EFFICIENT INSCRIBING OF NOISY RECTANGULAR OBJECTS IN SCANNED IMAGES

Cormac Herley

Microsoft Research
One Microsoft Way
Redmond, WA 98052

ABSTRACT

Objects identification in images is generally hard unless the objects are simple geometric shapes such as circles, rectangles or have very particular properties. Even simple geometric shapes can be hard to identify if they deviate even a little from the ideal. We examine the question of identifying and segmenting noisy rectangles, where edges may be ragged, corners may be missing and so on. We test the robustness of our scheme on receipts and small documents obtained from real scans.

1. INTRODUCTION AND BACKGROUND

We are interested in extracting approximately rectangular objects from images. In particular a scan of several small objects such as receipts, business cards, airline tickets, photos etc will contain several objects that are approximately rectangular. In [2, 1] we described a robust system to recursively simplify the problem of segmenting an image with multiple objects into several single object segmentations. The algorithm then performed several single objects segmentations. So long as the scanned objects are very close to being rectangular simple techniques for solving the single object problem worked well [2, 1].

In Figure 1 we show an example of a scan with several problem images. It can be readily seen that several of the objects are only very approximately rectangular. For example several have corners missing, another has rounded corners and another has ragged edges; *i.e.* none of the objects conform closely to the rectangular ideal. Since the application we target is scanning receipts and other small objects the problem is quite common: objects such as receipts will often be folded or torn or otherwise undergo deformations before they are scanned.

To dispel the risk of confusion it is worth spelling out that we are not seeking the Least Mean Square or Least Median Square fit of a rectangle to the observed data. While these techniques work well for objects that are very close to rectangular they can produce capricious and irregular segmentations for objects as non-ideal as those in Figure 1.

Following the approach of [2, 1] we assume that the background color, B, is known or has been estimated and that every pixel can be classified as a data pixel or a background pixel. In the simplest case this can be done using a threshold

$$Im(i,j) = \begin{cases} \text{data} & |Im(i,j) - B| > T \\ \text{background} & |Im(i,j) - B| \leq T \end{cases} \quad (1)$$

Thus in a given image we will have a certain number of data and background pixels. For example in Figure 2 a full 56 % of the pixels are classified as being data, that is differ by more than a threshold amount from the background (mid gray) color. However some pixels in the interior of the object will be classified as background (*e.g.* white documents or photos of snow scenes on a scanner with a white platen) and some pixels in the background will be classified as data (*e.g.* marks on the platen due to ink and noise). Thus, it will not in general be possible to find a rectangle that encloses all of the data pixels without including some background pixels also. While our objects may be misshapen or torn all of them have at least two edges that are almost exactly parallel. In principal we will want to find the parallel edges and then find the top and bottom. We wish however to carry out the segmentation efficiently. This is an important constraint since it rules out two obvious approaches. The first, to use a Hough transform [3] to find the edges is very computationally expensive. The second, to use a matched filter to find corners or edges is also very expensive. The complexity of each of these algorithms scales as at least the square of the source resolution. That is, the algorithm running on 600 dpi data will take four times the time it takes on 300 dpi data.

2. APPROACH

We briefly recall that the recursive algorithm of [2] splits a image with multiple objects into sub-images containing single objects. By transforming the two-dimensional segmentation problem into one-dimensional problems this was done simply and robustly with extremely low computational load [1]. From this point we assume that we deal with images (or sub-images) that contain at most a single object.

0-7803-8554-3/04/$20.00 ©2004 IEEE.

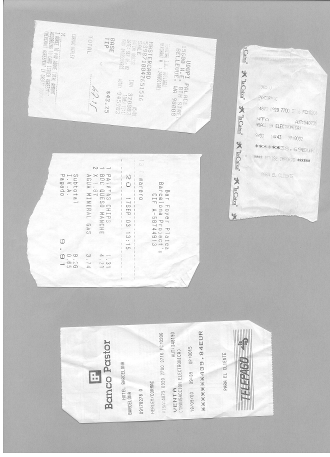


Fig. 1. Scanned image consisting of several objects. The algorithm of [2, 1] breaks the multiple-object segmentation problem into several single-object segmentations. Observe that each of the objects is only approximately rectangular. Efficient and accurate identification of the boundaries is thus difficult.

