An Improved Edge Detection and Ranking Technique

Declan Murphy*
Computer Science Dept, Copenhagen University
declan@diku.dk

Abstract

A method is presented for refining the Canny edge detection algorithm, and subsequently ranking the resulting edges relative to their distance (in a topological sense) from the perimeter. This facilitates the selection of the appropriate level of detail for computer vision, particularly of articulated objects.

The first issue is the way in which the existing Canny algorithm chooses only one path where edges cross each other, resulting in erroneous gaps and spurious doubling of edges. An efficient pragmatic post-processing of the Canny technique is presented, which systematically bridges these gaps appropriately. The next issue is the automatic ranking of the edges.

1 Motivation and Introduction

The rationale and motivation behind this work was the requirement for a connected perimeter extractor with a variable amount of edge detail, tracing in from the perimeter. This need arose as an image pre-processing stage of a hand posture tracking system, in order to facilitate a deterministic, geometrical technique for reconciling image views with an articulated model. (For further details of this, see [6].) It should serve useful in any situation in which perimeter extraction of the foreground image, along with a specifiable level of edge detail in from the perimeter, would be advantageous.

Section 2 briefly describes the Canny edge detection algorithm. Section 3 covers the edge ranking with a description of the perimeter extraction, §3.1, and gap bridging, §3.2. Section 4 points towards how this technique serves the further stage of processing of reconciling image views to a model of an articulated 3D body. Section 5 winds up with a summary and conclusion.

*Most of this work was carried out while the author was visiting the cART lab of CNR, Pisa, with subsequent refinement and write-up at the InfoMus lab of DIST, University of Genoa and at DAIMI, University of Århus.

2 Canny Edge Detection

One standard, or simple, approach to finding the perimeter of an object is to take its silhouette, which is very easy to compute if (as in the case motivating this work) there is a single foreground object of interest with a known or controlled background. However, the exact position of the silhouette border varies with the thresholding level used to compute it, and with both the source direction and the level of the ambient lighting.

A more accurate approach – almost immune to these weaknesses – is to use edge detection. As we may be dealing with low resolution images (to enhance real-time performance), such accuracy is desirable to offset coarse discretisation inaccuracy. Another significant advantage is the ability to make out some of the inner detail instead of perceiving just a single blob. This becomes particularly important if an articulated object is being tracked, as explained in §4.

Edge detection, in computer vision, is defined as the process of assigning a value to each pixel of an image in proportion to the likelihood that the pixel is on the boundary between two regions of different intensity values.

The Canny edge detection algorithm [1] is used – as implemented in the Intel Open Computer Vision Performance Library (OpenCV) [2]. This was chosen for its sharp results (in comparison to rival edge detection techniques), and for its efficient implementation (which was readily available).

2.1 How it Works

First the image is smoothed by Gaussian convolution; then a simple 2D first derivative operator is applied to the smoothed image to highlight regions of the image having high first spatial derivatives[1]. Edges give rise to ridges in the gradient magnitude image. The algorithm then tracks along the top of these ridges and sets all pixels that are not on top of the ridge to zero, so as to give a thin line in the output.

Two thresholds limit the tracking: t_1 and t_2, with $t_1 < t_2$. Tracking can only begin at a point on a ridge higher than t_2. Tracking then continues in both directions out from that point until the height of the ridge falls below t_1. This hysteresis helps to ensure that noisy edges are not broken up into multiple edge fragments.

2.2 The Y-Junction Effect

One problem with the basic Canny operator is that of so-called 'Y-junctions': places where three (or more) ridges meet in the gradient magnitude image. Such junctions occur where an edge is partially occluded by another object.

[1] *Spatial derivative* simply refers to how much the image intensity values change per change in image position. See e.g. [3, ch. 5] or [4] for further background on edge detection.

The tracker will treat two of the ridges as a single line segment, and the third one as a line that approaches, but doesn't quite connect to, the other two. See figure 1.

