Chapter 5

Edge Map Refinement: Sequential Edge Linking (SEL)

A problem which is often underestimated in literature concerns the correct edge position detection. In fact, most of the literature focuses on the edge detection which is different from the correct localization of the edge pixels.

Using classical spatial edge detector filters (see Chapter 4), each pixel of the edge detector output has a value proportional to the strength of the edge passing through that pixel. Theoretically, only edge pixels should have a value different form zero and proportional to the edge strength. Unfortunately, even leaving out the noise effects, the pixels near the edge have values higher than zero because of the transitory effect of the filters. Hence, after applying a threshold T to that output (e.g. in order to obtain a desired P_{FA}) the revealed edges are thicker than one single pixel.

A morphological thinning [38] is often used in literature to solve this type of problem and to make the edge thickness down to one pixel. Anyway, the previous operator suffers from two main problems. Firstly, it tends to select the central pixel as border one even when it is not always the case. Note that this result would be correct if the image was symmetrical near the edge, but it is well known that this case is seldom found in real images. Secondly, it gives rise to some “spurs” (parasitic branches) near the real edges which are difficultly removed without losing important parts of the true edges.

Another operator, which could be used to solve the thickness problem of the detected edges, is the one proposed by Bovik [39]. It performs the logical AND between two filter outputs already thresholded: the edge detector output and the Laplacian of Gaussian (LoG) output[7] [39]. This operator is expressly studied to tackle the multiplicative noise affecting SAR images but unfortunately, its results are strongly dependent on the parameter σ (standard deviation) which rules the LoG filter spatial extension. Furthermore, even though the revealed edges are one pixel thick, they are mathematically forced to form a close curve and this constraint produces a very smooth shape for each revealed edge.

In order to properly address the edge position detection problem, the algorithm proposed in this document uses a local search among near pixels. This search is used to find out which pixel has the highest probability of being an edge. This type of approach has been proposed by Canny [25] and consists of local non-maxima suppression (NMS) orthogonally to the edge direction (i.e. along the gradient direction). Clearly, if each pixel of the edge detector output is inversely proportional to the edge strength (Ratio of Averages-RoA filter) the procedure applied is the non-minima suppression. The main advantage of this type of procedure is the following: not only does it generate one pixel thick edges, but it manages to retrieve the most probable edge position. It is also worth noting that, when RoA filter is used for edge detection, the results of this procedure are the same as those obtained by Likelihood Ratio test which yields the optimum results. In fact, it is demonstrated in [7] that the non-minima suppression applied to the RoA yields the same results as the optimum method, with the huge advantage that the RoA probability density function (pdf) is completely known whereas the Likelihood Ratio test pdf is not.

[7] It is nothing but the second derivative of the signal firstly smoothed by a Gaussian filter.

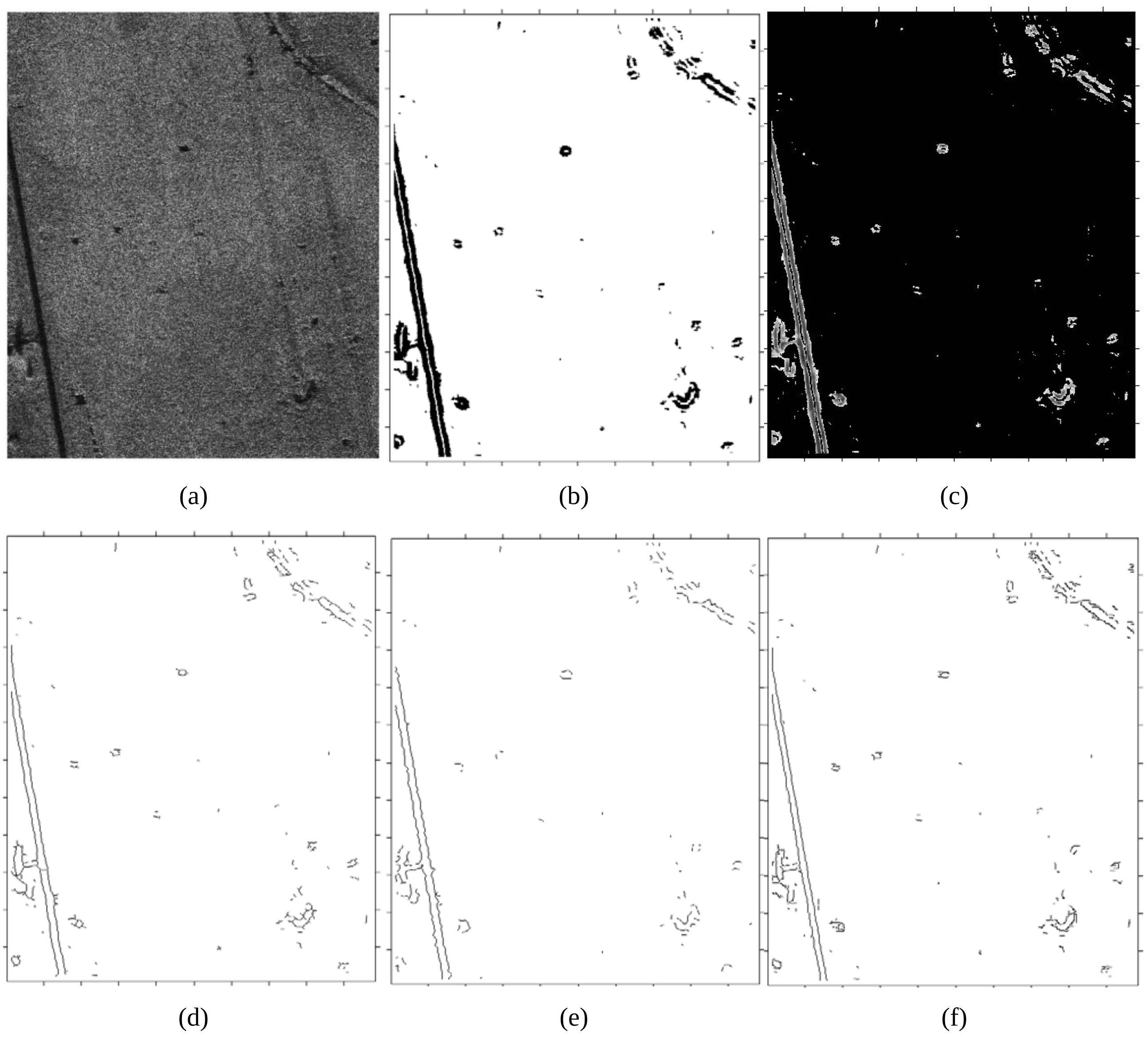


Fig. 5.1 – Results on MSTAR image "HB06173". (a) MSTAR image "HB06173". (b) Thresholded RoA output. (c) Output of the RoA where only pixel values under the threshold are reported. Pixels whose values are over the threshold are set to zero (d) Morphological thinning output. (e) Bovik operator with Gaussian standard deviation *σ=3* which provides the best results. (f) Non-minima suppression results.

In Fig. 5.1 some results of the previously mentioned methods are reported. In particular, in Fig. 5.1 (d) we can see some parasitic branches along the road borders yielded by the thinning operator, in Fig. 5.1 (e) the wavy shape of the boundaries for the Bovik's method is pointed out and finally, in Fig. 5.1 (e) we can see the clean results of the non-maxima suppression method.

In order to recover missed parts of the edges, a double thresholding (DT) method (hysteresis thresholding) could be applied. Also this procedure has been proposed by Canny [25] and it allows us to retrieve the most meaningful parts of the edges which are not revealed by the previous step. This procedure uses a weaker threshold than the one previously used in the edge detection step. In this way weaker edges are now revealed. Clearly not all the edges of greater strength than this second threshold are retrieved, but only the ones which have at least one pixel in common with a strong edge. In other words, only weak edges connected with a strong one are revealed by this algorithm. Although being very simple, this procedure is very effective and it does not modify the output P_{FA} set by the first threshold. Actually, the double thresholding used here is an improved version than Canny's. In this version, only *weak edges* which have a direction "near" (at most $\pm\pi/4$) from that of the *strong edge* are retrieved. This improvement enables us to limit the birth of "spur" edges. Anyway, if the boundary is curvilinear, this type of parasitic edges can still raise and we should limit the direction of weak edges to that of the strong one (i.e. without allows to go "near" but only in the same direction) at prize to lose some detections. Clearly, spur edges can be deleted after applying iteratively some morphological operators [37], but at the expense to lose edge information from small edges which have a length equal or less of the maximum spur edge length admitted [37]. In order to solve this problem, when applicable, many author suggest of using the Hough Transform because it is fast, simple, and can retrieve the total object boundary without the need of having revealed it completely at the edge detection stage. Nevertheless, when the image scene is complicated and some parts of the object boundary miss, the Hough Transform can link together boundaries belonging to different objects (see Chapter 6). This possibility is in inverse proportionality with the amount of the object boundary retrieved at the edge detection stage. Hence, the linking stage becomes necessary when we process most of the real images.

