Building Detection

Guillem Pratx – LIAMA

18 August 2004

1. Background

The technical objective of the project is to set up a methodology for map updating from very high resolution optical images on urban areas. This problem is of particular interest in the view of emerging applications such as GPS-guided navigation, traffic and pollution modeling, urban planning, etc. The fast urban growing observed in most of the major cities in China also requires rapid map updating.

The data are (i) very high resolution optical images and (ii) digital maps from Geographical Information System (GIS). The result of the process will be an updated digital map, that is, a map that translates the content of the image.

Therefore, it is necessary to have strong algorithms which detect the buildings on a satellite image to allow map updating.

2. Idea

2.1 Starting point

The idea I had was based on a demo program given with the OpenCV library, *square.c*. This program allows to detect squares on an image. The program was quite simple, it used a sequence of filters to obtains binary pictures, then from these pictures it returns the contours which :

-are convex
-have four sides
-have four right angles
-Have an area bigger than a given value

The demo program can be found on OpenCV website :
http://www.sourceforge.net/projects/opencvlibrary

2.2 Improvements

As long as a building is more than just a simple shape, improvements had to be made to the *square.c* program to detect buildings. I worked on two aspects : the image preprocessing and contours characterization.

2.2.1 Image preprocessing

Images I worked on are 8-bits grayscale images. In order to find building shapes, it is necessary to work on intensity gradient. Two approaches are possible. First approach can be to work on Canny transformed pictures, which gives you all the intensity gradients on a picture, which is two much to process. It is much better to consider filtering the picture by its intensity, which means keeping the pixels whose intensity is between I_1 and I_2. It allows to display shapes whose intensity is comprised between these two values, and detect gradients around I_1 and I_2. Next step is to scan and filter the picture with I_1 and I_2 taking values from I_{1min} to I_{1max} and I_{2min} to I_{2max}. For each one of the processed image, the polygons detection is launched.

2.2.2 Contours characterization

Given the complexity of buildings characteristics, a boolean definition was insufficient to characterize them. It drove me to adopt a probabilistic approach of the problem. That means that the function won't return a boolean value for each contour but will give a *score* reflecting the probability a given contour has to be a building. By keeping the highest score contours, it's possible to isolate buildings from noise (trees, roads, shades).

The main issue is to calculate a score that will reflects the probability each contour has to be a building. Thus, I considered the following characteristics :

-Convexity
-Area
-Angles
-Intensity (colors when working on 24-bit images)
-Accuracy of polygon approximation (which mean straight lines)
-Number of sides
-Gradients on image

When a contours, or its approximated polygon, has one of these characteristics, points are added to the score.

This score is then used for two things :
-To distinguish the buildings from noise
-When two polygons intersect, to choose the one that will best fit the building shape (i.e. that has the best score).

3. Implementation

3.1 The OpenCV library

The Open Computer Vision Library is a collection of OpenSource algorithms and sample code for various computer vision problems. I mainly used three kinds of resources :

3.1.1 The structures

OpenCV provides structures for images, points, geometric shapes, sequences of data (data may be any kind of data : points, polygons, even other sequences, etc …).
The use of *CvSeq* structure is complex because you have to manage both the physical storage and the pointers to the elements. Furthermore, it is impossible to release memory inside the physical memory storage. In the *findbuilding* function, even if the best has been done to avoid consuming too much memory, some unused memory is not released during the process (line 411 in *findbuildings.cpp*).
To circumvent the memory release problem, the function uses two physical storages : A temporary one (*temp_storage*, cleared regularly) and the main storage, from which memory cannot be released. Memory leaks happen if a polygon, which has been characterized as a building, is copied in the main storage and then a better polygon is found and is added to the sequence.
Anyway, the leak of memory is not important, as it only concern a few polygons (the size of each polygon is inferior to 160 bytes).

3.1.2 Image Processing and Analysis

I mainly used these algorithm to preprocess the image (filtering, removing noise), to find the contours and to draw the results.
I just noticed some problem with *HoughTransform*, basically some perfect lines are not returned by the function and moreover lots of noise is returned.

3.1.3 Structural Analysis

OpenCV provide some useful function which allow for example to approximate a contour with a polygon using a given accuracy, or to find the minimal circle enclosing a polygon.

3.2 Implementation of image preprocessing

The function *imageProcess* filter the pixel intensity between filter_down and filter_up, and remove the noise.

```
cvThreshold( src, temp, filter_up, 255, CV_THRESH_TOZERO_INV );
cvThreshold( temp, dst, filter_down, 255, CV_THRESH_BINARY );
```

These two lines will provide a binary picture in which the pixels whose intensity is between filter_down and filter_up appear in white, while the other pixels appears in black.

