CROSS
	cvMorphologyEx(morph_image, morph_image, temp_image,strel_open,CV_MOP_OPEN,8);


	//Prepare for restauration
	IplConvKernel* strel_dilate = 0;
	strel_dilate = cvCreateStructuringElementEx(5,5,1,1,CV_SHAPE_ELLIPSE);//ELLIPSE,RECT,CROSS


	//restauration : DO with rectangular structuring element, it's better
	CvScalar morph_pix;
	CvScalar thres_pix;
	CvScalar white;white.val[0]=255;
	cvZero(rest_image);
	for(int i=0;i<9;i++) {
		cvDilate(morph_image,morph_image,strel_dilate,1);
		for (int i = 0; i < src_size.height ; i++) {
		for (int j = 0; j < src_size.width ; j++) {
			morph_pix = cvGet2D(morph_image,i,j);
			thres_pix = cvGet2D(thres_image,i,j);
			if (morph_pix.val[0] == 255 && thres_pix.val[0] == 255){
				cvSet2D(rest_image,i,j,white);
			}
		}
	}
	}

	//Removing big parts
	CvScalar pix;
	CvScalar black;black.val[0]=0;
	out_image=cvCloneImage(thres_image);
	for (int i = 0; i < src_size.height ; i++) {
		for (int j = 0; j < src_size.width ; j++) {
			pix = cvGet2D(rest_image,i,j);
			if (pix.val[0] == 255){
				cvSet2D(out_image,i,j,black);
			}
		}
	}
```

```
	//OPENING the output
	cvMorphologyEx(out_image, out_image, temp_image,strel_open,CV_MOP_OPEN,3);

/*
	IplConvKernel* strel_erode = 0;
	strel_erode =
cvCreateStructuringElementEx(5,5,1,1,CV_SHAPE_ELLIPSE);//ELLIPSE,RECT,CROSS
	cvErode(out_image,out_image,strel_erode,1);
*/

	//==================
	//Count white pixels
	//==================

	//Count
	int result=0;
	for (int i = 0; i < src_size.height ; i++) {
		for (int j = 0; j < src_size.width ; j++) {
			pix = cvGet2D(out_image,i,j);
			if (pix.val[0] == 255){
				result = result+1;
			}
		}
	}

	//cout << "result : " << result << endl;
	if (result > 1000){result = 1;}else{result=0;}


	//=====================
	//Display
	//=====================

/*
	cvNamedWindow( "Picture" , CV_WINDOW_AUTOSIZE );
    cvShowImage( "Picture", thres_image);

    cvNamedWindow( "Picture2" , CV_WINDOW_AUTOSIZE );
    cvShowImage( "Picture2", out_image );

    cvNamedWindow( "Picture3" , CV_WINDOW_AUTOSIZE );
    cvShowImage( "Picture3", rest_image );

    cvWaitKey(0);

    cvDestroyWindow("Picture");
    cvDestroyWindow("Picture2");
    cvDestroyWindow("Picture3");
*/

	//========
	//Cleaning
	//========

	cvReleaseImage(&src_image);
	cvReleaseImage(&thres_image);
	cvReleaseImage(&temp_image);
	cvReleaseImage(&morph_image);
	cvReleaseImage(&rest_image);
	cvReleaseImage(&out_image);

	//=================
	//Save it to a file
	//=================

	ofstream feature_output(argv[2]);
	feature_output << result << " ";
	if(feature_output.fail()){printf("error saving feature to file .\n");}
	feature_output.close();

}
```

3. Source code of ring feature

```
int main(int argc, char** argv)
{

        //=================================
        //Read the image and initialization
        //=================================

        IplImage* src_image = cvLoadImage( argv[1] ,0);
        if(src_image==0){printf("Image was not loaded.\n");}
        CvSize src_size=cvGetSize(src_image);

        IplImage* fil_image = cvCloneImage(src_image);
        CvScalar white;white.val[0]=255;

        //=================================
        //Edge detection for one direction
        //=================================

