A local real-time bar detector based on the multiscale Radon transform

Ricardo Oliva-García[a, b], Óscar Gómez-Cárdenes[a], José G. Marichal-Hernández[a,*], Javier Martín-Hernández[b], Jonas Phillip-Lüke[a], and José M. Rodríguez-Ramos[a,b]

[a]Universidad de La Laguna, Industrial Engineering Department, ETSI, La Laguna, Spain, 38200
[b]Wooptix S.L., Av. Trinidad, 61, La Laguna, Spain, 38204

ABSTRACT

We propose a local bar-shaped structure detector that works in real time on high-resolution images. It is based on the Radon transform. Specifically in the muti-scale variant, which is especially fast because it works in integer mathematics and does not use interpolation. The Radon transform conventionally works on the whole image, and not locally. In this paper we describe how by stopping at the early stages of the Radon transform we are able to locate structures locally. We also provide an evaluation of the performance of the algorithm running on the CPU, GPU and DSP of mobile devices to process at acquisition time the images coming from the device's camera.

Keywords: Bar detector, Computer vision algorithms, Radon Transform, Real-time implementations, Code bar detector, Local detector

https://doi.org/10.1117/12.2617411
Ricardo Oliva-García, Óscar Gómez-Cárdenes, José G. Marichal-Hernandez, Javier Martín-Hernández, Jonas P. Lüke, and José M. Rodríguez-Ramos "A local real-time bar detector based on the multiscale Radon transform", Proc. SPIE 12102, Real-Time Image Processing and Deep Learning 2022, 121020A (27 May 2022) *Contact author, jmariher@ull.edu.es

1. INTRODUCTION

Bar detection in images is usually motivated by its use in one-dimensional barcode readers.[1] Barcodes were originally designed to be read by hand-held laser input devices, but these have become obsolete with the widespread adoption of smart mobile devices with cameras. In the first scenario, a user was expected to locate the codes and align the scanline of the scanner with the major axis of the code. On mobile devices, in order to speed up the reading process, it is advisable that the code localization phase imposes as few restrictions as possible, i.e. that it allows the

codes to be located regardless of the position, scale, rotation or even perspective projection with which they are observed by the device's camera. It is also intended that regardless of the specific symbologies and the number of codes, all are located with sufficient accuracy to determine a scanline through them for decoding. The task of precise alignment previously performed by the human user of the scanner will now be undertaken with computer vision techniques, allowing imprecise aiming and the reading of multiple codes simultaneously.

Our objective in this contribution –although it also serves for the location of linear barcodes– goes further and aims to locally characterize each small region (4, 8, 16, ... pixel environments) with two labels: the degree of existence of bars in that locality, and, if applicable, the orientation of the same. Note that for our method to be able to read barcodes, this localization phase should be complemented by a thresholding and clustering phase, and a decoding phase. That is not our goal in this contribution, but it is our goal to perform local detection of bar structures in as short a time as possible in order to be able to process high-resolution images in video acquisition time.

1.1 State of the art review

In order to locate barcodes, methods based on geometry, morphology, artificial intelligence (AI), and hybridization from those categories have been designed. Our proposal will be framed in the methods based on geometry, so we will focus on analyzing the proposals in this category, from which we can extract a starting point. In the conclusions, however, we will also compare with other methods, including those proposed in recent years based on artificial intelligence.

Among the geometry-based methods, the proposals of Zamberletti *et al.*[2] and Creusot *et al.*[3] stand out. Zamberletti uses the Hough transform to determine a histogram of line orientations, after applying an edge detector filter. For the detection to be effective, the transformed space must be studied in chunks. Finally, the method is hybridized with the AI category by checking the output of the zonal transform with a neural network to determine the existence of codes and their bounding-boxes. Creusot uses the Maximal Stable Extremal Region (MSER) detector, a quasi-linear complexity method, initially designed for stable region segmentation for stereo matching. Each segment at the MSER output may or may not correspond to an isolated bar, so a geometrical check is performed, and the normal to its centroid is determined from the bars that exceed it. Hough is applied to the set of all these normals to determine the barcode bisector, an imaginary line that crosses all the bars of a code, which allows their grouping and subsequent decoding.