The idea which underlies the sequential linking is just to draw the edges which are in an image automatically, in the same way a human being could do. Starting from this very clear reasoning, the question moves to what tools should be used in order to achieve that easy goal. In the same way a user would do, the basic idea is to draw an edge starting from a point and go on, step by step, adding a new piece each time. Since an edge has always a start and an end point, authors have clearly modelled this problem as a "shortest path" problem, i.e. how to go from the start point up to the end point in order to stay by the real edge closest than possible. The first literature refers to the work of [40], in which the problem is generalized as the very known and treated issue of finding the "shortest path" in a graph. In that document, the dynamic programming framework to solve that type of problem is proposed, but no general metric is presented to compare different paths together. The authors in [41]-[44] tackles those kinds of problems and also faces the issue of the heavy computational load of the dynamic programming. In order to clear these concepts, the model proposed by [41]-[44] is given hereafter.

5.1 Theoretical Background

Given a point (x_i, y_i), a path of length $N+1$ can be represented as an ordered set of points $p = \{(x_i, y_i)\}_{i=0}^{N}$. Nevertheless, it can be also specified by a start point (x_0, y_0), an initial direction $\boldsymbol{d_0}$, which can be estimated from the image gradient, i.e. $p = \{(x_0, y_0), \boldsymbol{d_0}, \{a_i\}_{i=i}^{N}\}$, with $a_i \in A$, $A = \{\text{"Left"},\text{"Straight"},\text{"Right"}\}$ an ordered set of moves. In order to measure, or indeed control the path trend, a path can be modelled as a Markov process that gives in output a move $a_i \in A$, for each step i. Exploiting Bayes rule, the joint probability of a path can be written as:

$$P(a_{i+1}, a_i, \dots, a_1) = P(a_{i+1}|a_i, \dots, a_1)\, P(a_i|a_{i-1}, \dots, a_1) \cdots \quad \cdots P(a_2|a_1)P(a_1) \tag{5.1}$$

Then, considering a simple first order Markov process $P(a_{i+1}|a_i, \dots, a_1) = P(a_{i+1}|a_i)$, it is enough to define the conditional probabilities $P(a_{i+1}|a_i), \forall a_i \in A$, to have the joint probability of the whole path (the probability $P(a_1)$ can be considered uniform and ignored because equal for all the paths). Moreover, considering the image $f(x,y)$ as a random field with $z_i = f(x_i,y_i)$, we suppose that it is possible to define the two following probability measures:

$$\begin{aligned} P(f(x_i,y_i) = z_i|H_0) &= P_{H_0}(z_i) \\ P(f(x_i,y_i) = z_i|H_1) &= P_{H_1}(z_i) \end{aligned} \tag{5.2}$$

where H_1 (or H_0) is the hypothesis of having (or not) an edge point at (x_i,y_i). Consequently, a quality measure $Q(p)$ of a path p can be obtained by the following likelihood ratio:

$$Q(p) = \frac{P_{H_1}(z_1, \dots, z_N)}{P_{H_0}(z_1, \dots, z_N)} \tag{5.3}$$

which can be written in the following form if the assumption of statistically independent pixels holds:

$$Q(p) = \frac{\prod_{i=1}^{N} P_{H_1}(z_i)}{\prod_{i=1}^{N} P_{H_0}(z_i)} \tag{5.4}$$

At this point, indicating with $P(p)$ the path joint probability, a complete measure (metric) of a path could be:

$$\Upsilon(p) = Q(p)P(p) = \frac{\prod_{i=1}^{N} P_{H_1}(z_i)}{\prod_{i=1}^{N} P_{H_0}(z_i)} \prod_{i=1}^{N} P(a_i|a_{i-1}) \tag{5.5}$$

which can be compute additively applying the logarithm:

$$\Upsilon(p) = \ln[Q(p)P(p)] = \sum_{i=1}^{N} \left\{ \ln\left[\frac{P_{H_1}(z_i)}{P_{H_0}(z_i)}\right] + \ln[P(a_i|a_{i-1})] \right\} \tag{5.6}$$

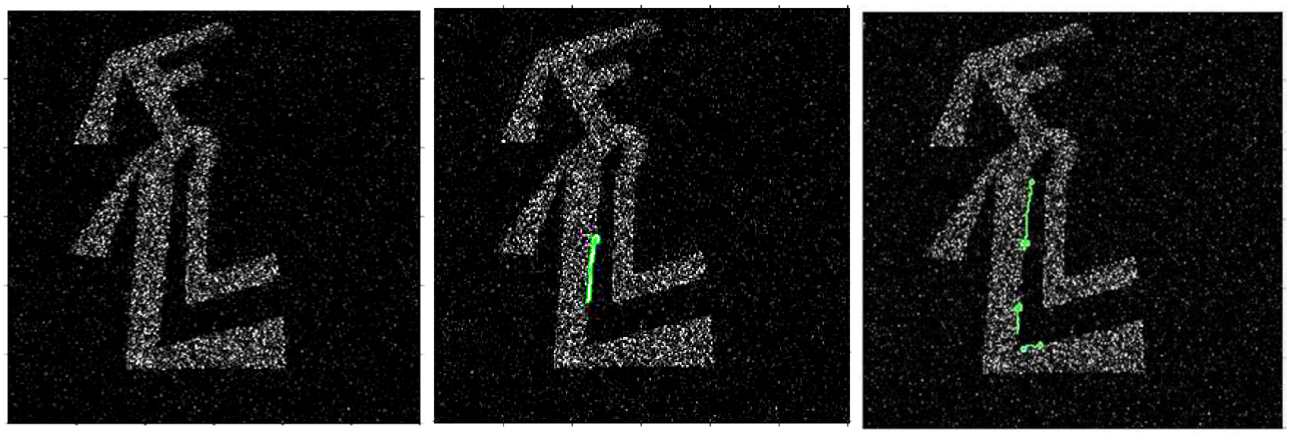
thus the metric at the step $n+1$ can be computed as:

$$\Upsilon(p_{n+1}) = \Upsilon(p_n) + \ln\left[\frac{P_{H_1}(z_{n+1})}{P_{H_0}(z_{n+1})}\right] + \ln[P(a_{n+1}|a_n)] \tag{5.7}$$

In [41] a simple but efficient method to solve the shortest path problem is given. In that method, starting from an edge pixel and given a direction, three neighbours (on the left, straight and on the right) are put in a priority queue with their respective metrics. The queue is ordered by the metric and the best path is always at the top. At each step only the neighbours of the path at the top are explored. Clearly, in this way not all the paths from source to destination are explored, but only those which have more probability to be the best.

5.2 Lacks in the Original Model

The most important drawbacks in this algorithm concern the complete lack in taking in account possible **loops in the paths**. In fact, applying the original algorithm to some simulated images, a lot of loops are generated and no closed boundary is drawn. Let's consider an example. Given the simulated image in Fig. 5.2(a) where pixels are independent samples from a Gamma distribution, the result in applying the previous algorithm to that image are reported in Fig. 5.2 (b), where only the first cycled path is shown (some other paths starting from different points are shown in Fig. 5.2 (d)).



(a) (b) (c)

Fig. 5.2 - Original sequential linking algorithm results. (a) Original image (number of looks equal to 1, background RCS equal to 100, foreground RCS equal to 300). (b) First path found which is superimposed to original image. (c) Subsequent paths found.

Another problem arising from using this algorithm concerns the computation of the likelihood ratio. In fact, to compute the term

$$\lambda_i = \frac{P_{H_1}(z_i)}{P_{H_0}(z_i)} \tag{5.8}$$

the **knowledge of the image distribution under the two hypotheses** is required. Even making easy assumptions on image distribution (e.g. Normal) brings to estimate some parameters of the pdf in question. In fact, whereas $P_{H_0}(z_i)$ could be simply modelled, $P_{H_1}(z_i)$ depends on the strength of the edge, or indeed, on the two different image intensities on the opposite sides of the edge.

Finally, another question can arise, namely how we have to set the **transition probability** to control the path shape. In fact, at the first move it does make sense to select some transition probabilities because the first direction has been estimated from the gradient and these probabilities measure the distance from this estimate. Hence, if the direction estimated for the first pixel (x_i, y_i) was $\hat{\theta}_i$, going straight at the subsequent step would mean to maintain the direction estimated, whereas going either on the left or on the right would mean change that direction of $\pm\pi/4$. From a mathematical point of view it can be seen that after *n* step, the probability to be in a state a_n are no longer tied to those of the initial transition probabilities. Let's see an example. The initial transition probabilities of the previous example (Fig. 5.2) were set to:

$$\begin{array}{cccc} & L & S & R \\ L & 1/2 & 1/4 & 1/4 \\ S & 1/4 & 1/2 & 1/4 \\ R & 1/4 & 1/4 & 1/2 \end{array} \tag{5.9}$$

where the term b_{ij} of the transition probability matrix $\boldsymbol{P}$ in Eq. (5.9) indicates the probability $P(a_{n+1}|a_n)$ to pass from state $a_n = i$ to state $a_{n+1} = j$ at step $n+1$ (in Eq. (5.9) the letters are the initials of "Left", "Straight", "Right", and naturally $\boldsymbol{P}$ does not depend on n). From the Markov theory we know that to find out the probability to be in a state j after n steps, starting from a state i, we have to read the term b_{ij} of the matrix $\boldsymbol{P}^{(\boldsymbol{n})}$ [45]. Moreover, we know that if the matrix is *regular* (i.e. a state j can be reached after exactly n moves starting at any state i), the matrix $\boldsymbol{P}^{(\boldsymbol{n})}$ with $n \to \infty$ tends to a matrix $\boldsymbol{W}$ which has the same vector $\boldsymbol{w}$ for each row. The matrix in Eq. (5.9), even at step $n = 8$ ($\boldsymbol{P}^{(\boldsymbol{8})}$) is:

$$\begin{array}{cccc} & L & S & R \\ L & 1/3 & 1/3 & 1/3 \\ S & 1/3 & 1/3 & 1/3 \\ R & 1/3 & 1/3 & 1/3 \end{array} \tag{5.10}$$

it means that even after *8* steps the probability to be in any state does not depend on the starting state. Hence, from this reasoning becomes clear that simply by setting the transition probabilities is not possible to control the shape of the path.