        //Create edge detection kernels
        float valsdown[] = {-1,-2,-1,0,0,0,1,2,1}; // Sobel kernel
        CvMat* kerneldown = new CvMat;
        cvInitMatHeader(kerneldown,3,3,CV_32FC1,valsdown);

        float valsup[] = {1,2,1,0,0,0,-1,-2,-1};
        CvMat* kernelup = new CvMat;
        cvInitMatHeader(kernelup,3,3,CV_32FC1,valsup);

        //Filtering with the kernels
        //cvFilter2D(src_image,fil_image,kerneldown);
        cvFilter2D(fil_image,fil_image,kernelup);

        //==================================
        //Inertial axis and angle computation
        //==================================

        //Covariance matrix based on the centered moments
        CvMoments* moments = new CvMoments;
        cvMoments( fil_image, moments );

        float valscov[] = {moments->mu20,moments->mu11,moments->mu11,moments->mu02};
        CvMat* cov_mat = new CvMat;
        cvInitMatHeader(cov_mat,2,2,CV_32FC1,valscov);

        //Center of gravity
        float valsgrav[] = {moments->m10/moments->m00,moments->m01/moments->m00};
        CvMat* center_grav = new CvMat;
        cvInitMatHeader(center_grav,1,2,CV_32FC1,valsgrav);

        //Compute and store the eigenvalues and eigenvectors of covariance matrix
        CvMat* e_vect = cvCreateMat( 2 , 2 , CV_32FC1 );
        CvMat* e_vals = cvCreateMat( 1 , 2 , CV_32FC1 );
        cvEigenVV(cov_mat, e_vect, e_vals );

        //Some output
        //cout << moments->m10/moments->m00 << " " << moments->m01/moments->m00 << endl;
        //cout << cvGetReal1D(center_grav,0) << " " << cvGetReal1D(center_grav,1) << endl;
        //cout << 100*cvGetReal2D(e_vect,0,0) << " " << 100*cvGetReal2D(e_vect,0,1) << endl;
        //cout << cvGetReal2D(e_vect,1,0) << " " << cvGetReal2D(e_vect,1,1) << endl;
        //cout << cvGetReal1D(e_vals,0) << " " << cvGetReal1D(e_vals,1) << endl;

        //Compute 2 lines for each inertial axis
        CvPoint point1;point1.x = cvGetReal1D(center_grav,0);point1.y =
cvGetReal1D(center_grav,1);
        CvPoint point2;
        point2.x = 100*cvGetReal2D(e_vect,0,0) + cvGetReal1D(center_grav,0);
        point2.y = 100*cvGetReal2D(e_vect,0,1) + cvGetReal1D(center_grav,1);
        //cvLine(fil_image,point1,point2,white,8);

        point2.x = 100*cvGetReal2D(e_vect,1,0) + cvGetReal1D(center_grav,0);
        point2.y = 100*cvGetReal2D(e_vect,1,1) + cvGetReal1D(center_grav,1);
        //cvLine(bw_image,point1,point2,white,8);
```

```
        //Compute the angle for the two inertial axis
        int e_vect_select=1;
        if (cvGetReal1D(e_vals,0) > cvGetReal1D(e_vals,1)){e_vect_select = 0;}
        double angle = (-1) *
cvFastArctan(cvGetReal2D(e_vect,e_vect_select,0),cvGetReal2D(e_vect,e_vect_select,1));

        //==================================
        //Rotation
        //==================================

        //Rotation initialisation
        IplImage* rot_image = cvCloneImage(fil_image);
        CvPoint center;
        center.x=src_image->width*0.5f;
        center.y=src_image->height*0.5f;
        double scale = 1.0;
        CvMat* rot_kernel = new CvMat;
        float valsrot[] = {0,0,0,0,0,0};
        cvInitMatHeader(rot_kernel,2,3,CV_32FC1,valsrot,CV_AUTOSTEP);

        //We do the rotation kernel
        cv2DRotationMatrix(cvPointTo32f(center),angle,scale,rot_kernel);
        //We do the rotation
        cvWarpAffine(fil_image,rot_image,rot_kernel);
        //Or cvTransform( src_image, dst_image, &MapMatrix, NULL );
        //Or cvGetQuadrangleSubPix(const CvArr* src, CvArr* dst, const CvMat* mapMatrix);
        //for subpix accuracy

        //================================
        //Pixel counting - create a vector
        //================================

