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

Internship report pattern recognition UNISA Italy

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

Thanks

I had like to thank Mario Vento for hosting me in the labs, give me the oportunity to see conferences of foreign computer science doctors and following me during the intership.

I thank Pierluigi Ritrovato who gave me my subjects and guided my to my targets and answered to my questions.

Thanks to Hugo Descombes too, who helped me during the first part of my internship.

Finally, I want to thank Allessia that contacted me for the documents and gave me good advices on the matcher system.

Table of contents

Chapter 1

Introduction

During the second year of engineering school at ENSICAEN we have 3 month internship that I decided to do in the MIVIA Lab of Salerno's university in italy. They are specialised in image synthesis which is my major course, moreover I am interested in a double diploma in this unversity which gives me the opportunity to see the place and teachers, learn the italian and make some contacts.

My internship subject has been splited in two. First a study on the Motorola HC1 helmet used for exemple in military domain. The aim was to see if we could make the helmet compile Linux for helmet. Then developpe an application capable of recognise patterns in an image and store the position, the number of patterns found and the type of each pattern. So as to determine the name of a global structure which the patterns are composants.

In this report I will present the laboratory and the place of the internship, the study of the hardware for the helmet and the software (OpenCV mainly) for the application. What exists and where my projects are situated in their environnements. After that I explain what I tried to slove the problems and what are the results.

Chapter 2

Lab and place

2.1 The MIVIA Laboratory

for Macchine Intelligenti per il riconoscimento di Video, Immagini e Media which means intelligent machines for video, images and media recognition. The laboratory is located in Fisciano, Campania, Italy, near Salerno and Napoli as we can see below.

Figure 2.1: Location of the University

As its name suggests, except theaching computer science, doctors of the lab work on pattern recognition, classification, media analysis and many parallele subjects like autonomus drones and robot vision.

Because of the opportunity of double diploma, the proximity with France, the possibility to learn a new language even if I know that everybody speaks english in lab. I was really optimistic about the place and the laboratory.

2.2 Place and life in Italy

The first month of internship, I lived in Carpineto, 25 minutes to the university by foot. It was envoying the view on the montains in the morning and the evening but when I wanted to move the week-end, buses didn't pass near my flat so that I had easily hours of walk to get back home. I moved on to Salerno, in the centro historico. Than it was easier to move the week-ends.

The main tourstic place around is the Amalfitan coast which is really beautiful. There is also Napoli and Pompei which are really near. There is also a lot of beaches and montains. A really nice place where you can have snow in the winter and 33 degrees during weeks in the summer.

Figure 2.2: Picture of Amalfi early in the morning from the harbour

In the picture 2.2 the beach is not set for summer so there is nowone, but in july, it becomes overcrowded because of tourism. Also, I found that this part of Italy is really human. It feels like you can trust people really quickly because they are really sympathics. In that way, there is few offical contract. That is suprising when you come from France and it requires a thousand of documents to get a flat. Here, the ID card is enough. People are always happy but a bit slow when the walk that is getting on my nerves when I am in a hury. I took advantage of the intership to make a lot of trekking in the amalfitan coast and around Fisciano. Visite ruins, Napoli and see a lot of churches because christianism is really present in Italy.

Chapter 3

Evaluation of the need

3.1 Helmet HC1 needs

The HC1 Helmet from Motorola is a professional uses helmet dedicated to wide or site conditions, that is why it costs around 3000$ with the camera. It is used to show images in augmented reality in the little screen in front of the left eye. Full voice commanded, it uses a embedded version of windows which was the real problem. You can look at the figure 3.1 to have an idea of what it looks like.

Despite the fact that the Helmet is thought to add functionnalities, a client of the MIVIA Lab asked them if it was possible to install Linux or Android on the helmet. Even if they loose the voice command system and the drivers to run the camera etc. They asked me to try, at least, to do it.

Figure 3.1: Picture of the HC1 helmet with the camera plugged

3.2 The software needs

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 the laboratory asked me to do.