In both methods, although with different purposes, the Hough[4] transform is applied, which performs the line integral over the entire domain of study to determine if there are lines and with what orientation and at what point of the outer edge they cut. The Hough transform is a particular implementation, with an explicit voting procedure, of the Radon transform. Among the variants of Radon transform we will work upon the discrete multiscale algorithm.[5]

One of the problems with these line integrals, which is highlighted in the first column of figure 1, is that the integrals are not local, but are performed over the entire image. One way to make it local is to piece the image and repeatedly call the transform over each piece. In the methods that operate in this way, in order to function in real time, they try to ensure that the

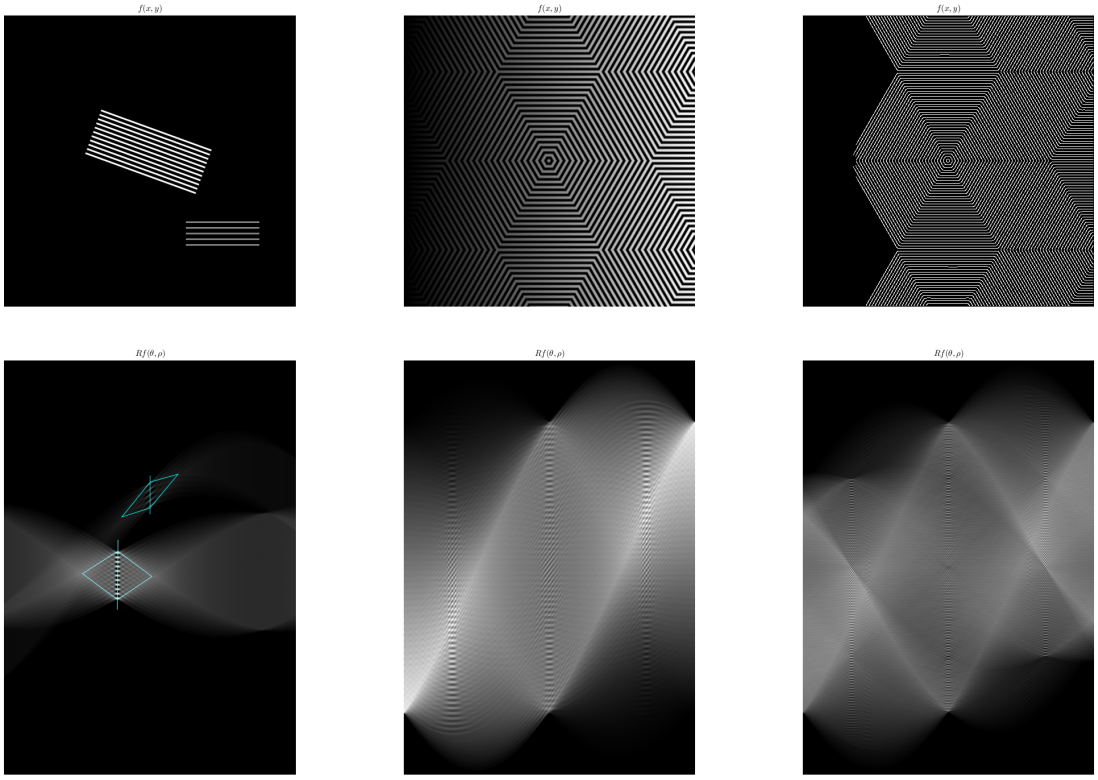

Figure 1: Radon transform applied to different images containing bar-shaped structures. From left to right: a simple example, a complicated one, the edges of the previous.

pieces are not numerous, and therefore not precisely small. This may be sufficient to determine the existence of barcodes in an area, but not to obtain an intensity and orientation value per pixel or small grouping of pixels.

The closest we find to truly local methods are methods based on the study of gradients for edge detection.[6] This can be a good indicator of areas containing barcodes, but they are computationally expensive, specially if the bar orientation is required to be precise, which needs larger kernels. They are usually used only in a preliminary thresholding phase which is then studied with Hough to determine the orientation.

There do not seem to be any specifically designed methods for locating bars locally, accurately, and in quasilinear time complexity.

Returning to the figure 1, in the Radon transform of the first example, it is marked on cyan the footprint left in the transformed space by a grouping of bars in the image space. The vertical lines, coinciding with one of the axes of each rhomboid, mark the orientation of the bars, while the two vertices of the other axis determine the bounding-box of each grouping of bars. Even in a synthetic uncluttered example without noise or background, it is clear that Radon –or Hough, for that matter–, allow, but are not optimal, to detect bars. In addition, an example of a complex bar structure has been included, in whose transformed space it is practically impossible to discern anything. In the rightmost column an edge detector, Canny,

has been applied to convert the bars of the complicated example into lines, but this does not provide an advantage. If several methods use Hough succesfully, it is because they either slice the image, or they have previously isolated the areas potentially containing barcodes and then study each area separately.