5.3 Proposed Parametric (LR) Model

Since we can easily estimate the **image gradient** (e.g. after applying an edge detector), we could use these pieces of information to improve the performance of the algorithm and to solve the previous lacks. For what concerns the **loop issue**, we can solve this problem imposing that the absolute path direction does not have to cross the estimated gradient direction orthogonally. In fact, let us indicate $(x_i, y_i)_n$ as the pixel crossed by the path at step n, next we indicate $\theta_{i,n}$ (with $\theta_{i,n} \in \{0, \pi/4, \pi/2, 3\pi/2\}$) as the absolute direction of the path at that pixel (direction formed joining the center of the pixels $(x_j, y_j)_{n-1}$ and $(x_i, y_i)_n$, bringing back the result to $[0, 3\pi/2]$). Then, we set $\hat{\theta}_{i,n}$ as the direction orthogonal to the estimated gradient direction. Avoiding spatial subscript in the notation, a very simple assumption can be done: given a pixel $(x, y)_n$ with an estimate path direction $\hat{\theta}_n$, the probability of a path crossing $(x, y)_n$ with a direction θ_n orthogonal to $\hat{\theta}_n$ has to be zero. In this way we manage to penalize every path which goes parallel to the estimated gradient direction, thus avoiding the birth of undesired loops. Summarizing, considering a first order Markov process where θ_n and $\hat{\theta}_n$ are statistically independent from their respective previous steps, the metric $P(p)$ at step n becomes:

$$P(p) = P(\theta_n, \dots, \theta_0 | \hat{\theta}_n, \cdots, \hat{\theta}_0) \approx P(\theta_n|\hat{\theta}_n) P(\theta_{n-1}|\hat{\theta}_{n-1}) \cdots P(\theta_0|\hat{\theta}_0) \tag{5.11}$$

Now, considering the **probabilities** $P_1 = P(\theta_n = \hat{\theta}_n | \hat{\theta}_n), P_2 = P(\theta_n = \hat{\theta}_n \pm \pi/4 \,|\hat{\theta}_n)$, and $P_3 = P(\theta_n = \hat{\theta}_n \pm \pi/2 \,|\hat{\theta}_n)$, the pdf $P(\theta_n|\hat{\theta}_n)$ can be seen as the symmetric matrix in Tab. 5.1.

		$\boldsymbol{\theta_n}$			
		$\mathbf{0}$	$\boldsymbol{\pi/4}$	$\boldsymbol{\pi/2}$	$\boldsymbol{3\pi/4}$
$\boldsymbol{\hat{\theta}_n}$	$\mathbf{0}$	P_1	$\frac{P_2}{2}$	P_3	$\frac{P_2}{2}$
	$\boldsymbol{\pi/4}$		P_1	$\frac{P_2}{2}$	P_3
	$\boldsymbol{\pi/2}$			P_1	$\frac{P_2}{2}$
	$\boldsymbol{3\pi/4}$				P_1

Tab. 5.1 - Probability density function $P(\theta_n|\hat{\theta}_n)$. Being symmetric only the top-triangle is provided

Furthermore, the matrix in Tab. 5.1 is a circulant matrix. This means that only the first row (or first column) is needed to completely know the matrix because a row (column) can be computed as the circular shift of the previous row (column). Moreover, thanks to the previous exemplificative hypotheses, this model can be easily extended to a generic Markov model of order *n*. In fact, the only consequence that arises from extending this second order model up to order *n* is the need to compute *n* sums at each step instead of only *2*. Anyway, it is worth noting that extending the model up to order *n* has to be done relaxing the condition on forbidden directions (i.e. values of conditional probabilities equal to zero should be avoided). In fact, avoiding going in orthogonal directions considering the past *n*-choices can imply denying following a corner for *n* consecutive steps.

Coming back to the model in Eq. (5.11), some constraints on values to set in $P(\theta_n|\hat{\theta}_n)$ have to be respected in order to stay in the probability domain:

$$\sum_{\theta_n} P(\theta_n|\hat{\theta}_n) = 1 \tag{5.12}$$

so that, since:

$$\sum_{\theta_n} P(\theta_n|\hat{\theta}_n) = P_1 + P_2 + P_3 \tag{5.13}$$

we obtain the following intuitive constraint:

$$P_1 + P_2 + P_3 = 1 \tag{5.14}$$

For example, in our experiments, we selected:

$$\begin{cases} P_1 = 0 \\ P_2 = 1/3 \\ P_3 = 2/3 \end{cases} \tag{5.15}$$

In Fig. 5.3 the result of our algorithm on the simulated image in Fig. 5.2 (a) is given. We can clearly see how our algorithm manages to draw a closed boundary avoiding the birth of loops totally.

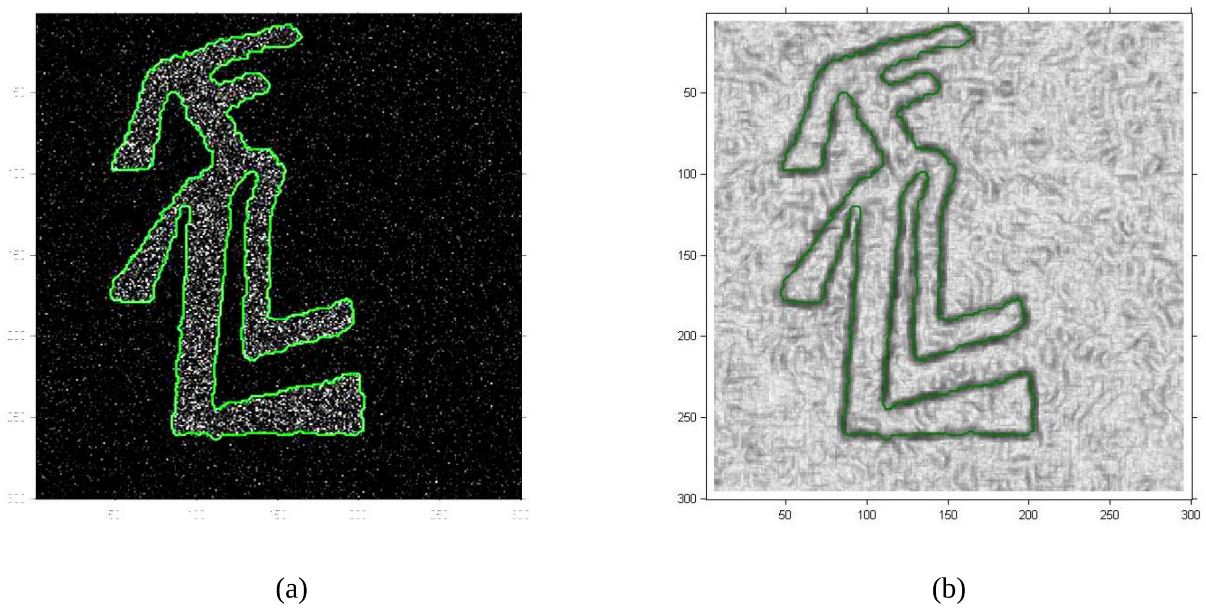


Fig. 5.3 - Result of our algorithm on the simulated image in Fig. 5.2 (a) Result on original image. (b) Result on image filtered with RoA (window size = 11)

Moreover, we should point out that in addition to solving the loop problem, our algorithm introduces **new features** which can drastically improve the performance of the original algorithm. One of these features is the introduction of a new type of priority stack in order to solve the **shortest path** problem better than before. Actually, the idea is to explore not only the best path at the top of the stack, but also other N_{best} paths. This expedient enables to inch the absolute optimum path more likely, keeping the idea of not exploring all possible paths. Furthermore, in order to save computational load, another feature has been added. In fact, in exploring N paths at the same time, two or more paths could cross each other on a pixel. Anyway, following the “trellis” code theory ideas, our algorithm compare which crossing path has the best metric so far and only the best path survives whereas the other paths are deleted. In this way a lot of computational and store load is saved.

In order to clarify the work of our algorithm, in Fig. 5.4 we report a brief example of the first three steps of it (final result is shown in Fig. 5.3) with $N_{best} =3$. In Fig. 5.4 (a) we can see the first step, i.e. starting from the initial pixel three paths are explored (the colours and the numbers shown refer to the path position in the stack). In Fig. 5.4 (b) we can see the second step, where for each of the three previous paths other three pixels are explored (the paths are not order by metric, but the previous paths are deleted from the top of the stack and the new paths are added at the bottom). In Fig. 5.4 (c) we can see the remaining paths, where paths orthogonal to the estimated gradient and crossing paths with worse metric have been eliminated (now the paths are sorted by their metric). In Fig. 5.4 (d) we can see the third step. At this point only the first N_{best} paths ($N_{best} =3$) are exploited (as before the paths are not sorted). In Fig. 5.4 (e) we can see the paths (sorted by their metric) that survive to the rules imposed by our algorithm. Clearly, this procedure will continue in this way until either a closed boundary is yielded or a path goes outside the image borders (or another imposed stopping rule is encountered).