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

Chapter 4

Study of the existing

Once again this part will be splitted in 2 because the subjects are totally differents. I am going to talk first about the helmet, that required 1 month of test and studies. I will explain the procedures in the next chapter.

4.1 Helmet study

4.1.1 Helmet itself

I began my study by acquire some knowledges about the helmet itself. It has been release in the end of 2013, build for harsh condition, its price makes it unaccessible for the public. The army and building companies sow a good opportunity in this technologie to ease the work by bringing communication into the field.

The helmet is equiped with a batterie, Wi-Fi and blutooth connections, a camera and it can be wear under the work helmet. Everything is voice commanded and very responsive thanks to Motorola's work. Windows CE 6.0 is used on the last release of the helmet image. It is good for the next section to understand that the booting system and the update system of windows CE are related. That means that you can change the file system for an update but it is windows itself that validate and copy the files on the intern memory from the SD card.

4.1.2 Embedded systems

I learned a lot about embbeded system, mainly on the boards and all the materials related to it. In our case, the materials inside the helmet are not know and not published on the Internet. Pierluigi contacted the company that brought us the helmet but they couldn't tell us which board was used in the helmet. I must have guessed which TI technology it was because the datasheet reference a TI OMAP 3 microprocessor.

That is mainly why I put my effort on the "ISEE – IGEP COM MODULE" built in with a TI OMAP 3 processor. It was at the top of the art when the helmet released and the smallest board with this processor. It corresponds well with the size of the hardware

slot. In any case, if the linux kernel is compatible with the processor a cross compiled file system should boot and at least it should show an image on the screen.

4.1.3 Cross-compilation

Description

The cross-compilation is compilation for a different achitecture than the architecture that makes the computations. In the figure 4.1.3 you can see that the source code can be compiled for different architectures. The aim here was to compile a Linux kernel compatible with TI OMAP 3 processors from an Intel x86 machine.

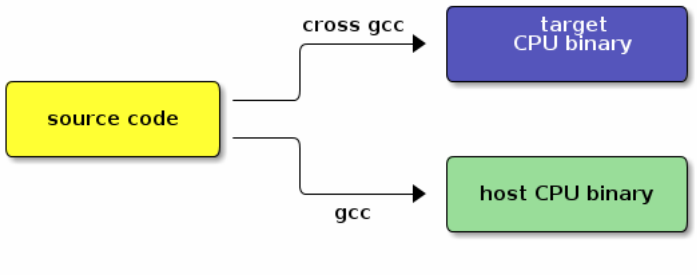

Figure 4.1: Cross compilation and compilation difference

Process

The cross tool chain is capable of compile a Linux kernel for a lot of achitectures. Of course it requires a long time to compile the toolchain and then cross compile the kernel to get the binaries but it is cost effective. The time necessere to compile the toolchain and the kernel on the TI OMAP 3 would be longer.

Toolchain

A toolchain is a set of distinct software development tools that are linked (or chained) together by specific stages such as GCC, binutils and glibc (a portion of the GNU Toolchain) [**?**].

A toolchain requires binutils such as assembler and linker, that produces the binaries. Also compilers for deferent languages like C, C++, Java etc, that transforms any language into another. A C library to gain access to kernel calls and a debugger that can be used or not during the compilation. We can see well on the diagram 4.1.3 from (avrfreaks.net's forum) where each composant is located :

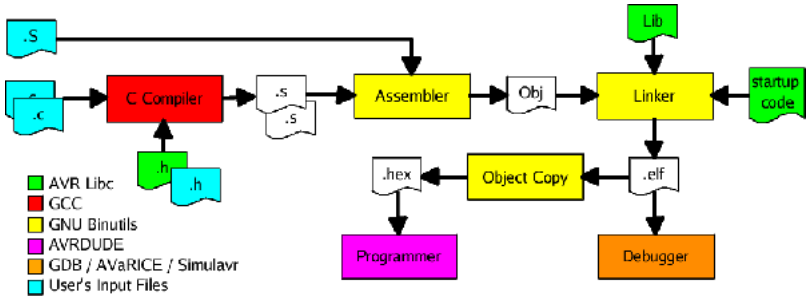

Figure 4.2: Cross compilation and compilation difference

4.2 Pattern recognition study

4.2.1 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 as a base and features2d modules.

