ENSICAEN
ÉCOLE NATIONALE SUPÉRIEURE D'INGÉNIEURS DE CAEN
& CENTRE DE RECHERCHE

Technical report pattern recognition system

Breton-Belz Emmanuel - 2nd Year Internship

Une grande école pour réussir

ENSICAEN
6, boulevard Maréchal Juin
CS 45 053 – F- 14050 Caen Cedex 4
Tél. +33 (0)2 31 45 27 50
Fax +33 (0)2 31 45 27 60

Table of contents

Chapter 1

Introduction

This application is produced by the University of Salerno. It has been made during an internship and is part of a global pattern recognition application. The final goal is to extract the documentation of materials in live from videos captured by a camera headset. The subject of this report concerns the extraction of multiple pattern in a single image.

In this document you will find a description of why the project have been created. Then a presentation of the components used like OpenCV features. And finally the application itself and how it detects patterns.

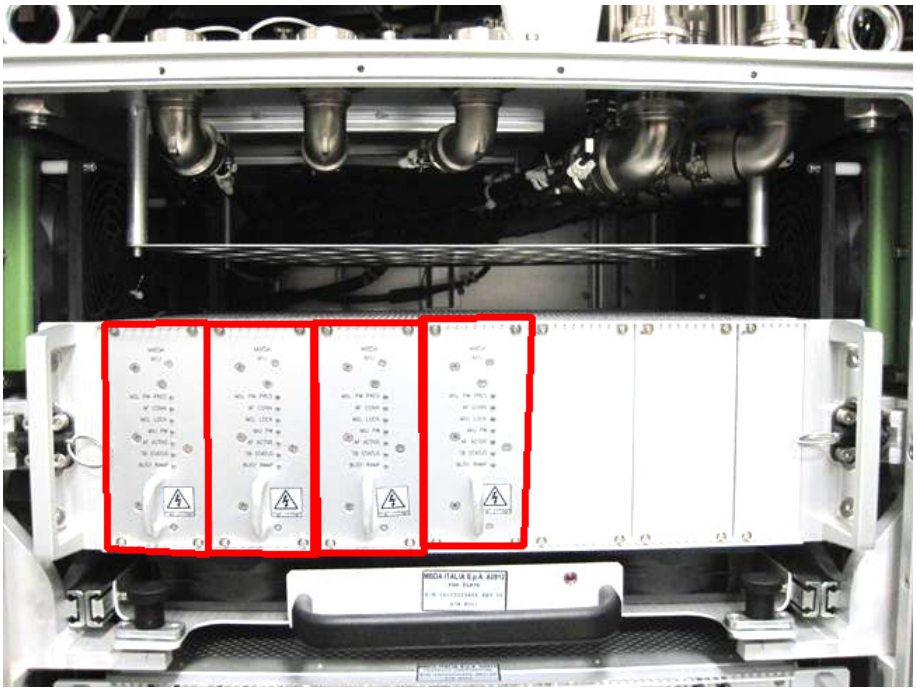

Figure 1.1: Application finding 4 patterns from a single training pattern.

Chapter 2

Need

At work, when people have to make maintenance of the material, they encounter a problem with the density of the maintenance manuals which can make around 700 pages. We try to ease the maintenance by recreating manuals that focuses on the material that the technician is looking at. To obtain this result, we have to analyze the images of the camera and extract the type of material. That is what this report deals with.

This solution can apply to a lot of other objects to find monuments and extract their description.

Figure 2.1: Photos of the type of helmet which the application could be used with

Chapter 3

Solution

3.1 Introduction

To resolve the problem, I only worked on Linux LMDE. It should work on other unix based operating system. For windows it requires Qt3 to open the windows that show the images, otherwise it should work if OpenCV 2.4 is installed. First, I will introduce OpenCV because all the application is based on it. I will talk more precisely about the key points and the descriptors that are computed from them. I will also explain the homography system. All of this component is embedded in the OpenCV library. Then I will explain how I used them in the explication, how I tested it and what I added to obtain the information about the patterns.

3.2 Composants used

3.2.1 OpenCV

OpenCV is an image analysis and synthesis library that brings all the necessary for video and photo analysis. Introduced in 2000, it is now used a lot in the fields that require image analysis. It is built around a modular system that allows users to install the library depending on their needs. Except the non free module, the library is licensed BSD which means that we can reuse and modify all the components freely. In this document I will talk more about precise modules of OpenCV like base and features2d modules.