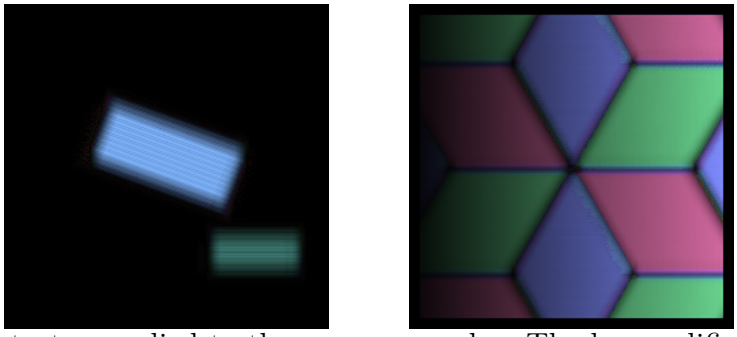

Figure 2: Our bar detector applied to the same examples. The hue codifies the slope detected.

With figure 2 we want instead to exemplify what our method sets out to do, and in fact achieves. To calculate for each location in the image an indicator of the existence of bar-shaped structure in that zone and its orientation, where present. To visualize this joint output we have encoded the orientation in the hue channel, and the intensity of detection in the value channel, of an HSV image, with constant saturation. Each area of the input image with bars of matching orientation should result in an area of homogeneous color. In areas without bars or without a clear orientation, e.g. because of a confluence of different bars, the intensity will be zero or low. This is a dense result, where only the boundaries of the input image remain unlabeled.

1.2 What defines a bar?

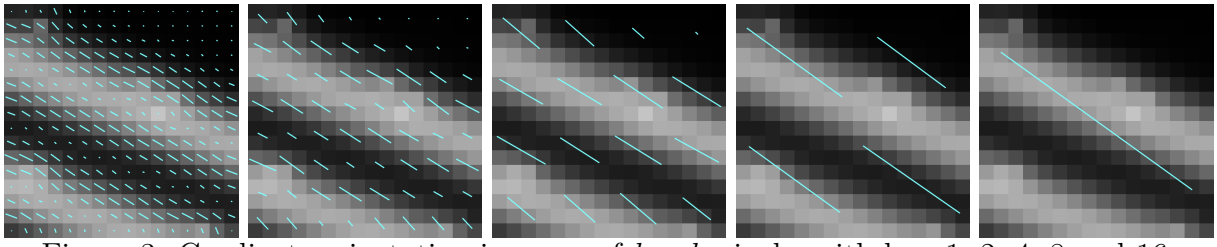

Figure 3: Gradients orientation in areas of $k \times k$ pixels, with k = 1, 2, 4, 8 and 16.

In a bar zone the intensity changes occur in a preferential orientation, i.e. if we start from any point the intensity will remain constant if we move in a certain direction, because we remain inside that bar. And the intensity changes will be maximum if we move in the orthogonal direction, because we are crossing through the bars. The statement, and preferential direction, should remain constant in the zone, as shown in figure 3.

2. DRT AS THE SEED OF A BAR DETECTOR

2.1 Discrete Radon Transform

Radon transform[7] dates back a century and is defined as follow:

$$\Re f(\theta, \rho) = \iint f(x, y)\ \delta(x\ cos\ \theta + y\ sin\ \theta - \rho)\ \mathrm{d}x\ \mathrm{d}y, \tag{1}$$

or equivalently using the absolute *s*lope and *d*isplacement form, with $|s| < 1$,

$$\Re_{|\theta| \leq \frac{\pi}{2}} f(s, d) = \int f(u,\ u\ s + d)\ \mathrm{d}u, \tag{2}$$

$$\Re_{|\theta| \geq \frac{\pi}{2}} f(s, d) = \int f(u\ s + d,\ u)\ \mathrm{d}u. \tag{3}$$

The Discrete Radon Transform, DRT, dates back to the late 1990s,[5,8,9] and was designed to compute sums of data placed on *loose* discrete lines, without interpolation. The discrete lines that the method defines are a sequence of integer positions of approximately constant slope.

The discrete lines are expressed in a intercept-slope form, with slope between 0 and 1, as in $y = x \cdot slope + d$. The algorithm exhibits linearithmic complexity, $O(N \log(N))$, to compute all the sums of such lines crossing an input of size $N \times N$, with N power of two.