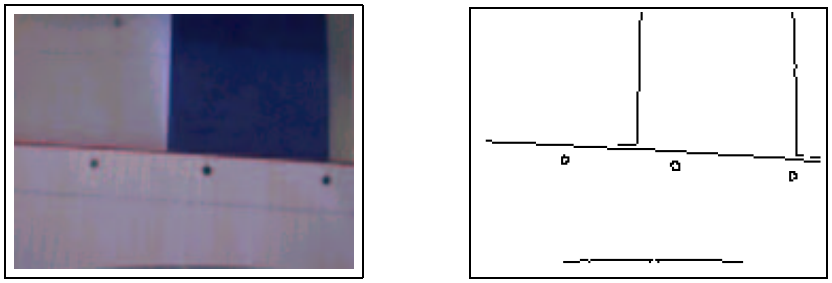

Figure 1: *Canny Edge Detection and its 'Y-Junction' effect. Left: camera image; right: Canny edge image. Note how the two vertical lines approach but fail to contact the central horizontal line, resulting in a spurious doubling up of a length of the horizontal edge.*

3 Edge Ranking

A rank (low non-negative integer) is associated with each edge according to the following rule:

- 0 corresponds to edges on the perimeter,
- 1 corresponds to edges terminating on the perimeter which are not of rank 0,
- n corresponds to edges terminating on an edge of rank $n-1$ which are not of rank $< n$.

See figure 2.

Later stages of processing have the option of having an image frame composed of all edges up to a specified rank, thereby selecting the appropriate level of detail.

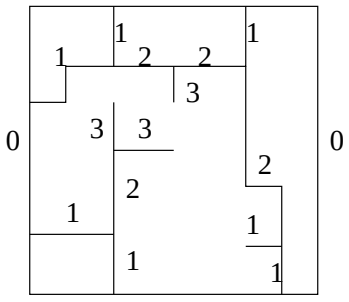

Figure 2: *An example of some edge ranks.*

3.1 Perimeter Extraction

Taking the edge detection from above, §2, the perimeter is in turn extracted. While this may sound trivial – as indeed it is to our human eyes – it is not so for the computer: there may be (and in general will be) some spurious noise outside of the object of interest, and the perimeter will not be continuous in general.

First the image is scanned, from an outer edge of the frame inwards, for any "blob"[2] in such a manner that any enclosed shape will be approached from the outside (even if this shape is not connected[3]). (Actually, there is a very remote possibility that the scan may first encounter the object of interest from the inside, but in fact this does not interfere with the success of the algorithm. At worst, it will only marginally slow the process in the case that only low rank edges are required.)

Next, the perimeter of the found shape is traced, paying careful attention to always keep to the *outside* of the shape while considering the next pixel along the perimeter. This is achieved by:

1. maintaining the concept of "the last outer pixel", for each pixel as we trace along the perimeter,

2. rotating, from the last outer pixel to the next pixel of the shape, in a direction (always clockwise or always anticlockwise) consistent with the angular polarity of the trace.

In other words, every time we locate another perimeter pixel, we record the last non-perimeter pixel (on the outside). In order to find the next perimeter pixel, we rotate about the immediate neighbouring pixels – starting with the next pixel after the last outer pixel, in a consistent direction of rotation. See figure 3.

We are assuming from here on that edges outside of the object of interest (if any) have been removed or are disregarded in the processing. This may be achieved by such techniques as gauging the length, whether it lies close to other edges, etc., as described in, for example, [5]. This has not been a problem in the prototype.

3.2 Bridging Gaps

Now if we try to trace along the outside of the shape we have found, we will find that, very often, the trace only runs around a section of the desired object perimeter because of gaps left by the Canny edge detection (§2.2).

[2]Here, a "blob" simply means a number of pixels all above a certain intensity value, each one of which is adjacent to at least one other such pixel.

[3]The term *connected* is borrowed from topology: here it simply means that the shape in question can by entirely traced through adjacent pixels.