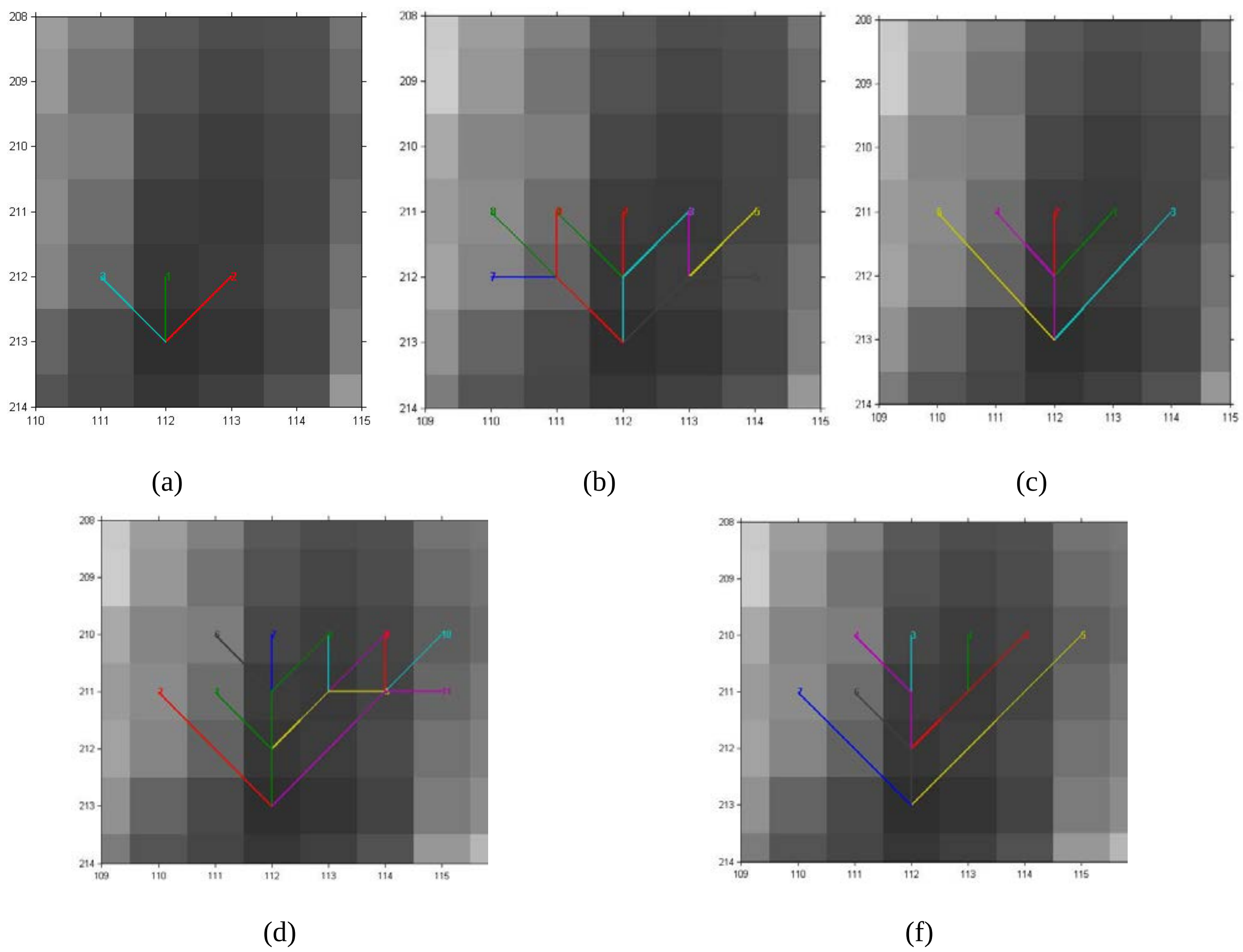


Fig. 5.4 - Illustration of the first three steps of our algorithm. (a) First step with paths sorted by their metric. (b) First part of the second step. (c) Second part of the second step with paths sorted by their metric. (d) First part of the third step. (e) Second part of the third step with paths sorted by their metric

As said before, another problem which arises from using the original algorithm concerns the computation of the likelihood ratio. In fact, to compute the likelihood ratio term, the knowledge of the image distribution under the two hypotheses H_0 and H_1 is required. Usually, at the output of an edge detector, neither $P_{H_0}(z_i)$ nor $P_{H_1}(z_i)$ is exactly known. Nevertheless, using the Ratio of Averages (RoA) edge detector (see Section 3.1.2), which namely computes the ratio between sample means $\bar{I}_1$ and $\bar{I}_2$ on the two sides of an edge, both $P_{H_0}(z_i)$ and $P_{H_1}(z_i)$ can be analytically computed. In particular, given a detector window of size DxD, divided in two parts of N pixels each, and oriented in a direction θ, the detector output $r_{i,\theta}$ at a pixel (x_i, y_i) is given by $r_{i,\theta} = \bar{I}_1/\bar{I}_2$. If Gamma pdf in Eq. (2.12) is assumed, and μ_1 and μ_2 are the intensity means on the respective edge sides, the following equalities hold:

$$\begin{aligned} P_{H_0}(z_i) &= \frac{\Gamma(2NL)}{\Gamma(NL)^2} \frac{\left(r_{i,\theta}\right)^{NL-1}}{\left(1 + r_{i,\theta}\right)^{2NL}} \\ P_{H_1}(z_i) &= \frac{\Gamma(2NL)}{\Gamma(NL)^2} \left(\frac{\mu_1}{\mu_2}\right)^{NL} \frac{\left(r_{i,\theta}\right)^{NL-1}}{\left(\frac{\mu_1}{\mu_2} + r_{i,\theta}\right)^{2NL}} \end{aligned} \tag{5.16}$$

Nevertheless, in order to have only one image as output of the RoA edge detector, for each pixel (x_i, y_i) we can select as output the value r_i belonging to the oriented window that has the maximum probability to have detected an edge in that pixel. Introducing the quantity $R_{i,\theta} = \min\left(r_{i,\theta}, r_{i,\theta}^{-1}\right)$, which is always limited between 0 and 1, the orientation θ^* of such window can be easily computed as:

$$\theta^* = \underset{\theta}{\operatorname{argmin}} R_{i,\theta} \tag{5.17}$$

so that:

$$r_i = r_{i,\theta^*} \tag{5.18}$$

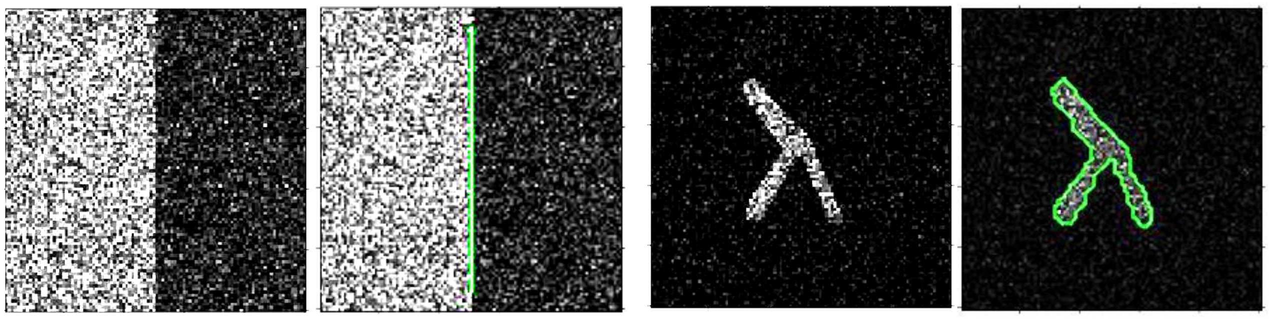
Now, exploiting Eq. (5.16) and substituting μ_1 and μ_2 with their Maximum Likelihood (ML) estimates $\bar{I}_1$ and $\bar{I}_2$, the log-likelihood ratio (LR) $\ln \lambda_i$ can be computed as:

$$\ln \lambda_i = -2NL\ln(2) - NL\ln(r_i) - 2NL\ln(r_i + 1) \tag{5.19}$$

Hence, at the step $n+1$ the metric $\Upsilon(p_{n+1})$ of the path p is:

$$\begin{aligned} \Upsilon(p_{n+1}) &= \Upsilon(p_n) + \ln\left[\frac{P_{H_1}(z_{n+1})}{P_{H_0}(z_{n+1})}\right] + \ln[P(a_{n+1}|a_n)] \\ &= \Upsilon(p_n) - 2NL\ln(2) - NL\ln(r_i) - 2NL\ln(r_i+1) + \ln[P(a_{n+1}|a_n)] \end{aligned} \tag{5.20}$$

In Fig. 5.5 some preliminary results of the proposed parametric (LR) algorithm are reported.