The situation then is as shown in Figure 2 where a sub-image from Figure 1 has been segmented. Call $P(j) = \sum_{i=0}^{N-1} Sgn(|Im(i,j) - B| > T)$ and $Q(i) = \sum_{j=0}^{M-1} Sgn(|Im(i,j) - B| > T)$. That is, $P(j)$ is the number of data pixels (as defined in (1)) on the j-th row and $Q(i)$ is the number of data pixels on the i-th column. These have also been plotted in Figure 2. In [1] it was found that the knee-points of $P(j)$ and $Q(i)$ could be used to determine the approximate vertices of the object, and then a search could yield the exact corner. There are a few problems with this approach however

- Many objects of interest are only approximately rectangular, and hence the knee-points of the $P(j)$ and $Q(i)$ trapezoids will be difficult to determine.
- Many objects entirely lack one or more corners
- Searching for the exact corner has complexity that grows with the square of the resolution.

Our goal will be to develop an algorithm that robustly determines an inscribing rectangle with complexity that scales linearly with resolution.

2.1. Complexity of Calculating Number of Data Pixels in a Rectangle

We wish to achieve, if possible, an algorithm that scales linearly with resolution. That this is challenging can be seen from the fact that even checking the number of data pixels in a rectangle scales as the square of resolution. That is, if we wish to determine how many data pixels, as defined by (1), lie in the rectangle with vertices $\mathbf{V}_0, \mathbf{V}_1, \mathbf{V}_2, \mathbf{V}_3$ we generally would expect the complexity to be proportional to the number of pixels in the interior (and hence scale a the square of resolution). We first show how this can be avoided. Instead of calculating $P(j)$ which is the number of data pixels on the j-th row, suppose we calculate and array $gP(i,j)$, which is the number of data pixels on the j-th row to the left of column i. This is calculated for an $M \times N$ image as follows:

```
[gP] = global(Im,B,T){
for i = 0, M-1{
     for j = 0, N - 1{
          sum = 0;
          if ( |Im(i, j) - B| > T)
               sum++ ;
          gP(i, j) = sum;}}}
```

Clearly $P(j) = gP(i_{M-1}, j)$. Label the left and right boundaries of a rectangle $L(j)$ and $R(j)$ respectively: that is $L(j)$ is the column number of the leftmost element of the rectangle on the j-th row. Similarly for $R(j)$. For the j-th row the number of data pixels in the rectangle is $gP(R(j), j) - gP(L(j), j)$. So if we sum over all rows

$$\sum_{j=0}^{N-1} [gP(R(j), j) - gP(L(j), j)] \tag{2}$$

we get the area of the rectangle. For a rectangle centered at i_0, j_0 an angle θ and dimensions L and W calculating $L(j)$ and $R(j)$ is a trivial matter. We will define a function

```
[ds,ts] = rectOccup(i0,j0,theta,L,W);
```

that returns the number of data pixels and number of total pixels in a rectangle using (2).

Observe that (2) requires only traversing the perimeter of a rectangle to determine the number of data pixels inside, and thus scales linearly with resolution. Determining the number of data pixels in a rectangle at 600 dpi will cost only twice as much as at 300 dpi once the single upfront cost of calculating gP is paid. In fact, $gP(i,j)$ is no more expensive to compute than $P(j)$, and since $P(j)$ was required by the splitting algorithm anyway, no additional cost has been incurred.

2.2. Finding the Center

A first approximation of the center of the object may be taken to be (i_x, j_x) where i_x is chosen such that $\sum_{i=0}^{i_x} Q(i) = \sum_{i=i_x}^{N-1} Q(i)$; *i.e.* the column for which half the mass of $Q(i)$ lies to the left and half to the right. Similarly for j_x. It must be stressed that this is an approximate value only, as the objects we deal with are not necessarily perfectly symmetric.

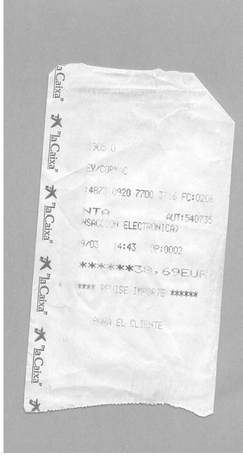


Fig. 2. Sub-image of the scan from Figure 1. This is the object from the upper right portion of the image.