4.2.2 Existing code

To get some examples of code and see how it works I simply looked at the OpenCV tutorials that are available on the internet. For example check it in the annexe section. It brings a simple way to extract a pattern and make the homography to draw the corners with lign OpenCV function.

Key points

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.

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 [?]:

- 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.

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 [?].

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 [?].

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.

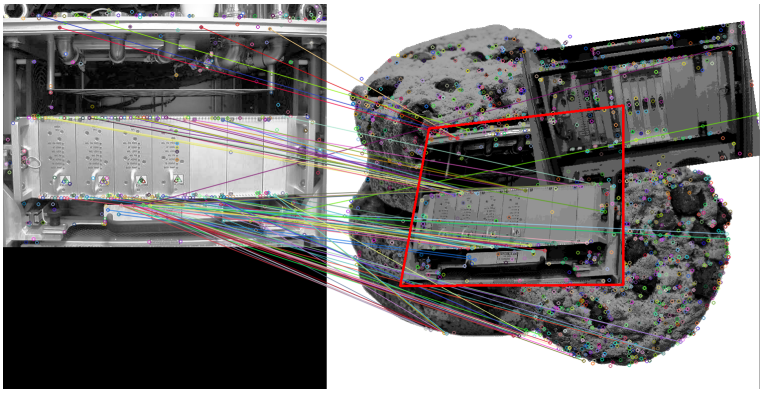

Figure 4.3: Homography with a corner hidden with openCV

Chapter 5

Achievements

This part will a melting pot of issues and success and how I found out how to correct them if I could. I will gather the installations for OpenCV and the cross toolchain because the main idea is the same. Then I will spit the technical part in two as usual otherwise it will not be clear enough.

5.1 Installation and tests

5.1.1 Helmet

Cross GCC

It is pretty easy to install, you have to get the archieve and compile the cross compiler for your architecture. First of all I saw in the tutorial that I was trying that a Ubuntu 10.04LTS[1] was required. I used virtual box to get a virtual machin[2] of this distribution on my windows OS.

Kernel compilation

Test

5.1.2 Pattern scanner

OpenCV installation

I installed OpenCV 3.0 which is the last version and tried it. It has a base for basic image applications and modules that you can install by adding them to the module folder and compile them. Unfortunatly one module required to perform the test was not working with the 3.0 version. It is the non free module. I tryed several times but I finally decided to switch back to a previous version. After uninstall everything and install the 2.4.9 version. The module was functionnal. I tryed the code of the tutorial I quote earlier

[1] Long Lime Support
[2] Software to emulate operating systems

and I had the first pattern recognition result. That took me 4 day in totall including the eclipse configuration that require adding all the libraries in the configuration of the compiler.

First build on eclipse

As I just said, to make it work on eclipse I had to set the configuration for OpenCV project. Otherwise I does not compile. Right click on the project, propreties, than go to the settings of my compiler (GCC C++) and select includes. There I added the path to the local install folder where the OpenCV's includes are located. After that I had to set the libraries to link one by one. In the same window I went in libraries section and add their names like opencv_core... And I added the search path of my local lib folder. Then I saved the setting and compilation was working.

5.2 Main problems

5.2.1 Related to the helmet

Hardware

Emulator

•

5.2.2 Related to the software

The matcher

After 3 days of stagnation making some tests and learing about the matching system. Allessia, the thesis companion of Mario Vento made me realise that was on the wrong way using the match function of OpenCV. In fact I couldn't find multiple pattern with that method because it selects the nearest descriptors[3]. The result was that I had to launch serval time the algorithm to get new matches again and again. I found the knnmatcher that perform a full matching and keeps the N best matches. That was not enough so I switched to the radius matcher that need a threshold parameter. That it finally how I introduce the threshold in the program. This value depends on the object, the distance and the conditions of the scene. In an other way the matcher keeps the matches which differ between the scene and the pattern less than the threshold.