DRT basic quadrant algorithm needs to be applied three more times to mirrored and/or rotated versions of the input in order to obtain the results for cases $y = -x \cdot slope - d$, $x = y \cdot slope + d$ and $x = -y \cdot slope - d$, getting a complete discrete *sinogram* covering 180 degrees of projection.

We have chosen DRT precisely because of its multiscale, divide-and-conquer, approach, where to perform the integrals of length 4 it first performs the integrals of length 2, and with those of length 4 it will then perform those of length 8, etc. All these intermediate results, that constitute the partial stages of transformation, will normally be discarded, but for our purposes they will be essential.

2.2 Partial stages of DRT

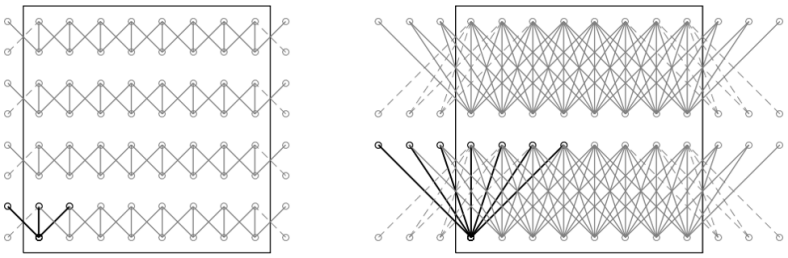

Figure 4: First two stages of computation of vertical lines of DRT in a 8×8 grid. In coarse lines the segments starting from bottom left coordinate. Note that there are 8/2 stripes of length two, in the first stage; and 8/4 stripes of length four, in the second one.

The m-th partial stage of the DRT will have calculated the summations over all line segments crossing image stripes of width 2^m. In a previous contribution[10] we proposed to perform jointly the horizontal line integrals, eq. (2); but separately from the vertical ones, eq. (3). That is, two quadrants of the original formulation performed at a time, for better parallelization. This process is illustrated by figure 4.

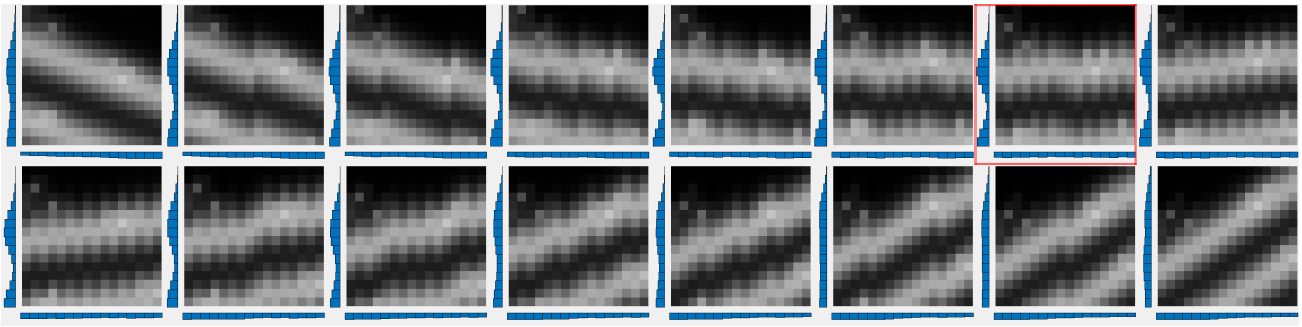

Figure 5: An image region of size 16×16 sheared to reflect how DRT digital lines traverse it with different slopes increasing from left to right, and then downward. The actual DRT values, the summations per row and column, are shown as blue bars.

Such a formulation will also be beneficial in establishing that the DRT partial stages *contain the seed* of a bar detector. In figure 5 a mapping of an image area of size 16×16, containing bars and some noise, has been made attending to the coordinates of the horizontal digital lines that cross through it in DRT at stage $m = 4$. That is, rows within a subfigure show the pixels traversed by horizontal lines of a certain slope and adjacent displacements, $f_s(x, x \cdot s + d)$. Each subfigure, as we move to the right and then downward through the subfigures, correspond to the positive slopes for that size. The unaltered image is shown in the top left subfigure. Note that the positive increments on the y-axis are downward in image coordinates. The figure reveals that lines traverse the input describing a shear transform of angles between 0 and 45 degrees, but without interpolation and keeping the center point static. Indeed, these images are not explicitly generated by the DRT, but they explain its inner workings. What the DRT will compute is the cumulative per horizontal line –a row in these depictions–, which is shown to the left of each subfigure. The DRT would have transformed the 16×16 pixel image input, into the 16×16 accumulated values shown as horizontal blue bars, in a computationally optimal manner, in $m \times N \times N$ operations, in this case $m = 4$. In another pass, consuming another $4 \times N \times N$ operations, the same method, but aplied to the transposed input, will compute the summations through vertical lines, $x = slope \cdot y + d$. In the figure are shown at the bottom of each subfigure in vertical blue bars charts.