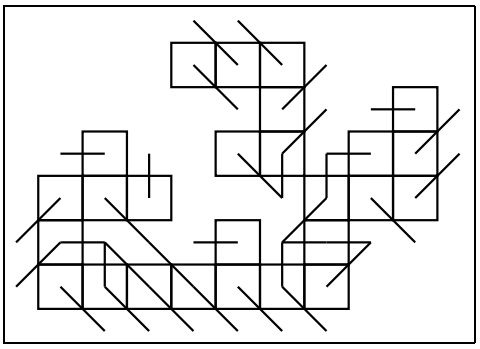

Figure 3: *Maintaining the "last outer pixel". Each pixel of the perimeter is represented by a square, with a straight line segment from its centre to the centre of its last outer pixel, rotating anticlockwise.*

Such gaps come in various forms, so that a general technique to bridge them all is required. Some examples taken from observation of real data appear in figure 4. The technique found to cover all cases, tested with real (hand) data, is as follows:

1. Starting with the first perimeter pixel found, continue tracing anticlockwise.
   (a) If trace describes a full circuit, then we are finished.
   (b) Otherwise record the end pixel and the last outer pixel of the second[4] from end pixel.
2. From the initial pixel, trace clockwise.
   (a) Record the end pixel and the second[4] from end last outer pixel.
3. For all ends, try to join them.
   (a) Rotate from outside to inside, scanning along a radius of the appropriate size[6].
   (b) If another segment is located, join to the nearest pixel on it.

[4]We use the last outer pixel of the second to end pixel because that of the end pixel will have swung around into the inside. Recovery of the outside may be made by rotating back[5](if the segments tend to be long) or by placing the last outer pixels in a two cell Last In First Out (LIFO) buffer (if the segments tend to be very short).

[5]In rotating backwards, the temptation to take the short cut of rotating on through the edge must be resisted! See figure 5.

[6]There is a certain radius (or aperture) size observed to be both large enough to bridge all Y-junctions and small enough to not interfere with other edges in the vicinity, for a given image resolution and a given distance from camera to subject.

(c) If this new segment is already on the list of ends, merge it (on the list) with its new partner; otherwise, trace the extent of the new segment, updating its end data.

4. If any segments remain (implying that all segments end in spurious diversions), try to join them. (This case occurs only very rarely in practice.)

   (a) Back-trace along the segment away from the end, testing within the aperture from outside to in, and otherwise proceed as in 3.

   (b) If no join is found, delete end from list (consider as noise).

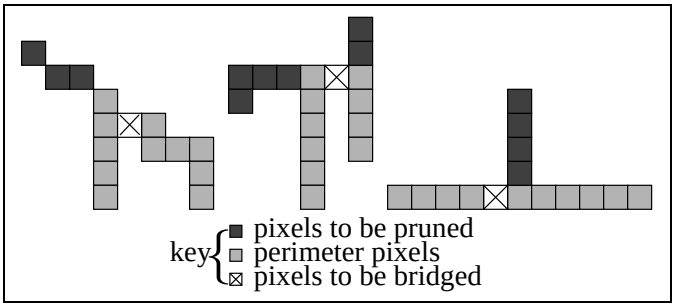

Figure 4: *Bridging the perimeter gaps. Internal bridging is similar.*

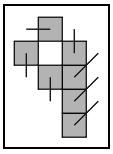

Figure 5: *A counter example of why rotation must be backwards, not onwards through the edge (which would get stuck in the hole!).*

3.3 Ranking

If the desired rank is greater than zero, then all perimeter ranks are first assigned their zero rank. Then, using the above gap bridging technique (but without further heed to maintaining a sense of outside, and joining to all other edges within the seek radius), all adjoining edges are given rank one. This process is iterated until the desired rank is attained: all edges adjoining those of the current rank, n say, which have not yet been assigned a positive rank, are now given rank $n + 1$.

