Project: Vision goes Vegas

Recognition of the orientation of playing cards

by Stephan Senn, D-ITET

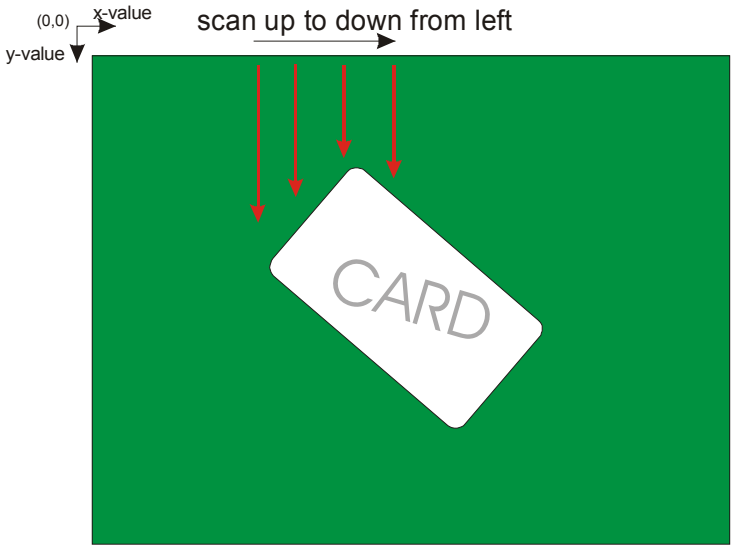
You got a coloured image in PNM-Format and you have to say where the card is, is not an easy task. And it is even more difficult if you take into account that the card could be everywhere in that image, in every orientation and your background is not always conformable. Your green carpet on which you set your cards may be covered with dirt (e.g. hairs, particles,...). And even if you try hard to keep your carpet clean the light may differ. The focus of your digital camera could take an image that is not as sharp as before. So you see, recognizing the exact position of the playing card depends on many factors. Implementing a program that take these problems into account was a basic part of our project of recognizing a set of French playing cards to enable our computer playing Black Jack.

First of all you have to scan your image to find the card. And even this simple task may drive you bananas. As I mentioned before the carpet is not conformable. So a little particle can be detected as a part of your playing card because the only way to find your card is the simple idea that the card is bright and therefore has got a colour that is mostly white or gray and your carpet is dark (green or black). So if the difference[1] between two following pixels is higher than our threshold value due to a particle on the carpet then our program will detect this pixel as a part of the border of our card and this may be totally false. The border of the taken picture is another problem. The image may contain a black border. Bad camera settings (square PAL instead of normal PAL) or problems due to inhomogeneity of light intensity on the detector of the camera are some reasons. So what can you do to face these problems? - The border problem will be solved quickly. You begin scanning the image from an inner border close to the original one. But what about the particles? - Here you can take a Laplace filter[2] that will sharpen the transitions of the card to the background much more than a particle to the background if you imagine the background as colours in a certain range. But be careful a too strong filter will also rise the transitions of the particles and then you got a lot of detecting points outside of your card and this make you very disappointed. So I found out that a smooth Laplace filter will do the job best.

By scanning the image from up to down from left and doing the same from right in a down-to-up-mode you will get all pixels on the border of your card. All? - Beside of the fact that you will use a certain step (5 seems to be good enough) there will be a case where you only detect pixels on two borders of the card instead of four. This happens when your card is parallel to the border of your picture. And even if your card is not parallel but in bad orientation of nearly parallel than you got less pixels on other borders. So what can you do? - Yes, you are right. You can make two other scans from left to right from downside and the same from right to left upside.

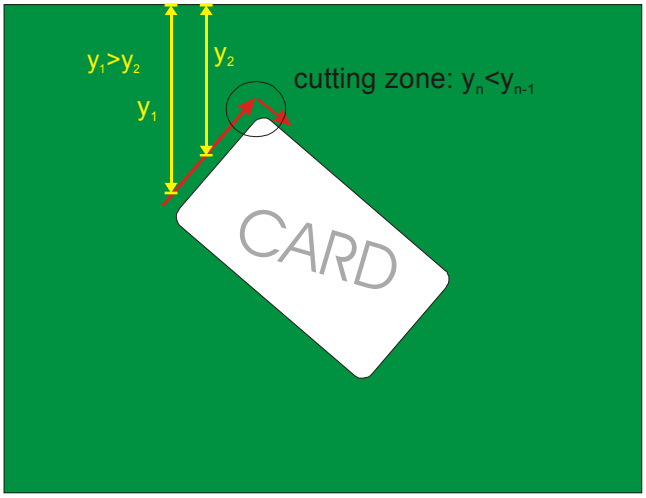
1 If you have a coloured image you can take the sum of the three basic colours (green, blue, red) to calculate the difference between two pixels. Each basic colour is represented by an unsigned integer value between 0 to 255. 0-0-0 means black and 255-255-255 means white.