(a) (b)

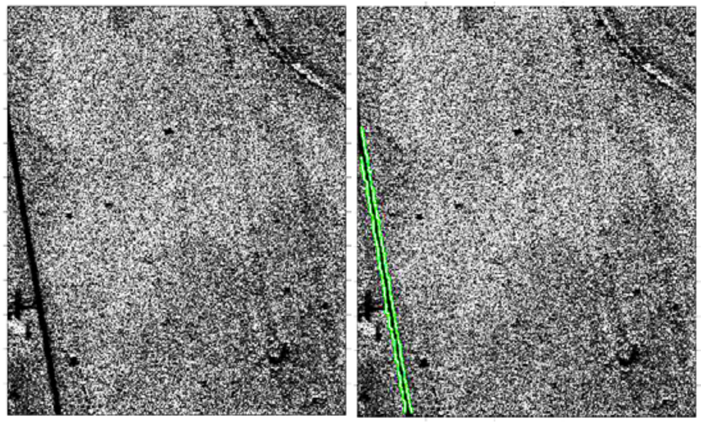

(c)

Fig. 5.5 - Results of parametric (LR) algorithm.
(a) Simulated image (Gamma sample, RCS ratio is 3, number of looks is 1).
(b) Simulated image (Gamma sample, RCS ratio is 5, number of looks is 3).
(c) Real image (MSTAR "HB06173").

5.4 Proposed Non-Parametric (Phy) Model

In active contour methods [46]-[48], the functional to minimize, which drives the evolution of the curve C, parameterized as $\boldsymbol{r}(t) = (x(t), y(t))$, is usually defined in the following way:

$$E = E_{\text{int}} + E_{\text{ext}} = \int_{\mathbb{R}} \left[\alpha|\boldsymbol{r}'| + \beta|\boldsymbol{r}''|\right]dt + \int_{C} f(\nabla I)dC \tag{5.21}$$

The term E_{int} assures a certain regularity of the curve, as the term $P(p)$ in SEL metric does. Instead, the term E_{ext}, which is a function (reverse proportional) of the image gradient module $f(\nabla I) \sim 1/|\nabla I|$, takes the role of LR term in measuring the quality of the path. If the image is presupposed to be differentiable, a general edge detector that estimates ∇I can be used applying Eq. (5.21) directly as a new SEL metric. However, differentiability assumption is nearly never true and diagonal edges have to be estimated separately with an appropriated oriented window. By indicating with $\hat{\boldsymbol{k}}$ and $\hat{\boldsymbol{n}}$ the unit vectors respectively tangential and orthogonal to the curve C, with $\hat{\boldsymbol{v}}_{\mathbf{1}}$and $\hat{\boldsymbol{v}}_{\mathbf{2}}$the unit vectors at $\pm 45°$ from $\hat{\boldsymbol{n}}$, the term E_{ext} in Eq. (5.21) can be rewritten as:

$$\begin{aligned} E_{\text{ext}} = & \int_C f_1(\nabla I \cdot \hat{\boldsymbol{k}})dC + \int_C f_2(\nabla I \cdot \hat{\boldsymbol{n}})dC \\ & + \int_C f_1(|\nabla I \cdot \hat{\boldsymbol{v}}_{\mathbf{1}}| - |\nabla I \cdot \hat{\boldsymbol{v}}_{\mathbf{2}}|)dC \\ & + \int_C f_2(|\nabla I|)dC \end{aligned} \tag{5.22}$$

where the functions $f_1(\cdot)$ and $f_2(\cdot)$ are:

$$\begin{aligned} f_1(s) &= |s|/(1 + |s|) \\ f_2(s) &= 1/(1 + |s|) \end{aligned} \tag{5.23}$$

The term in Eq. (5.22) manages to account for the orthogonal direction of the gradient with respect to any edge curve and that, in an ideal step edges, the terms $|\nabla I \cdot \hat{\boldsymbol{v}}_{\mathbf{1}}|$ and $|\nabla I \cdot \hat{\boldsymbol{v}}_{\mathbf{2}}|$ are equal. It should be noticed that with Phy metric, which is obtained substituting Eq. (5.22) in Eq. (5.21), there is no need to know the image pdf and it can be applied to any image. Moreover, unlike SEL and LR metric, which may assume negative values, Phy metric is positive so that the existence of the shortest path between any two points is assured.

5.5 Improved Parametric (LR) Model

In Section 5.3 we mapped in the likelihood ratio term the ratio between the probability to have an edge and the probability not to have an edge in a certain pixel. Following the reasoning behind the Eq. (5.22), we could use, at each step, the information coming from the edge detector windows at every orientations. In fact, an improved version of the likelihood ratio term should have the probability to have an edge with direction $\hat{\boldsymbol{k}}$ and the probability not to have an edge with direction $\hat{\boldsymbol{k}}$ in a certain pixel. Consequently, differently from Eq. (5.8) where we computed the ratio between the probability to have an edge and the probability not to have an edge in a certain pixel, in this case the numerator and denominator of the LR term has to account the probability to have an edge with a direction different from $\hat{\boldsymbol{k}}$.

In this last case, the General Likelihood Ratio, considering the usual four directions of the filtering windows, is:

$$\lambda_i = \frac{P_{H_1}(r_{i,\theta_1}, r_{i,\theta_2}, r_{i,\theta_3}, r_{i,\theta_4})}{P_{H_0}(r_{i,\theta_1}, r_{i,\theta_2}, r_{i,\theta_3}, r_{i,\theta_4})} \tag{5.24}$$

Where H_1 (or H_0) is the hypothesis to have (or not) an edge with a direction $\theta_1$at the pixel i, and r_{i,θ_k} is the value of the RoA edge detector when a filtering window of direction θ_k is applied. Now, omitting the dependence from the spatial position i, indicating with $P(E_{\theta_k})$ (or $P(\bar{E}_{\theta_k})$) the probability to have (or not) an edge with direction $\theta_k$at pixel i, and supposing the variables r_{i,θ_k} independent:

$$\lambda_i = \frac{P_{H_{E_{\theta_1}}}(r_{i,\theta_1})P_{H_{E_{\theta_1}}}(r_{i,\theta_2})P_{H_{E_{\theta_1}}}(r_{i,\theta_3})P_{H_{E_{\theta_1}}}(r_{i,\theta_4})}{P_{H_{\bar{E}_{\theta_1}}}(r_{i,\theta_1})P_{H_{\bar{E}_{\theta_1}}}(r_{i,\theta_2})P_{H_{\bar{E}_{\theta_1}}}(r_{i,\theta_3})P_{H_{\bar{E}_{\theta_1}}}(r_{i,\theta_4})} \tag{5.25}$$

Exploiting the result of Section 3.3, and the computation in Appendix K:

$$\begin{aligned}
\lambda_i = {} & 5^4 P(r_i|E_{\theta_\parallel})\big|_{r_i=r_{i,\theta_1}} P(r_i|E_{\theta_\angle})\big|_{r_i=r_{i,\theta_2}} P(r_i|E_{\theta_\angle})\big|_{r_i=r_{i,\theta_4}} P(r_i|E_{\theta_\perp})\big|_{r_i=r_{i,\theta_3}} \cdot \\
& \cdot \left\{P(r_i|E_{\theta_\perp}) + 2P(r_i|E_{\theta_\angle}) + P(r_i|\bar{E})\big|_{r_i=r_{i,\theta_1}}\right\}^{-1} \cdot \\
& \cdot \left\{P(r_i|E_{\theta_\parallel}) + P(r_i|E_{\theta_\perp}) + P(r_i|E_{\theta_\angle}) + P(r_i|\bar{E})\big|_{r_i=r_{i,\theta_2}}\right\}^{-1} \cdot \\
& \cdot \left\{P(r_i|E_{\theta_\parallel}) + P(r_i|E_{\theta_\perp}) + P(r_i|E_{\theta_\angle}) + P(r_i|\bar{E})\big|_{r_i=r_{i,\theta_4}}\right\}^{-1} \cdot \\
& \cdot \left\{P(r_i|E_{\theta_\parallel}) + 2P(r_i|E_{\theta_\angle}) + P(r_i|\bar{E})\big|_{r_i=r_{i,\theta_3}}\right\}^{-1}
\end{aligned} \tag{5.26}$$

where $P(r_i|E_{\theta_\parallel})$, $P(r_i|E_{\theta_\perp})$, $P(r_i|E_{\theta_\angle})$, and $P(r_i|\bar{E})$ can be completely known allowing a little approximation on the real pdf (otherwise it cannot be computed in a closed form), see Section 3.3.