4 3D Reconstruction of Articulated Objects

In trying to reconcile a model of an articulated body with camera images, it is generally useful to take note of the edges arising from creases along the articulation joints, thus yielding cues to the location of the joints, and facilitating recognition of the angles of flexion of the joints. Finer level detail, however, is generally undesirable at this stage, as it only introduces noise or needless complication into further stages of processing.

Moreover, in viewing the projection of articulated (or, indeed, non-concave[7]) three dimensional bodies to two dimensional images, some (or many) of the external edges of the body in three dimensions become internal edges after projection. See figure 6.

Figure 6: *Note how the edges of the fingers (external to the three-dimensional hand) become internal edges in the two-dimensional image. In this image, all the visible edges of the fingers are of rank 0–3, while the white sheen of the four distal interphlangeal joints and the dark crease of the distal interphlangeal joint of the thumb are of rank 2 or 3. Thus, an image of all edges of rank* ≤ 3 *is most appropriate.*

The doubling up of edges can be dealt with by judicious setting of the the two thresholds (t_1, t_2 from §2.1).

Thus armed, with this technique for accurately extracting perimeters and edges of a given rank, and knowing the rough position/orientation of the body (e.g. from previous frames, velocity and model constraints), it becomes relatively straight-forward to employ a technique for selecting those edges which correspond to the perimeter of the body in three dimensions, and to have articulation creases and other such desirable features at will.

5 Summary and Conclusion

The technique is demonstrably (see figure 7) neat and efficient enough to be simply appended to existing edge detection algorithms with negligible real-time performance penalty. By supplying edges with (or up to certain) ranks, the appropriate level of detail is immediately brought into focus (including, by way of corollary, perimeter extraction), thereby directly facilitating the reconciliation of camera images with models of articulated objects and significantly reducing redundancy in further computer vision processing.

[7] $X \subset \mathbb{R}^3$ is concave means that $\mathbf{x}_1, \mathbf{x}_2 \in X \Rightarrow \{(1-\lambda)\mathbf{x}_1 + \lambda\mathbf{x}_2 : 0 \leq \lambda \leq 1\} \subset X$

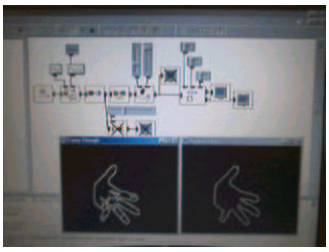

Figure 7: *Screen-shot of an EyesWeb patch having this technique embedded as an image pre-processing block. Note how the sub-image on the left (Canny edges) is cleaned showing only the perimeter in the right sub-image.*

The source code in C++ may be freely (GPL) downloaded from [7] both as an EyesWeb block and as standalone code. A report of the emerging hand posture tracker may also be found there.

References

[1] John Canny. A computational approach to edge detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-8(6), November 1986.

[2] Intel Corporation. Open source computer vision library. WWW. `http://www.intel.com/research/mrl/research/opencv/`.

[3] E.R. Davies. *Machine Vision: Theory, Algorithms and Practicalities*. Academic Press, 1997. 2nd edition, pp. xxxi + 750.

[4] Robert Fisher, Simon Perkins, Ashley Walker, and Erik Wolfart. The hypermedia image processing reference. `http://www.dai.ed.ac.uk/HIPR2/hipr_top.htm`.

[5] Declan Murphy. Extracting arm gestures for VR using EyesWeb. Barcelona, Spain, Nov 2001. Audiovisual Institute, Pompeu Fabra University. ISBN: 84-88042-37X.

[6] Declan Murphy. Building a hand posture recognition system from multiple video images: A bottom-up approach. Technical report, interim, cART lab, CNR, Pisa, Italy, March 2002. `http://www.diku.dk/users/declan/pub/cart.pdf`.

[7] Declan Murphy. A hand posture recognition system. Anonymous FTP GPL downloadable software, updates, reports, papers, and links to applications as they become available, 2002. `ftp://ftp.diku.dk/diku/users/declan/hand/`.