3.2.2 Scale-Invariant Feature Transform

SIFT points are points of interest in the image. They give the position of an area where all the pixels have approximately the same properties. Using the Laplacian of Gaussian of the image, maxima and minima which correspond to these areas can be extracted.

The biggest advantage of these points is the invariance in transformation like rotation, scaling and translation. Pattern detection algorithm based on this point is insensitive in term of scale and rotation. That is mainly why it has been chosen for this application.

As a result, by the time that the resolution of the pattern is over an acceptable threshold. We can move and rotate the object without altering the detection.

3.2.3 SIFT Descriptor

The descriptors are computed from the key points. The descriptors associated with the SIFT key points are locally oriented histograms around a SIFT key point [3]:

- We divide space around each key point (x, y) in N^2 squares of 4 by 4
- We compute the gradient $G_x(a, b, \sigma)$, $G_y(a, b, \sigma)$ for the 4 by 4 by N^2 points (a, b)
- For each 4 by 4 square, we compute a histogram of the orientations in 8 directions, multiplying by: (1) the module of the gradient (2) the inverse of the distance to the key point (x, y).
- To be invariant in rotation: the local orientation of the key point $\theta(x, y)$ is used as the origin (null orientation) of the histograms.

All the key points and the descriptors extracted and computed in the Constructors.

3.2.4 Homography

Just to give a simplified idea, the familiar Cartesian plane is composed by a set of points which have a one-to-one correlation to pairs of real numbers, i.e. X-Y on the two axis [1].

In our case, the points are the key points extracted. By extracting the homography matrix, we can transform the corners of our image to 4 other points representing the position of the object on the scene [2].

On the image behind we can see that the homography doesn't depend on the corners. All the key points can be a base to analyze the transformation. And using the perspective transform of OpenCV that multiplies the corners by the homography matrix. We obtain the position of the corners on the captured image.

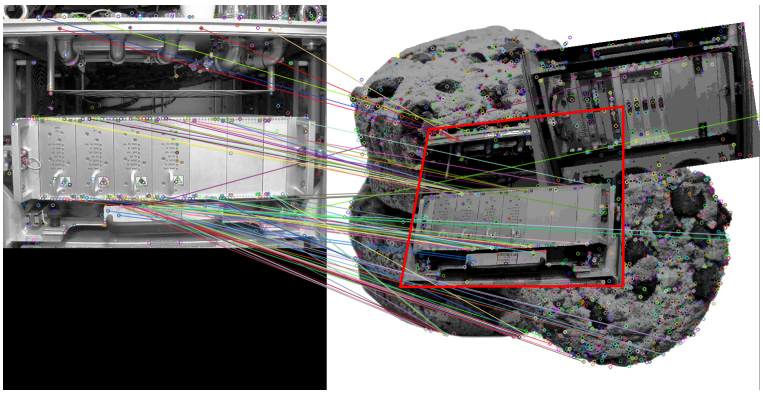

Figure 3.1: Homography with a corner hidden with openCV

3.3 Main functions

3.3.1 Main

The main idea of the system is to extract all the key points of the scene and all the key points of the pattern in the constructors. Like that you don't have to worry about it in the main program.

Then you compute the associated descriptors. Both of these containers (Keypoint, Mat) are implemented in OpenCV and really easy to use. The main loop consists to match the descriptors of the patterns one by one. Comparing them to the scene image we obtain a vector of DMatches that contains the information of each match. That is made by the radius matcher included in OpenCV.

Then you can perform a homography that uses the descriptors of the matches found, to extract an image that looks like the pattern we are looking for. The advantage of this method is the fact that it doesn't depend on the position and the scaling of the image. We also obtain the corners of the object found which allows us to remove the points of key points and descriptors associated from their respective vectors. That method avoids recomputing all the key points and descriptors which is very expensive in resources.

To stop the loop there are 2 options. First, there is not enough match between the two images. Secondly the number of key values removed after finding a pattern is too low. In both cases the match is not drawn on the image and the next pattern to analyze is set.

3.3.2 Constructors

Scene

The scene represents the vision of the camera and the current position of its analysis. It requires only the location of the image to be built. It sets the key points and computes the descriptors.

It uses a SiftFeatureDector from OpenCV with a minHessian (exigence of the detector) value of 10000 by default. It is a very high value but the tests shows that more this value is high, better are the results[1] and the quality of the image itself.