2 If you have got a gray coloured image you are lucky to implement Laplace Filtering for only one colour channel instead of three for coloured images.



So now you have a lot of points on the borders of your playing card. What next? - Yes, you want to find out the orientation of your card and therefore you need the corners of the card. There are many ways to calculate these corners. I tried a lot of algorithm to achieve a very good result. While implementing a lot of ideas you have to learn that a corner of a card is round – so not a real corner! But you can only calculate corners of a rectangular and on my point of view only these corners can be used. So the rectangular of the card is a bit larger than the card itself due to the corners. As if you want to cut out the card you will cut out a rectangular first and then you will cut out the round corners. My algorithm works as followed:

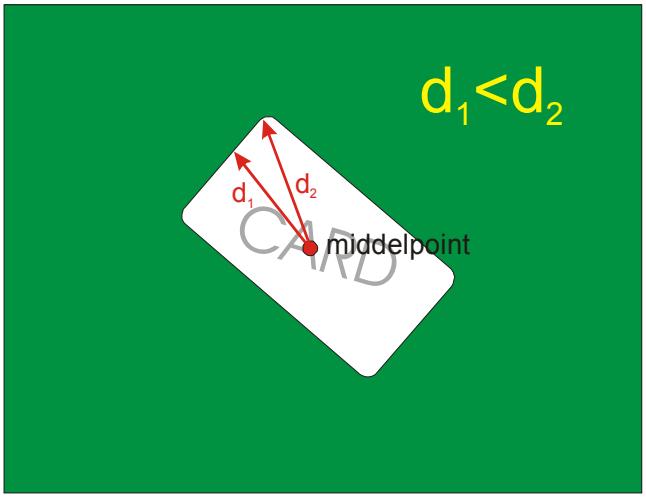
- The scanned data will be placed into a matrix where the first column stands for the x-value and the second one for the y-value.
- Now you have to divide the matrix so that every matrix represents points of only one border of the card.
- Next we will use Least Square Algorithm to calculate the rise and the offset of a border line.
- Then we will be able to calculate the intersection points of the lines and we will get our corners of the card.



This algorithm will work fine if you are able to cut a data scan into two pieces so that every piece represents points on only one border. And as you can imagine this can be implemented in a fast way. You only have to look whether following points are rising or whether they are falling. The

turning point will give you the place of cutting the data scan. The problem arises when your card is nearly parallel to the border of the picture and your rises and offsets of calculated lines tends to infinity. So you have to check first if your card is in parallel condition. This can be checked in an easy way: You only have to look whether the y-values of the top and down border are more or less equal to eachother. A bigger problem is the nearly parallel case. There you have to count the points of a data scan. If a scan of a border is less than 6 you have to scan from left to right and vice versa to achieve better results.

A better idea to split the data that you get from the scans is to calculate the middelpoint of the card. This special point can be easily found by calculating the average of all x-values and of all y-values that you found during the two scans from top and from down. Now you can calculate the distance between the middelpoint and one of your data points. The biggest distance of your radius is your cutting region of the data. But what about the first and the last values in the collected data matrix. There we have got also corners and therefore also biggest distances. So you have to begin with some data points a bit far from the corners so that you do not get into troubles. Perhaps you think that this method is very slow due to the fact that you have to calculate the square root. The square root is not necessary. You only have to calculate square of the radius. But you are right. This approach is truely slower than the first one.



There are many tricks to achieve good results and to shrink time. For example you can implement a flag so that the picture will be only read into memory but never written to harddisk back. Or you can work with pointers to avoid overhead handling big two-dimensional arrays. That is the reason why I implemented an own class for handling matrices. Although I tried hard to implement it economically there are always ideas to make it better. For example you can cut down time if you limit the scan range for a second scan because the card is only in a certain range. There are also faster implementations of the Laplace filter. So it is up to you to implement it better. And I am sure you will enjoy it.

Zurich, 4th February 2004

Stephan Senn