The algorithm return a vector of vector of DMatches[4] so I put it in a simple vector of DMatches and it gave me very good results. The algorithm was able to find multiple time the same match on the scene. In the figure 5.1 we can see the fact that every time the pattern is represented it has matches in it. Not just the best ones which here corresponds to those of the first on the left.

[3] Those which have a minimum hamming distance likeness
[4] DMatch : A structure representing each match

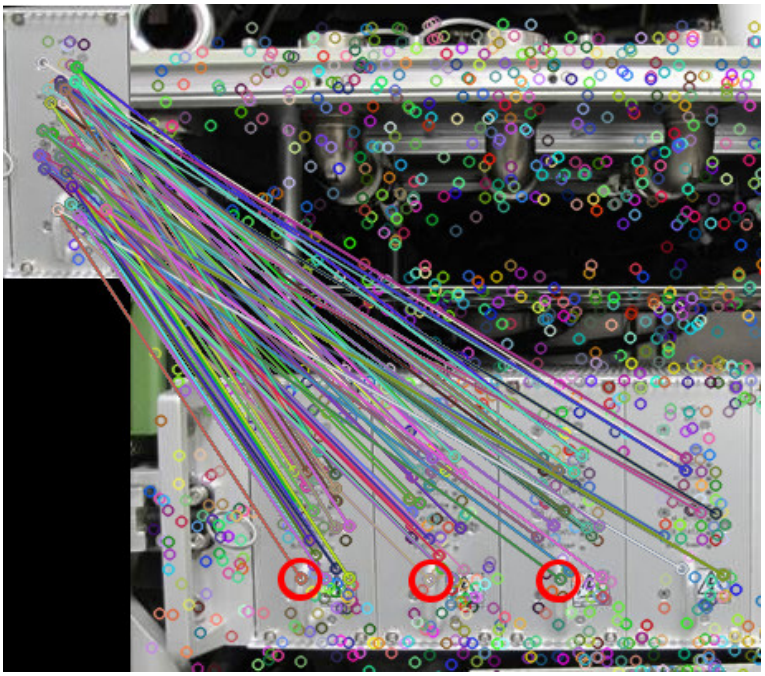

Figure 5.1: Simple draw of radius match with 3 times the same match surround in red

Get many time the same pattern

On this part we extract two possible methods with Pierluigi. The easy one was to draw in black the image and than make the homography another time. The hardest one was to look for the key points and descriptor inside the corners and earase them of their containers. Than do the homography again. I began by the second one that sounded more relevent.

To be clear I have to present the DMatch structure that is detailed in the annexes. The important attributs are the train index and the query index. They represent the fact that a descriptor has been found on the two images. But only the index of the descriptor is stored.

I first thought I could get easily the points that gave me the corners after the homography. In fact I needed to remove the points that concern the pattern that has been recognized by the homography. There was no other way but to extract all the indexes of the points inside the corners I just found and remove them from the vectors. By chance, the homography take to vectors of points describing the positions of the descriptors (and the key points by the way).

On the first version of this algorithm I was looking if the point was in the bounds represented by the 4 segments of the rectangle of the pattern found. I found out when I was rotating the image that the points was removed the wrong way because the shape of the pattern was a lozenge. A lot of points was concidered out of the bounds and other

that had nothing to do with the pattern was removed.

I found on the internet a method [?] that provides the answer of my problem in O(n). This method find the number of edges that ligne from a point cross to the infinite with a polygon. If it is odd then the point is inside the polygon otherwise it is outside. The picture 5.3 illustrate the idea.