Let us define a quantity $y(\theta, r)$ to represent the percent of data pixels on a wavefront at angle θ and radius r from the center (i_x, j_x). This can be calculated as:

$$y(\theta, r) = \sum_{k=-50}^{49} sgn(|Im(k\cos(\pi/2+\theta) + r\cos(\theta) + i_x,$$

$$\sin(\pi/2+\theta) + r\sin(\theta) + j_x) - B| > T).$$

Clearly when the wavefront is fully inside an object $y(\theta, r) \approx 100$ and when it is fully outside $y(\theta, r) \approx 0$. We define

$$d(\theta) = \min_r \{r \text{ s.t. } y(\theta, r) < 5\}. \quad (3)$$

In words: $d(\theta)$ is the least distance a wavefront at angle θ must travel from (i_x, j_x) before only 5 % of its pixels are data. We chose 5 % as a suitable threshold. It ought to be clear that $d(\theta)$ will be minimum when the wavefront is normal to an edge of the object. In Figure 4 we plot this quantity for the object in Figure 2. Clearly, as one might expect there are hills and valleys. A hill occurs at a corner, when the ray has to travel farthest to leave the object; a valley occurs when the ray points at right angles to an edge, since then the ray has to travel the least distance to leave the object. Further observe that the valleys are spaced $\pi/2$ apart, as one might expect. Hence, determining the angle of one of the edges of the object is oriented can be roughly estimated by

$$\theta_x = \min_{\theta \in [0, \pi/2)} \{d(\theta) + d(\theta+\pi/2) + d(\theta+\pi) + d(\theta+3\pi/2)\}. \quad (4)$$

2.3. Refining the angle

This estimate of θ_x is of course very crude and will need to be improved. We expect edges at roughly $\theta_x, \theta_x + \pi/2, \theta_x + \pi$ and $\theta_x + 3\pi/2$. Call these angles $\theta_0, \theta_1, \theta_2, \theta_3$. As can be seen from Figure 1 not all four of these directions will yield clean edges for all objects. In fact we expect at some of the θ_i no clean edge may exist.

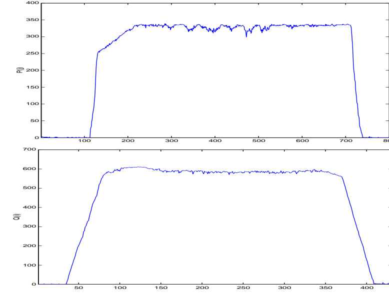


Fig. 3. The $P(j)$ and $Q(i)$ functions for image in Figure 2

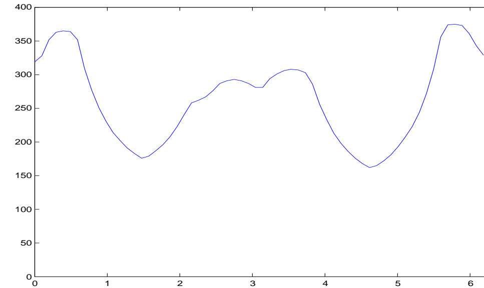


Fig. 4. The function $d(\theta)$ as defined in (3) evaluated for the image shown in Figure 2

We refine the angles using a simple bisection method on θ_i. That is if our and an upper estimates of θ_i are θ_l and θ_u we replace θ_l by $(\theta_l + \theta_u)/2$ if $d(\theta_l) > d(\theta_u)$, and we replace θ_u otherwise. By first evaluating (4) at a sparse set of values, and then iteratively refining in this way we can accurately estimate each θ_i economically.

Recall that by assumption the object has at least two parallel edges. Having independently produced the refined estimates $\theta_0, \theta_1, \theta_2$, and θ_3 we next chose the pair closest to parallel. We choose θ_0, θ_2 if

$$|\theta_0 - (\theta_2 - \pi)| < |\theta_1 - (\theta_3 - \pi)|$$

and θ_1, θ_3 otherwise. The smaller of the two quantities in the inequality above can be taken a a measure of how close the edges we have found are to parallel. A difference of greater than a degree or so can be taken as indication of a problem with the algorithm. We examine this on real data in the results section.