This image will contain blocks of white pixels (building for example) and noise (Fig. 1).

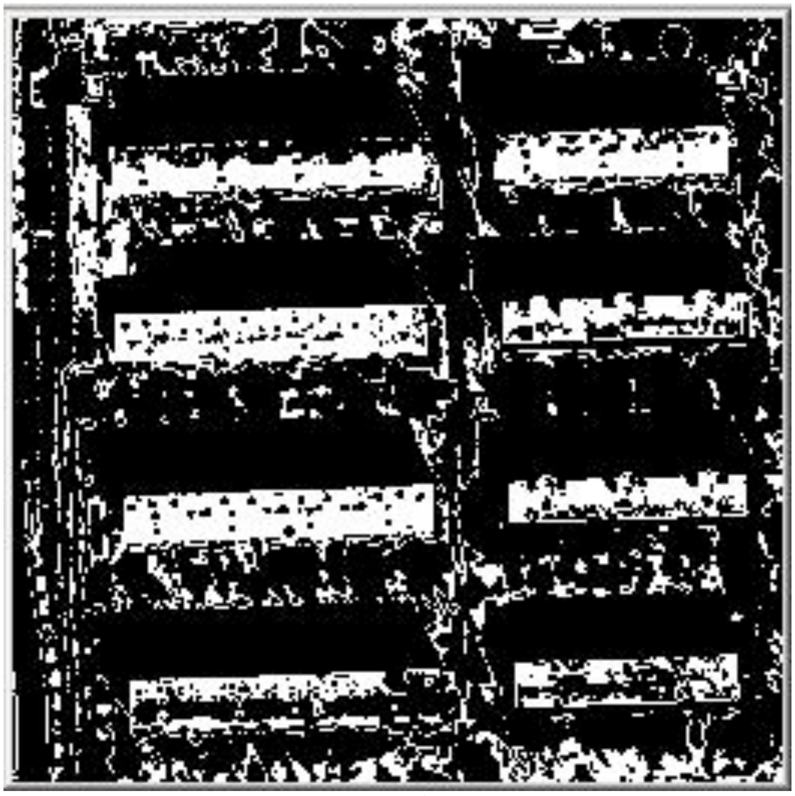
Fig. 1

The noise removal process will remove pixels whose intensity is very different from the average intensity calculated on a block of 3x3 pixels or 5x5 pixels. *The cvAdaptiveThreshold* function is used to do so. The result after noise removal is on Fig. 2.

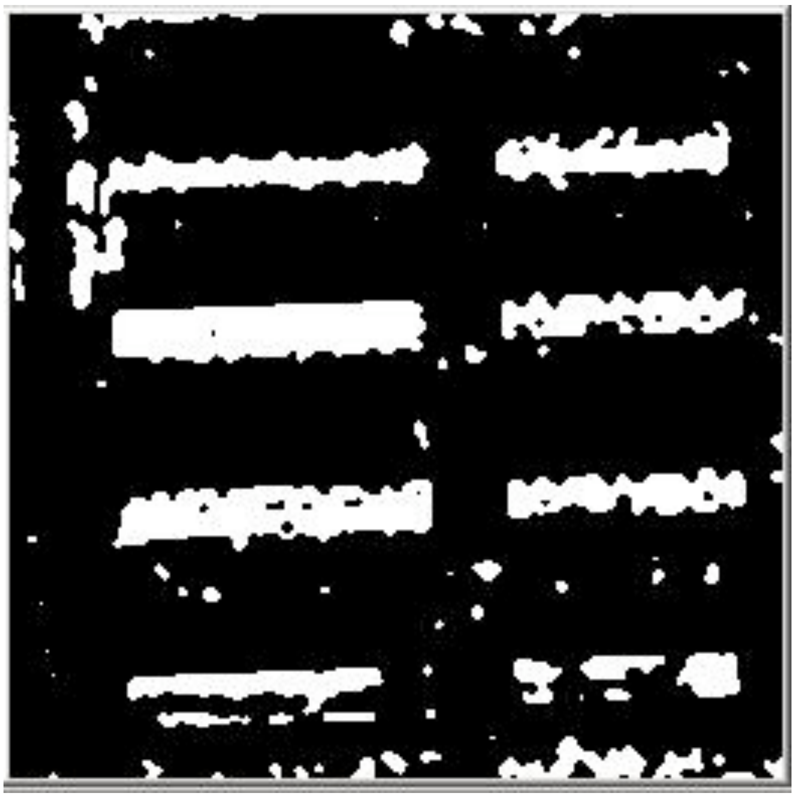
Fig. 2

3.3 Implementation of the building detection

The process may be divided in four steps :