The subfigure corresponding to the slope $s = 6$ has been marked in red, since in its horizontal and vertical line integrals occurs the largest variation on one axis and, simultaneously, the smallest on the other. The bar orientation estimation, $\tan^{-1}(6/15)$, is more accurate than the one achieved by the gradient's method suggested in rightmost subfigure 3 and it takes far fewer calculations. The black zone without gradients does not affect now the estimation.

The pseudo-code to perform the computations of sums through horizontal lines up to stage $\log_2(\text{tile\_size})$ of an image I of size $N \times N$ is shown as Algorithm 1. The vertical DRT can be

achieved with that same code, transposing the input.

3. PROPOSAL FOR BAR DETECTOR

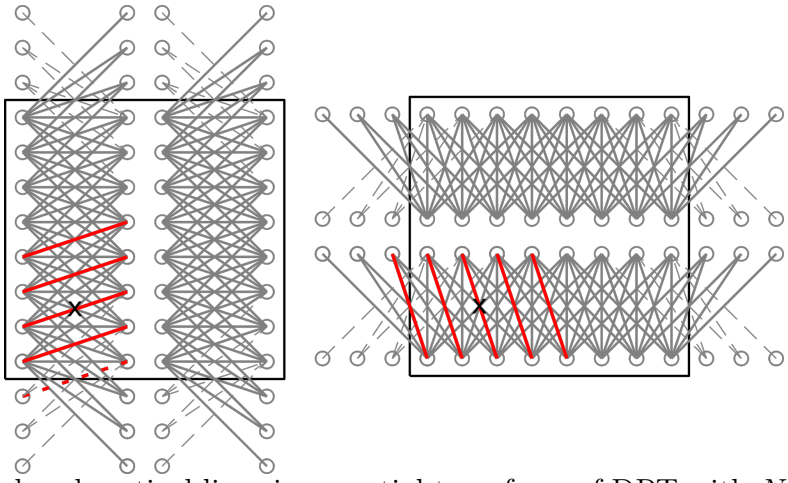

Figure 6: Horizontal and vertical lines in a partial transform of DRT with $N = 8$, $m = 2$. Lines of orthogonal orientation, in red, in the neighborhood of the region whose center is marked with a black cross.

Suppose an input of size $N \times N$, where horizontal and vertical DRT up to stage m has been computed. Figure 6 depicts that there are certain positions (x_k, y_l), with $x_k = 2^{m-1} + k \cdot 2^m$, $k \in [0, \frac{N}{2^m} - 1]$, and similarly for $y_l = 2^{m-1} + l \cdot 2^m$, $l \in [0, \frac{N}{2^m} - 1]$; where it is possible to evaluate whether there is a region of bars of a certain orientation. This can be achieved comparing the set of integrals of horizontal lines with displacements close to y_l in the k-th vertical band, with the set of integrals of vertical lines of slope orthogonal to those in the environment of displacements close to x_k in the l-th horizontal band. In the figure it is highlighted in red the set of lines with $s = 1$ in the neighborhood of (x_0, y_0) marked with a black cross.

Figure 7 shows the 4×4 regions that appear when running up to $m = 3$ the DRT on an input of size 32×32. The four bands of width 8 into which the image is divided are shown in blue, for vertical bands; and gray, for horizontal bands. By zones, the 8 sets of orthogonal positive slopes that emerge are illustrated. There will be another 7 negative slopes, not depicted, that should be considered.

The pseudo-code listed as Algorithm 2 manages to estimate the intensity and orientation of bars, per tile of size tile_size, in the image I. The horizontal and vertical DRTs of I will be calculated as a previous step.

The method itself is simple, it performs $2 \cdot \text{tile\_size} - 1$ estimations consisting of accessing tile_size values in the horizontal DRT, reducing them to the absolute cumulative of their differences, and comparing them with the same in the vertical DRT. We keep the slope with the maximum absolute difference. This is calculated by tile, of which there are $\text{n\_squares} = \frac{N}{\text{tile\_size}}$ per dimension. Access to horizontal and vertical values is on lines 18 to 23, and reduction and comparison goes from line 24 to 30.