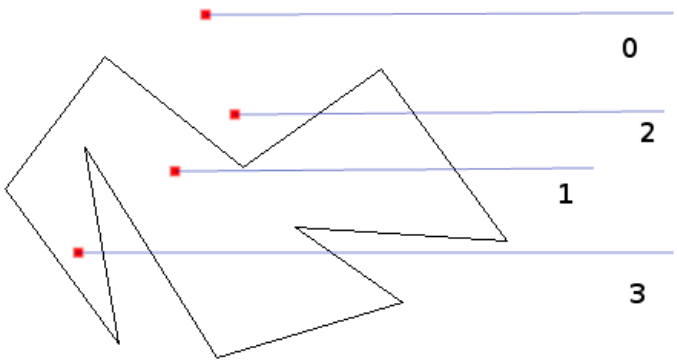

Figure 5.2: Illustration of the algorithm that tell if a point is inside a polygon

Corners

OpenCV does not implement a structure for the corners. The corners correspond to the vertices of the polygone that represents the pattern. I isolated them in the first place beaucoup that is a good proof of the fact that the algorithm found efficiantly a pattern. It also helped me to find out that the first version of the removing point algorithm was not working perfectly.

I let the structure as it is because I told myself that, in the future, the application could need the corners of the pattern to compute move analysis for exemple.

Others

My biggest parallel achivement is an algorithm that helps the user of the application to find the best threshold for his patterns. It depends of the number of pattern you want to recognize because often the threshold must be higher for certain conditions and more there is pattern in the image, more there is different conditions.

The algorithm take a scene and a pattern in input plus the number of time the pattern is represented in the image. The result of the algorithm is the first threshold that got all the patterns in the image. It computes a range of 150 values, that is why I can take 2 or 3 secondes sometimes.

```
manu@manu:~$ cd workspace/Scanner/Debug/
manu@manu:~/workspace/Scanner/Debug$ ./Scanner launcher_test MIU 4 test
base value threshold : 200
base value minHessian: 400
init done
opengl support available
component : MIU.jpg
Corners :
(1174.82,444.884) - (1161.82,540.462)
(995.836,418.483) - (980.346,513.714)
component : MIU.jpg
Corners :
(1189.17,345.908) - (1174.21,449.613)
(1009.87,322.949) - (989.461,427.661)
component : MIU.jpg
Corners :
(1216.4,166.815) - (1200.3,255.987)
(1033.75,147.362) - (1019.67,237.496)
component : MIU.jpg
Corners :
(1197.68,259.142) - (1189.82,354.518)
(1023.08,233.946) - (998.239,332.142)
```

Figure 5.3: Screenshot of the algorithm finding four times the MI unit pattern

We can see here that four units has been found with, for each, the positions of the corners. The treshold and minimum hessian values used are printed on top of the image. This values are generally good, but they can be higher if it needs to be less specific.

Chapter 6

conclusion

Bibliography

Chapter 7

Annexes

- **OpenCV tutorial :**
  `http://docs.opencv.org/doc/tutorials/features2d/feature_homography/feature_homography.html#feature-homography`

- **Code of the DMatch Structure :**

```
struct DMatch
{
    DMatch() : queryIdx(-1), trainIdx(-1), imgIdx(-1),
               distance(std::numeric_limits<float>::max()) {}
    DMatch( int _queryIdx, int _trainIdx, float _distance ) :
            queryIdx(_queryIdx), trainIdx(_trainIdx), imgIdx(-1),
            distance(_distance) {}
    DMatch( int _queryIdx, int _trainIdx, int _imgIdx, float _distance ) :
            queryIdx(_queryIdx), trainIdx(_trainIdx), imgIdx(_imgIdx),
            distance(_distance) {}

    int queryIdx; // query descriptor index
    int trainIdx; // train descriptor index
    int imgIdx;   // train image index

    float distance;

    // less is better
    bool operator<( const DMatch &m ) const;
};
```

List of Figures

Summary