5.6 On the Optimal Path Search

Authors in [41]-[44] maintain that their problem of linking edges together can be mapped in a shortest path problem (SPP) which can be solved by the most common SPP algorithms as Dijkastra or his generalization[8] A*[49]. Actually, this is not true. In fact, the previous algorithms assure of finding the shortest path only if some conditions hold. In order to deepen these concepts, some notation is introduced. A graph $G(N, L)$ is defined as a set of nodes $\{n_i\} \in N$ and a set of arcs (or links) $\{l_{ij}\} \in L$, where l_{ij} is the arc which links the node n_i and the node n_j. On each arc l_{ij} a cost (or metric) written as $c(l_{ij})$ is defined and, moreover, it has to follow some constraints. In fact, in order to find the shortest path with the algorithms Dijkastra or A*, the following statements have to be true:

$$c(l_{ij}) \geq 0, \quad \forall\, l_{ij} \in L \tag{5.27}$$

and

$$c(l_{ij}) \leq c(l_{ik}) + c(l_{kj}), \quad \forall\, n_i,\, n_j,\, n_k \in N \tag{5.28}$$

Practically speaking, Eq. (5.27) and (5.28) assure that **loops are completely avoided** during the SPP procedure. Unfortunately, using a cost which respects Eq. (5.27) and (5.28) to link edges in an image, (in order to apply Dijkastra or A*), the optimal **path tends to be shorter than possible**, i.e. it does not follow the edge boundary but it tries to cross it in order to reach the destination quicker than possible. This fact can be seen in Fig. 5.6 where the optimal path between two pixels in the boundary is shown. In that example the A* algorithm has been used with the metric defined by Eq. (5.22) and exploiting the following information: a path does not go parallel to the gradient of the image. It should be noted that the cost defined in Eq. (5.22) verifies Eq. (5.27) and (5.28).

[8] The algorithm A* is the Dijkastra algorithm where the exploration of neighbour nodes of a just explored node is driven by some *a priori* information, e.g. we can explore the neighbour nodes of the actual best path first, or we can explore neighbour nodes along a certain direction first, because it is known that the target node (or destination) can be reached going in that direction.

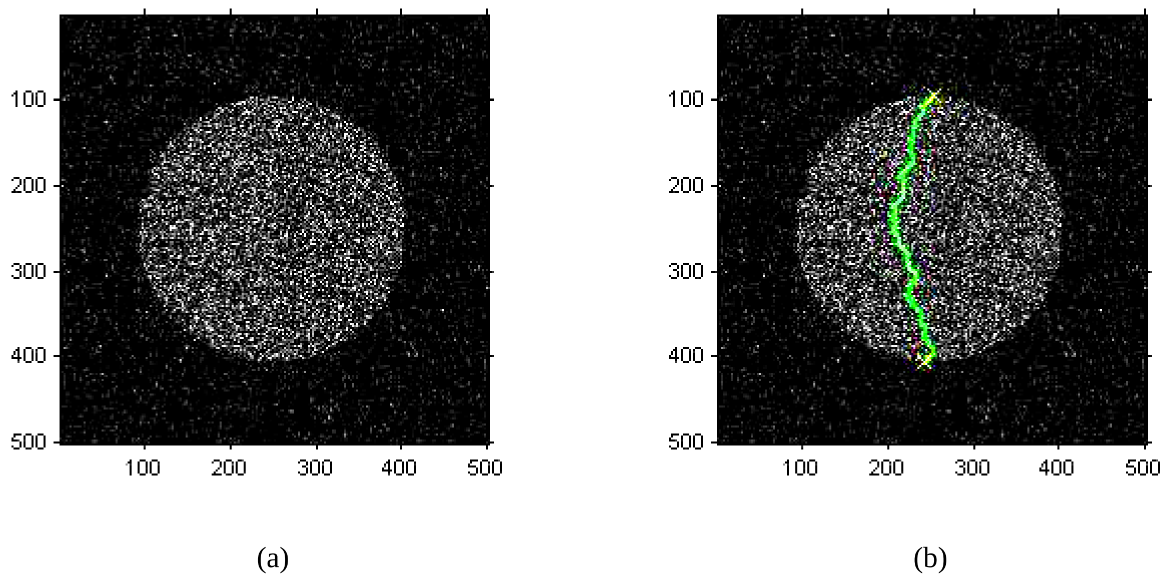
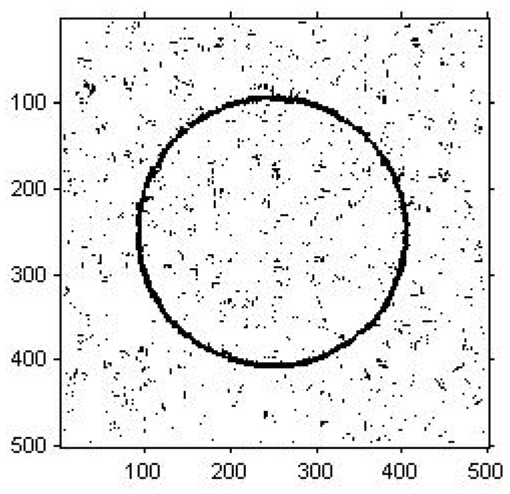


Fig. 5.6 - (a) Simulated image with independent Gamma samples, with number of look equal to 1, RCS of the inner circle equal to 300, and RCS of the background equal to 100. Shortest path between two pixels in the object boundary (signed with a yellow cross). The A* algorithm with metric in Eq. (5.22) has been used

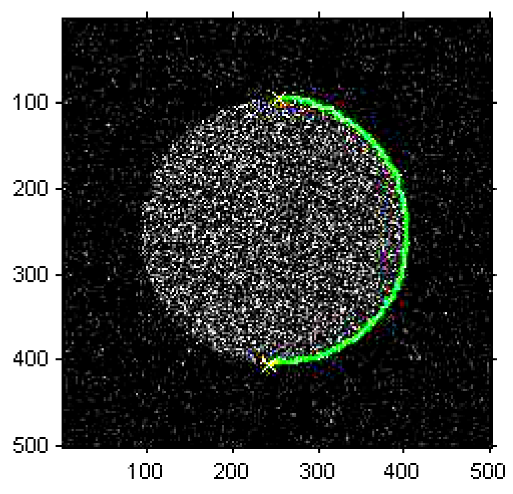
The result shown in Fig. 5.6 is very intuitive. In fact, it does not matter if the path goes through pixels which have a bad metric, at the end, when all *feasible* paths are evaluated, the path which links the least number of nodes will have the lowest metric (the shortest path is not the segment between the two pixels because that path cross some intermediate pixels with a direction parallel to the estimated gradient, i.e. it is not feasible). The reason why the best path tries to reach the destination quicker than possible is due to the validity of Eq. (5.27), i.e. linking more pixels can leave the metric unvaried only for a path which follows exactly the edge boundary. Nevertheless, that path does not exist in a discrete set where directions can only vary as multiple of $\pi/4$. As consequence, raising the number of nodes in a path implies, nearly always, raising the path metric. Nevertheless, we can move in several ways to solve this type of problem.

The first simple way it is of using a **heuristic procedure** (as SEL algorithm shown in Section 5.1) which enable us to explore, at each step, only the pixels near the actual best path, so that the best path remains always near the pixels with low metric. Naturally, we completely lose the optimality of the search.

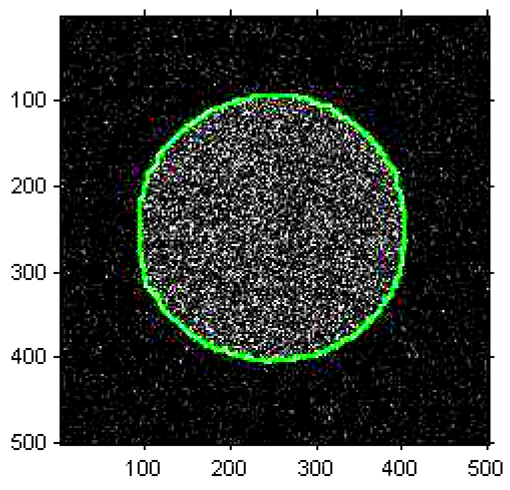
The second, it is that of **reducing the number of feasible paths** to allow the propagation of the shortest path only around the boundaries. It should be noted that this solution is very simple to applying. In fact, it suffices to define an *admissible region* around the edges in some way, e.g. applying a very weak threshold to the edge detector output or using the method in [42], where the zero-crossing of the Laplacian of Gaussian (LoG) operator with a certain extension is used. Anyway, this last solution is not always valid because the found region it is not defined only around boundaries. Nevertheless, in [42] the admissible region is used only as stop criterion: when the actual best path exits from the admissible region, the algorithm stops the search and returns the actual best path. Instead, in the optimum algorithm, the admissible region is used as a channel which drives the shortest path propagation. Applying at the A* algorithm an admissible region found as thresholding of the RoA output (Fig. 5.7(a)), we manage to find the optimal path without any problem (Fig. 5.7(b)-(d)).