2.4. Expand from Center

Once the center of the object and θ have been accurately estimated it remains only to expand outward to find the largest

rectangle contained in the object. To do so we expand outward from the center, increasing the size of the rectangle. Every time we increase we measure how many of the contained pixels are data by using rectOccup(). It should be clear that we can split a rectangle into four quadrants; that is the overall rectangle [ds,ts] = rectOccup(i0,j0,theta,L,W) will be equivalent to the sum of the four rectangles [dk,tk] = rectOccup(ik,jk,theta,L/2,W/2), where

$$i_k = \cos(\theta + k\,pi/2)L/2 + \sin(\theta + k\pi/2)W/2$$

$$j_k = -\sin(\theta + k\,pi/2)L/2 + \cos(\theta + k\pi/2)W/2.$$

We estimate the dimensions of the object in sequence:

```
for Lt = minLen:maxLen
  for k=0:3
    [dk,tk] = rectOccup(ik,jk,theta,Lt,minWid);
  end
  if (threeQuadrantsNotGood(d0,d1,d2,d3)
    break;
  end
end
```

On termination Lt will be our estimate of one dimension, and similarly to obtain the other dimension Wt.

3. RESULTS

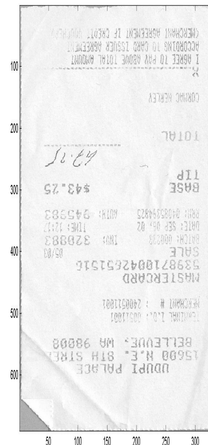
We implemented the algorithm and tested on a suite of 20 composite images, each of which consists of several small objects. Figure 1 is representative. The algorithm of [2, 1] was used to simplify the 20 multiple object images into 69 sub-images containing a single object each. Again Figure 2 is representative of a typical sub-image. All 69 images were segmented in a manner that seems reasonable to human observers. The worst case difference between the parallel edges was 1.76^o when we used 20 iterations in the bisection refinement algorithm. The average difference 0.96^o. In Figure 5 we show the segmentations for each of the objects from Figure 1. As can be seen the rectangle chosen by the algorithm accords well with our expectations.

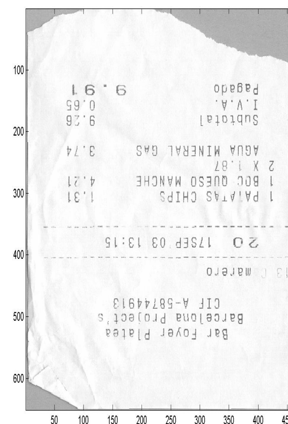
4. REFERENCES

[1] C. Herley. Recursive method to detect and segment multiple rectangular objects in scanned images. *IEEE Trans. Image Proc.*, 2003. Submitted. Available as MSR-TR-2004-01.

[2] C. Herley. Recursive method to extract rectangular objects from scans. *Proc. ICIP*, 2003.

[3] A. K. Jain. *Fundamentals of Digital Image Processing*. Prentice Hall, Englewood Cliffs, NJ, 1989.





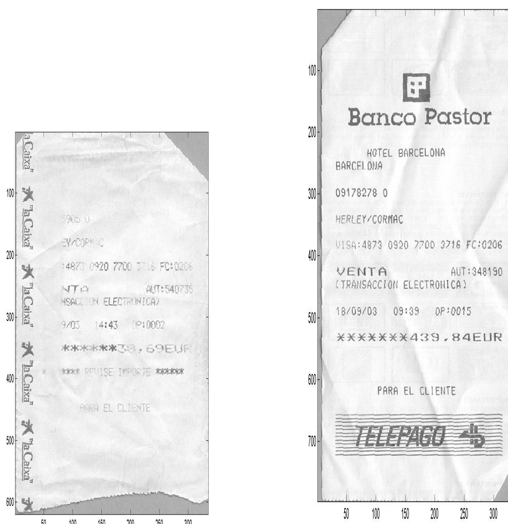


Fig. 5. Segmented objects from the scan in Figure 1. The object in Figure 2 is third. Observe that the segmentations in each case accord well with our expectations.