Algorithm 1 Compute the DRT of horizontal lines of an image

Input: $I(x, y) \longrightarrow$ Image consisting of $N \times N$ data
Input: $N \longrightarrow$ Size of a side of the image
Input: tile_size $\longrightarrow$ Size of the tiles
Output: $F(square, slope, displacement) \longrightarrow$ Result of the transform

function horizontal_drt(I, N, tile_size)
  tile_size_bits $\leftarrow \lfloor \log_2(\text{tile\_size}) \rfloor$
  for stage $= 0$ to tile_size_bits **do** ▷ Allocate memory and fill with zeros.
    [n_squares, slope_size] $\leftarrow$ **get_size_horizontal_drt**(N, stage, tile_size)
    $\text{F}_{\text{stage}} \leftarrow$ **zeros**(n_squares, slope_size, N)
  end for
  for $i = 0$ to $N - 1$ **do** ▷ Copy input to F_0
    for $j = 0$ to $N - 1$ **do**
      $F_0(i, 0, j) \leftarrow I(i, j)$
    end for
  end for
  for stage $= 0$ to tile_size_bits $- 1$ **do**
    current_stage_size $\leftarrow 2^{\text{stage}}$
    out_stage_size $\leftarrow 2^{\text{stage}+1}$
    [out_n_squares, out_slope_size] $\leftarrow$ **get_size_horizontal_drt**(N, stage $+ 1$, tile_size)
    for out_y_square $= 0$ to out_n_squares $- 1$ **do**
      for unsigned_slope $= 0$ to out_slope_size $- 1$ **do** ▷ slope as unsigned index
        slope $\leftarrow$ unsigned_slope $-$ out_stage_size $+ 1$ ▷ positive and negative slopes
        ab_s $\leftarrow |\text{slope}|$
        s_sign $\leftarrow 1$
        if slope < 0 **then**
          s_sign $= -1$
        end if
        s2 $\leftarrow \lfloor \frac{\text{ab\_s}}{2} \rfloor$
        rs $\leftarrow$ ab_s $- 2 \times$ s2 ▷ remainder of absolute slope
        slopeM $\leftarrow$ current_stage_size $- 1 +$ s2 $\times$ s_sign ▷ slopes of previous segments
        incIndB $\leftarrow$ s_sign $\times$ (s2 $+$ rs) ▷ displacement among segments
        for writeIdx $= 0$ to $N - 1$ **do**
          readIdx $\leftarrow$ writeIdx
          A $\leftarrow 0$
          B $\leftarrow 0$
          if $0 \leq$ readIdx $< N$ **then**
            A $\leftarrow F_{\text{stage}}$(out_y_square $\times 2$, slopeM, readIdx)
          end if
          if $0 \leq$ readIdx $+$ incIndB $< N$ **then**
            B $\leftarrow F_{\text{stage}}$(out_y_square $\times 2 + 1$, slopeM, readIdx $+$ incIndB)
          end if
          $F_{\text{stage}+1}$(out_y_square, unsigned_slope, writeIdx) $\leftarrow$ A $+$ B
        end for
      end for
    end for
  end for
  return $F_{\text{tile\_size\_bits}}$
end function

function get_size_horizontal_drt(N, stage)
  stage_size $\leftarrow 2^{\text{stage}}$
  n_squares $\leftarrow \lfloor \frac{N}{\text{stage\_size}} \rfloor$
  slope_size $\leftarrow 2 \times$ stage_size $- 1$
  return [n_squares, slope_size]
end function