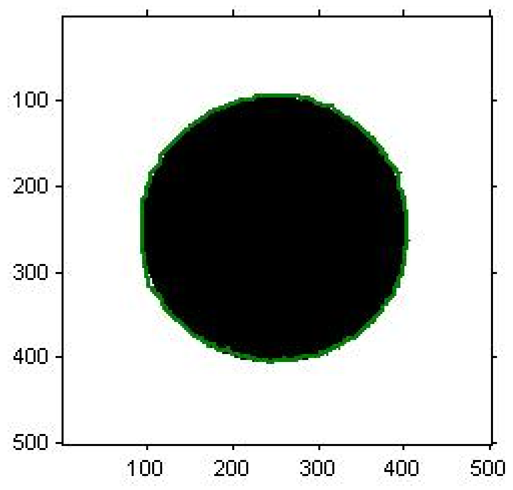
(a)



(b)



(c)



(d)

Fig. 5.7 - (a) Admissible region (in black) as thresholding of RoA edge detector output of the image in Fig. 5.6. (b) Shortest path between two pixels in the object boundary (signed with a yellow cross). The A* algorithm with metric in Eq. (5.22) has been used. (c) Optimum closed path. (d) Optimum closed path superimposed to the binary mask of the circle

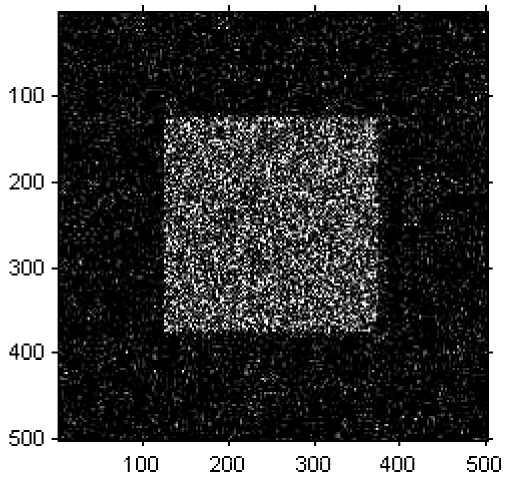
The third way concerns the use of graphs where **negative costs** between nodes are allowed. In this way, if we could assign negative costs at arcs between pixels on boundaries and positive costs otherwise (e.g. with the metric in Eq. (5.20)), we would avoid paths as that in Fig. 5.6 (b), without using any admissible region. Nevertheless, using negative costs, we have no guaranties about the absence of loops, and thus, we need to solve the loop problem in some way[9]. It is worth noting that if no loops with negative costs exist in a graph, it is always possible to transform a graph with negative costs to another graph with positive costs which shares the same shortest paths of the former from a source node *s* to any node *n* [51].

[9] The Bellman-Ford-Moore (BFM) algorithm [50] can be used to solve the SPP problem when costs are even negative. Note that when costs are allowed to be negative, the shortest path between a source *s* and a node *n* exists only if there are no cycles with negative cost from *s* to *n*. Moreover, BFM algorithm suggests a specified procedure to find negative cycles and, in that case, stopping the search getting back an error message.

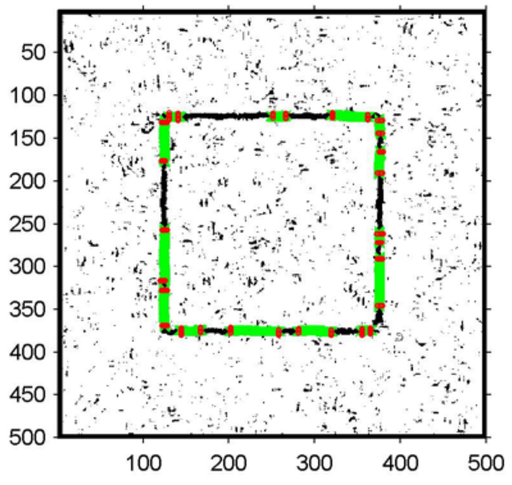
Another problem that arises in using the optimal path concerns the automatic implementation of the optimal algorithm. In fact, despite SEL algorithm, **optimum path** algorithm is not so simple to implement. The main **questions** concern:

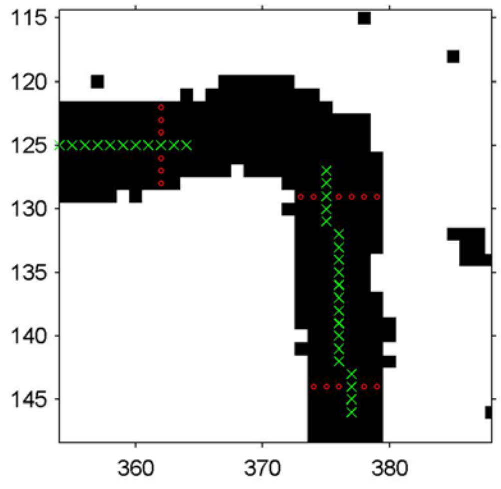
- setting certain stopping rules;
- avoiding waste of time in waiting useless shortest path propagation flows;
- solving the loops when the metric is negative (e.g. metric in Eq. (5.20)).

If the metric is always positive, e.g. **Phy metric**, we can use the admissible region to avoid wasting time. Moreover, we can put manmade "wall" near the endpoints in order to stop the algorithm when the flow hits the wall (Fig. 5.8).



(a)





(b) (c)

Fig. 5.8 - (a) Simulated image with independent Gamma samples, with number of look equal to 1, RCS of the inner square equal to 300, and RCS of the background equal to 100. (b) Binary image of the admissible region on which the output of the NMS and DT are superimposed as green crosses and the walls as red. (b) Magnification of the result at the top-right corner

Nevertheless, **if the admissible region is too extended**, the problem previously discussed can give raise, i.e.: the **shortest path does not follow the edge boundary, but it tries to cross it** in order to reach the destination quicker than possible. If we set the admissible region very tight, we can have no path between endpoints and no general rule exists to reconstruct the edge within the admissible region if no destination has been reached. Furthermore, if negative metric is allowed, the problem becomes much more complicated.

For all these problems an automatic implementation of the optimal path algorithm is very difficult in general. Anyway, in order to evaluate the performance of the SEL algorithm, we have implemented it for the test images presented in this document, even for the LR metric. In our implementation we have used the BFM algorithm with admissible region (computed as thresholding of an edge detector output), manmade walls (built automatically at distance 2 pixels from each endpoint), and some constraints on the direction of the propagation flow. In particular if a path arrives at a pixel with a direction parallel to the gradient direction estimated at that pixel, then the path is not accepted. Furthermore, in order to lower the probability of negative cycles, in addition to the admissible region, a new path which arrives to a pixel with a direction greater than $\pi/4$ compared to the previous best path is discarded.

5.7 Results

Some experiments on simulated and real images have been carried out in order to measure the performance of the previous linking algorithms. We have compared the SEL algorithm with the optimum path when both use the same metric, and then, varying the metric. Since SEL is a linking algorithm, we have applied it at the edge map provided by NMS and DT procedures. The two thresholds of the DT block have been chosen in order to yield a more complete edge map than possible, avoiding any “spur” edge. To evaluate the performance of the algorithms, we have used indexes of quality as Pratt Figure of Merit (FoM) in Eq. (4.79), Completeness (computed as the ratio between reconstructed and real number of pixel of the boundary), and the mean distance (MD) between the boundary truth mask and the reconstructed one.

The experiments concern the evaluation of SEL and optimum path when the metric in Eq. (5.22) (which we call “Physical metric” or “Phy”,), or Eq. (5.20) (which we call “Likelihood Ratio metric”or “LR”), is applied at both algorithms. The E_{int} term of the Physical metric has been put to zero so that not to account any a priori information on the object shape, as LR metric. We want to evaluate performance when both linear and curvilinear edges are involved. Hence, two simulated images, one with a circle and one with a square, have been used as test. The image with a circle is that in Fig. 5.6(a), whereas the image with the square is shown in Fig. 5.8(a). The results of these experiments are shown in Tab. 5.2

		Circle			**Square**		
		Comp.	***MD***	***FOM***	***Comp.***	***MD***	***FOM***
Phy metric	***SEL***	1	0.85	0.96	1	0.65	0.98
	Optimum	1	0.76	0.96	1	0.69	0.98
LR metric	***SEL***	1	0.72	0.96	1	0.62	0.99
	Optimum	1	0.72	0.97	1	0.69	0.99

Tab. 5.2 - Table of results.

As we can see from in Tab. 5.2, the results are surprising. In fact, they make clear the following happenings:

- the results are all excellent (maximum Completeness, high FOM, and low MD);
- the metric LR yields the best results;
- SEL indexes of performances are always near those of the optimum path and they can even cross the latter on linear borders.

Nevertheless, the last fact has to be relaxed. In fact, before computing the SEL indexes we have applied two morphological operators to SEL results. The explanation of these further operations is provided in Fig. 5.9. As we can see in that figure, some “double borders” can appear with the SEL algorithm. Anyway, after a morphological closing by a disk with radius 3 and a thinning operation the problem is completely solved. Furthermore, sometimes SEL algorithm follows paths that are far from the object boundary. Nevertheless, even in this case, we can recognize the problem and stop the path growth. In our version of the algorithm we stop the path search when three consecutive pixels of the actual best path are over the admissible region (Fig. 5.9 (c)). For the sake of completeness we report also the execution time of the previous algorithms in Tab. 5.3.

		Circle	**Square**
		Time (s)	***Time (s)***
Phy metric	***SEL***	1.21	1.11
	Optimum	14.53	5.06
LR metric	***SEL***	1.16	1.08
	Optimum	16.54	4.86

Tab. 5.3 - Algorithm execution times.