3.3.1 Contours detection

The function will scan and filter the images with several filter ranges, in order to find different intensity level buildings. The image undergo two different scans, a high resolution scan to find building with a good precision and a low resolution scan to find building whose color is not homogenous. For each scan, two parameters may be adjusted : FILTER_WIDTH and FILTER_STEP. The function will filter the picture to find all pixels between [i - FILTER_WIDTH , i + FILTER_WIDTH], i varying from DARKEST_BUILDING to LIGHTEST_BUILDING and being increased by FILTER_STEP.

From the resulting binary image, the function first detects all the contours using the *cvFindContours* function. It returns a sequence of contours, each contour being a sequence of points. This sequence is stored in the temporary storage because it will be released when working on the following image. Then, for each contour in this sequence, a polygonal approximation is launched.

3.3.2 Polygonal approximation

Building shapes are mainly polygons, except from some which may be circles. Approximating the contours with a polygon allows to get a better shape for the building. c*vFindContours* takes a contour and return a polygon (a sequence of points). One important parameter is POLYGONS_ACCURACY. Given that the contours are far from being perfectly regular, a low value for this parameter will result in a fuzzy polygon, while a higher value will return a regular polygon. Values between 0.03 and 0.04 are usually suitable. The algorithm is very sensitive to this parameter.

Then, the algorithm tries to improve the polygons shape, by removing useless points for example (i.e. a point for which there is an angle of 180 degree), or by moving points to obtain 90 degree angles.

After that, for each polygon, the score is calculated.

3.3.3 Score calculation

It is now necessary to decide whether the found polygon is a building or not. The score takes into account several characteristics, intrinsic to the polygon or which are related with the original image, the processed image or the Canny transformed image.

First, the polygons whose area is under SMALLEST_BUILDING_AREA or over BIGEST_BUILDING_AREA or which have more than 15 sides are eliminated to save computing time.

Then, parameters intrinsic to the polygons are its convexity, its area, its angles and the number of its sides. The coefficient for each characteristic have been found in an empiric way, by successive tries :

```
#define SCORE_AREA_SMALL 12
#define SCORE_AREA_MEDIUM 20
#define SCORE_AREA_BIG 12
#define SCORE_CONVEX 17
#define SCORE_RIGHT_ANGLES 50
#define SCORE_4_BORDERS 18
#define SCORE_5_TO_7_BORDERS 10
```

Some parameters also take into account the source image or one of its transformation : the ratio between the area of the contour and the area of the polygon, the ration between the perimeter of the contour and the perimeter of the polygon, the matching between the Canny transform and the polygon.

```
#define SCORE_INTENSITY 50
#define SCORE_BORDERS_MATCHING 12
#define SCORE_PERIMETER 35
```

The last parameter is a handicap given to the low resolution scan which tends to return too much noise.

```
#define SCORE_HANDICAP 10
```

When the score has been calculated, only polygons whose score is over a given value will be added to the sequence of good polygons.

3.3.4 Polygons fusion

When adding a new polygon to the list, it is necessary to check if there isn't a similar polygon in the list. The operator & allow to calculate the ratio of the area of the polygons intersection by the polygons union. This ratio is 1 for to identical polygons, and 0 for polygons which have no common intersection. This allows to keep the sequence up-to-date with the best polygon possible (the polygon which has the best score). If the new polygons match with another polygon from the list, then the best one is kept.

There is only one case where the best polygon should not be kept : it's when the new polygon is recovering more than one polygon. Then, the new polygon should be kept even if it hasn't the best score.

4. Future steps

I see two directions for the improvement of the program. First, it should be possible to work on color images. It would allow to remove noise, most of the time caused by vegetation. Then, the building should have a color signature, thus by doing statistics on the buildings color it may be possible to increase the score respecting to the color of the building.

Then, all the coefficient have been determined in an empiric way. The fact is that the algorithm is very sensitive to a change of these coefficients. A statistical study of the result for identified building should help to find them with a better accuracy, by looking for coefficients which maximize the score for buildings.

5. Conclusion

It’s impossible actually to find 100 % of the building on the satellite map, so it is necessary to return found building with their probability.