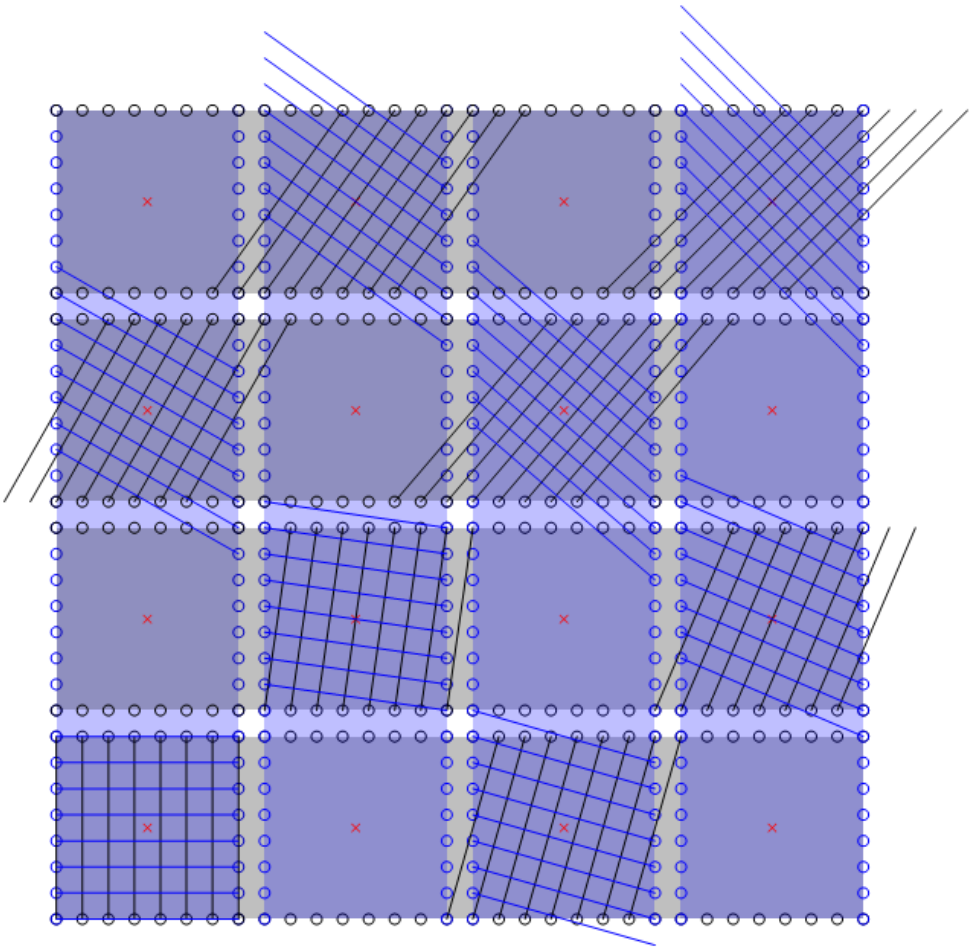

Figure 7: Blue vertical, and gray horizontal, stripes of width 8 in a 32×32 grid. They overlap on 4×4 regions whose centers are marked with red crosses. In alternative regions they are shown the sets of vertical and horizontal lines of orthogonal positive slopes that can be compared to estimate bar presence in them.

As a drawback, there is a trade-off between the estimation accuracy, which increases with the tile_size; and the number of estimations, that is $\frac{N}{\text{tile\_size}}$ per dimension.

The number of operations, considering the estimation alone, is

$$\overbrace{2 \cdot \text{tile\_size}}^{\text{slopes to be estimated}} \times \overbrace{2 \cdot \text{tile\_size}}^{\text{cost of estimator per slope}} \times \overbrace{\text{n\_squares} \times \text{n\_squares}}^{\text{size of output}} = 4 \cdot \text{tile\_size} \times N \times N.$$

Only 4 times the size of the tiles times the size of the input, almost linear if we consider that tile_size should be kept small, approximately 2 times the length of the wider bar to be detected. This number of operations are those of the estimator, and to consider the whole it should be

added twice the number of operations of a DTR. In total, if we express the size of tiles as 2^m:

$$O\left(2^{m+2} \times N \times N + 2(m \times N \times N)\right) \approx O(N^2 \log N), \text{ because } 2 \cdot m \leq 2^m \text{ and } 2^{m+2} \approx \log_2(N).$$

Algorithm 2 Bar detection using the partial DRTs

Input: $I(x, y) \longrightarrow$ Image consisting of $N \times N$ data
Input: $N \longrightarrow$ Size of a side of the image
Input: tile_size $\longrightarrow$ Size of the tiles
Output: intensity$(x, y) \longrightarrow$ Per tile detection of bar
Output: max_slopes$(x, y) \longrightarrow$ Per tile slope of the detected bar

function bar_detector(*I*, tile_size)
  N ← **size**(*I*)
  vertical_sums ← **horizontal_drt**(I^T, *N*, tile_size)
  horizontal_sums ← **horizontal_drt**(*I*, *N*, tile_size)