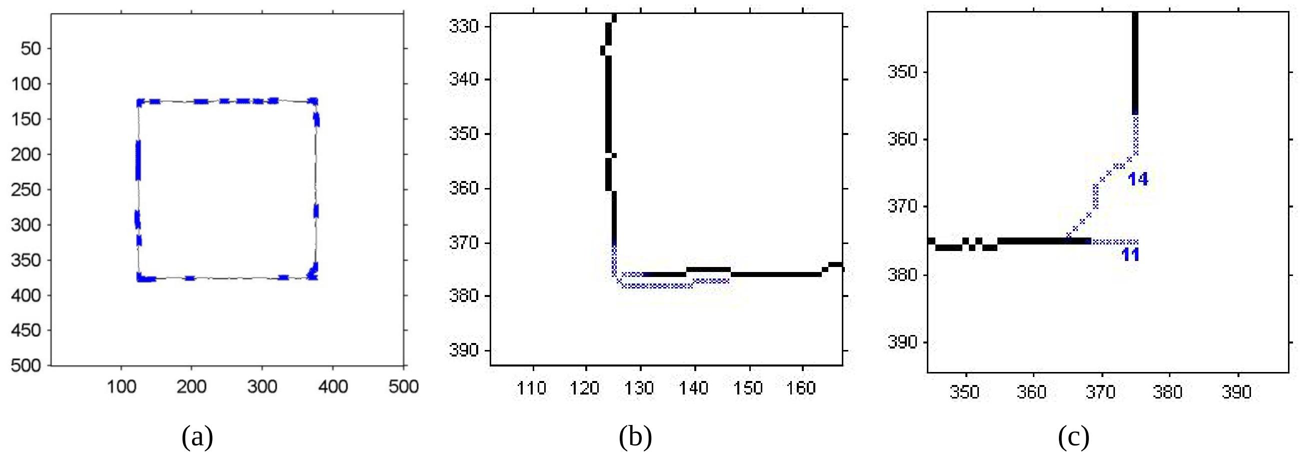


Fig. 5.9 - Result of SEL with metric LR on "Square" image. The black pixels are the output of the non-maxima suppression and double thresholding block; the blue crosses are the borders reconstructed with SEL. (a) Final result where blue crosses are shown bigger than black pixels for display purpose. (b) Magnification of the final results which shows the "double border" problem. (c) Path indicated as 11 has been stopped because three consecutive pixels were out of the admissible region

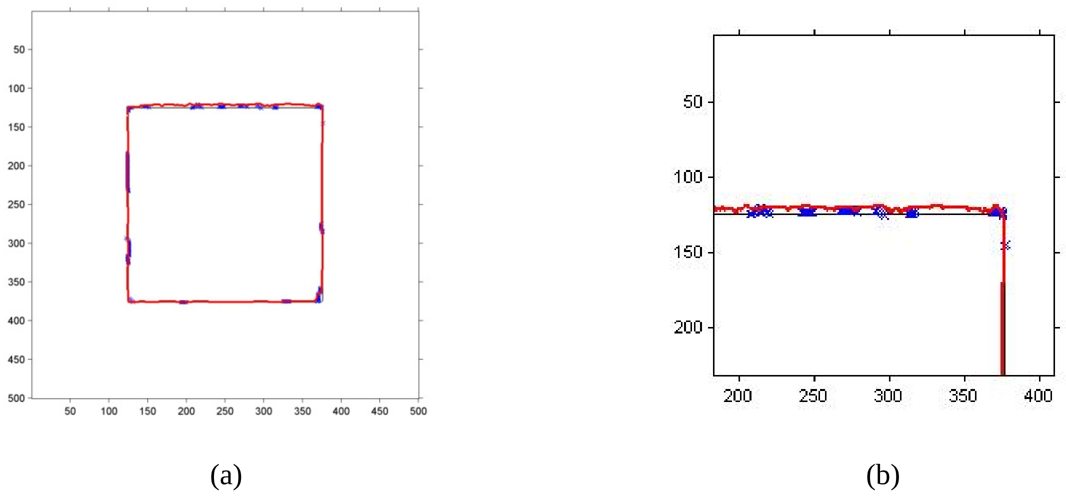
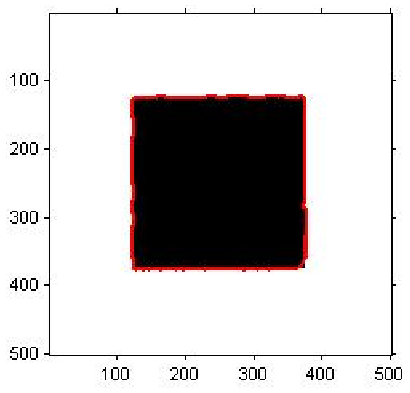
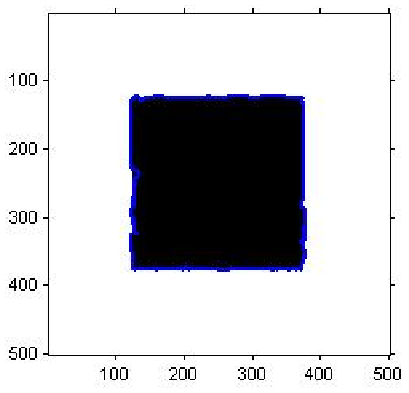


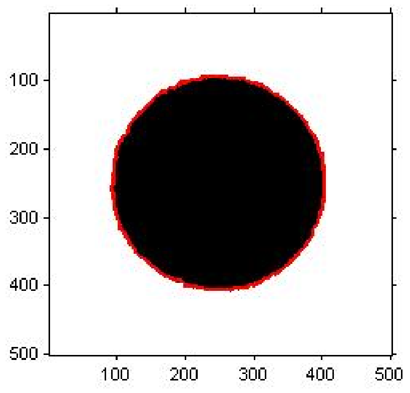
Fig. 5.10 - Result of the optimum path on "Square" image with metric LR. The black pixels indicate the truth mask of the border; the blue crosses are the borders reconstructed with optimum path using endpoints provided by non-maxima suppression and double thresholding procedures; the red line shows the optimum path of the whole border. (a) Final result where blue crosses are shown bigger than black pixels for display purpose. (b) Magnification of the final results which shows how blue crosses are nearer the black pixels

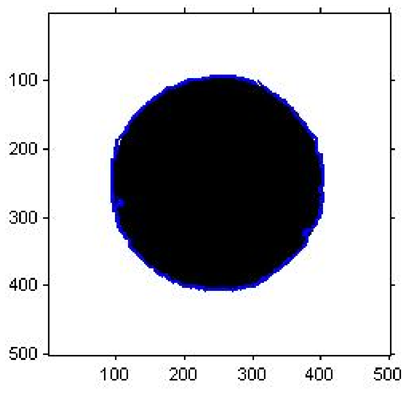




(a) (b)

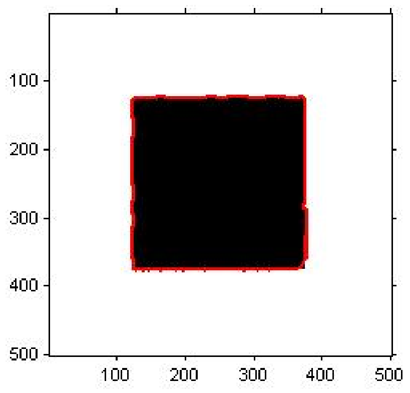
Fig. 5.11 - Binary mask of the square object on which is superimposed the boundary extracted with SEL algorithm. (a) LR metric. (b) Phy metric

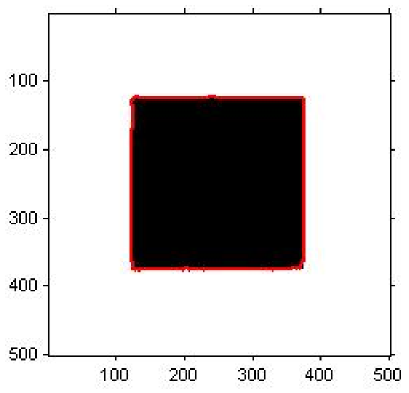




(a) (b)

Fig. 5.12 - Binary mask of the circle object on which is superimposed the boundary extracted with SEL algorithm. (a) LR metric. (b) Phy metric





(a) (b)

Fig. 5.13 - Comparison between LR and Improved LR metric. The boundary extracted with SEL algorithm is superimposed on binary mask of the square object. (a) LR metric. (b) Improved LR metric.

It is interesting noting the difference in applying the previous algorithms at the output of the NMS and DT block (i.e. limiting the application of the algorithms at the endpoints of the binary image) or, instead, in using the previous algorithm trying to draw the whole border entirely. As shown in Fig. 5.10, using the information provided by non-maxima suppression and double thresholding we manage to stay nearer the real border easier.

Even though the quality indexes for LR and Phy metric are always near each other, we can better appreciate the difference between these two metrics visually. In Fig. 5.11 and Fig. 5.12 results of this comparison are reported.

As we can see, even though both boundaries are near the true one, the LR metric enable to draw a better profile without any ripple. This result is very intuitive, in fact, despite Phy metric, LR metric takes advantage of the knowledge about the probability distribution of the pixels filtered with RoA edge detector. Nevertheless, as said at the end of Section 5.5, LR metric can be further improved by information of all edge detector windows. Practically speaking, the improved LR in Eq.(5.26) uses information which comes from the four directions jointly, whereas the LR in Eq. (5.20) uses the information which comes from the most probable direction. The improvements in the final performance can be seen visually in Fig. 5.13 where we have applied this metric to the SEL algorithm on the image with the square.