Pattern

A pattern represents an object that you want to be recognized in the image. It has simple attributes like a name and a matrix corresponding to the image it represents. But it also has more complex attributes like the vector of key points that we extract in the constructor[2], the matrix of descriptor that represents the properties of each point and the thresholds. The thresholds are separated in two different attributes, the minimum hessian that we told about just before in the scene section and the threshold properly spoken. The second one refers to the radius matching system, it is the maximum acceptable

[1] It also depends on the pattern minhessian value
[2] It can also be stored in a file

distance between two descriptors during the matching process. There is no universal value so we use numbers between 150 to 300 that we have to test one by one to see which one gives the best results.

The program contains a little algorithm that brings the possibility to try different values quickly.

Corners

Corners is a simple class containing 4 "point2d" (x, y point implementation of OpenCV). It also contains a method that determines if the point is inside the polygon represented by the corners. It uses the fact that a point is inside a polyline if a line drawn from the point to the infinity cross odd number of segments. It is use when we delete from the list the matches of the last pattern found.

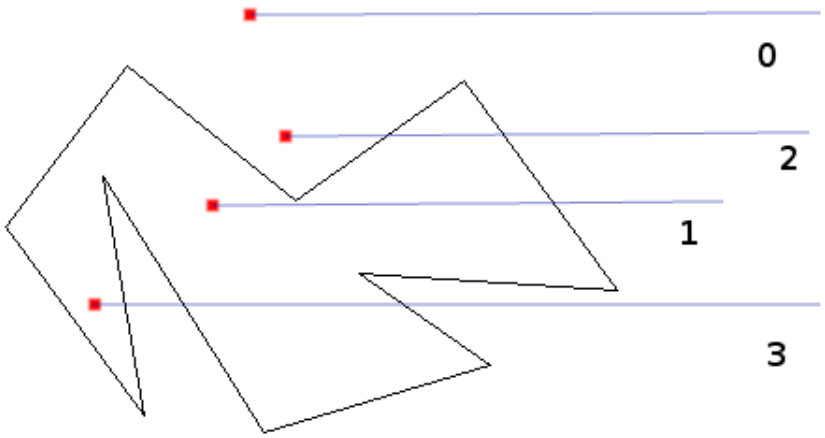

Figure 3.2: Illustation of the method to determin if a point is inside a polyline

Chapter 4

Run the application

4.1 Create a pattern

A pattern is simply a picture so take a picture of the pattern. Be carefull the edges of the pattern musn't correspond to something else than the pattern. Otherwise any algorithm will look for this type of edges instead of inner edges. Pay attention to the quality of the image. RBG is enough because the images don't have to be to heavy but the global quality have to be really good. Four time the size of the image you want to recognize on the scene is good, I will explain later.

Try the test algorithm on each pattern with : "./Scanner <location of the image> test". It will automaticly run the most efficants values and print 10 images. If you have good results everywhere put 400 minHessian and 150 as threshold they are the most commun values. Otherwise pick the best values. Pay attention that you can change the number of pattern you have to find in the code.

4.2 Launch the program

When you obtain the values, add to the pattern vector all the patterns you want to be analyse in the program. Create the scene and initiate the final image within the code.

If your not satified there is 2 possibility. The pattern can be out range, meaning that the SIFT points found on the scene don't exist in the pattern image and the homography is impossible to compute. Or one of the thresholds is bad, meaning that there is to many or to few matches. In both case the homography is impossible. That is why test the values before running the application is important. The success of the recognition depends on the the number of pattern you want to find[1].

[1] More there are several times the same pattern one the scene more the values have to be precise

List of Figures

Bibliography

[1] CorrMap. The homography transformation. http://www.corrmap.com/features/homography_transformation.php, 2013.

[2] D.J. Fleet & A.D. Jepson M. Brubaker, S. Mathe. Tutorial: 2d homographies. http://www.cs.toronto.edu/~jepson/csc2503/tutorials/homography.pdf, 2004-2007.

[3] Antoine Manzarena. Cours etasm. http://www.google.it/url?sa=t&rct=j&q=&esrc=s&source=web&cd=1&ved=0CCIQFjAA&url=http%3A%2F%2Fwww.ensta-paristech.fr%2F~manzaner%2FDownload%2FEPF_Bucarest2006%2FCours_EPF%2FChapitre3.ppt&ei=K4ujVdrjMcOssgHHg4bYDw&usg=AFQjCNEgM2Usx3DKDfBQbo3dXfu6dQ2nzw&bvm=bv.97653015,d.bGg, 2006.