    //We create a vector to count the white pixels on each row
    int num_bins = 11;
    vector<int> pixel;
    int temp;
    CvScalar pix;
    for (int x=0;x<src_size.height;x++){
                temp = 0;

                for (int i=-(num_bins-1)/2;i<=(num_bins-1)/2;i++){
                        if (x+i < src_size.height && x+i > 0){
                                for (int y=0;y<src_size.width;y++) {
                                        pix = cvGet2D(rot_image,x+i,y);
                                        temp = temp + pix.val[0];
                                }
                        }

                }
                pixel.push_back(temp);
        }


        //================================
        //Vecor processing and normalization
        //================================

        //NORMALIZATION

        // store max value
        temp = 0;
        for (int x=0;x<src_size.height;x++){
                if (pixel[x] > temp) {
                        temp =pixel[x];
                }
        }

        //Scaling the x axis
        int max=temp;
        int factor=max/100;

        vector<int> norm_pixel;
        for (int x = 0 ; x<(int)pixel.size() ; x++){
                norm_pixel.push_back(round(pixel[x]/factor));
        }
```

```
//PROCESSING

//length of the feet
temp=0;
while (norm_pixel[temp]==0){
	temp++;
}
int a = temp; // beginning of non-zero values

temp=norm_pixel.size()-1;
while (norm_pixel[temp]==0){
	temp--;
}
int b = temp; // end of non-zero values

// keep only values above 95
int info[temp];
int size_temp = temp;
temp=0;
for (int x =0; x< (int)norm_pixel.size();x++){
	if (norm_pixel[x]>95){
		info[temp]=x;
		temp++;
	}
}


int tp1,tp2,tp3=0;//,tp4=0;
int info2[size_temp];

// Control at the beginning if there is values inside the forbidden zone to remove
for (int x = 0 ; x < size_temp; x++){
	info2[x]=info[x];
	tp2 = x-1;
	if(tp2<0){tp2=0;}
	if (info[x] <= a+10 || tp1==1) {
		tp1 = 1;
		info2[x]=0;
	}
	if (info[x]>a+10 && (info[x] - info[tp2] ) > 5) {
		tp1=0;
	}
}

// Control at the end if there is values inside the forbidden zone to remove
for (int x = size_temp ; x > 0; x--){
	tp3 = x+1;
	if(tp3>size_temp){tp3=size_temp;}
	if (info[x] >= b-20 || tp1==1) {
		tp1 = 1;
		info2[x]=0;
	}
	if (info[x] < b-20 && (info[tp3] - info[x] ) > 5) {
		tp1=0;
	}
}

//=================
//Display the vector
//=================

//To show a line at the beginning and the end of non-zero values
norm_pixel[a] = 100;
norm_pixel[b] = 100;

//Creating the size of the "hist"
CvSize new_size;
new_size.height = src_size.height;
new_size.width = round(max/factor);

//Creating the "hist" image
IplImage* hist = cvCreateImage(new_size,8,1);
cvZero(hist);

//Setting the pixels according to the vector content
for (int x=0;x<(int)norm_pixel.size();x++){
	for (int i=0;i<norm_pixel[x];i++){
		cvSet2D(hist,x,i,white);
	}
}
```

```
    //=================
    //Displaying
    //=================
/*
    cvNamedWindow( "Picture" , CV_WINDOW_AUTOSIZE );
    cvShowImage( "Picture", rot_image);//rot_image);

    cvNamedWindow( "Picture2" , CV_WINDOW_AUTOSIZE );
    cvShowImage( "Picture2", hist );

    cvWaitKey(0);
    //cvDestroyWindow("kernel");
    cvDestroyWindow("Picture");
    cvDestroyWindow("Picture2");
*/

        //==================
        //Compute the output
        //==================

        int output=0;
        for (int i=1;i<size_temp;i++){
               if (info2[i]>0){output=1;break;}
        }

        //========
        //Clearing
        //========

        cvReleaseImage(&src_image);
        cvReleaseImage(&fil_image);

        //=================
        //Save it to a file
        //=================

        ofstream feature_output(argv[2]);
        feature_output << output << " ";
        if(feature_output.fail()){printf("error saving feature to file .\n");}
        feature_output.close();

}
```