  n_squares ← **size**(*vertical_sums*)[0]
  n_slopes ← **size**(*vertical_sums*)[1]
  intensity ← **zeros**(n_squares, n_squares)
  max_slopes ← **zeros**(n_squares, n_squares)
  tile_size ← $\frac{N}{n\_squares}$
  for x_square = 0 to n_squares − 1 **do**
    for y_square = 0 to n_squares − 1 **do**
      x_central ← x_square × tile_size + $\frac{\text{tile\_size}}{2}$
      y_central ← y_square × tile_size + $\frac{\text{tile\_size}}{2}$
      max_val ← $-\infty$
      max_slope ← −tile_size + 1
      for slope = −tile_size + 1 to tile_size **do**
        start_h ← y_central − $\frac{\text{tile\_size}}{2}$ − $\frac{\text{slope}}{2}$
        end_h ← y_central + $\frac{\text{tile\_size}}{2}$ − $\frac{\text{slope}}{2}$
        values_h ← horizontal_sums(x_square, slope + tile_size − 1, start_h : end_h)
        start_v ← x_central − $\frac{\text{tile\_size}}{2}$ + $\frac{\text{slope}}{2}$
        end_v ← x_central + $\frac{\text{tile\_size}}{2}$ + $\frac{\text{slope}}{2}$
        values_v ← vertical_sums(y_square, −slope + tile_size − 1, start_v : end_v)
        v1 ← $\sum$ |**diff**(values_h)|
        v2 ← $\sum$ |**diff**(values_v)|
        v ← |v1 − v2|
        if v > max_val **then**
          max_val ← v
          max_slope ← slope
        end if
      end for
      intensity(x_square, y_square) ← max_val
      max_slopes(x_square, y_square) ← max_slope
    end for
  end for
end function

4. RESULTS

4.1 Implementations and time measurements

As more and more computer vision problems are designed with camera-equipped smart mobile devices in mind, we will address whether we can achieve video acquisition rate processing times on a modern system-on-chip (SoC). Specifically our interest lies on the Qualcomm Snapdragon™ SoC architecture, where its Kryo™ CPU, Adreno™ GPU and Hexagon™ DSP are all interesting computations units where our algorithm can be executed on.

Normally a hand-crafted implementation on such three diverse platforms would be very time and effort consuming because of their different set of instructions and architectural characteristics. Thankfully there exists a language, Halide,[11] that separates the *definition* of the algorithm from its *scheduling*, allowing for a single functional representation of the algorithm and then a scheduling for each target architecture. It also allows the programmer to apply and test different acceleration approaches without knowing the details of each architecture. Halide also provides a set of *auto-schedulers* that can give a starting *schedule* by using heuristics[12] that can be improved upon.

Both parts of the solution, the `horizontal_drt` and the `bar_detector` have been programmed as Halide pipelines and then the results have been validated with the original implementation. Then a manual scheduling, in opposite to auto-scheduling, have been crafted with the goal of achieving the smallest execution time possible in each architecture. Although the optimal scheduling may differ depending on the input sizes, the size of 512×512 with a tile_size of 8, has been chosen as sole target of our experiments.

Designing a good schedule is a process of trial and error. It begins by making an assumption based on the characteristics of the algorithm and the target architecture. For example, when targeting the CPU of the Snapdragon™ 865,[13] and writing a schedule for `horizontal_drt`, a first approach could be to use a coarse-grain parallelization to split the computation of the outer loop into several chunks that can be run at the same time on different CPU cores. Since the loop over `writeIdx` does not have dependencies over iterations, it can be parallelized any way we want. Note that `writeIdx` is the outer variable in our Halide implementation, instead of `out_y_square` as it was in the code presented above. The Snapdragon™ 865 CPU has 8 cores, so theoretically a speedup of 8 times could be achieved.

Actually this speedup can not be achieved due to factors such as parallelization overhead, memory locality and amount of cache misses. Additionally, in the case of the Android Operating System, a governor algorithm will be throttling the CPU in order to keep temperature and energy consumption controlled. This last factor is dependant on the manufacturer of the device. For these reasons, once a hypothesis has been mustered, it must be tested empirically.

Continuing with the `horizontal_drt` example, an initial scenario where a coarse-grain parallelization on the CPU cores is applied will be shown. First, the sequential execution of the algorithm must be measured in order to be able to know the achieved speedup. For this, the algorithm is executed a number of times that is enough to get a low standard deviation of the measured times.

The mean execution time for this schedule took 2.88 milliseconds in the mentioned CPU. If we just parallelize the outer variable, `writeIdx`, along the eight cores, the execution averages to 0.73 milliseconds. This represents a speedup of almost 4x. There are other strategies that can be combined to find the best schedule possible for each algorithm such as vectorization, splitting and fusing variables, reordering variables and unrolling. The following is an example of the schedule crafted for `horizontal_drt` on CPU, yielding a speedup of 8.5x:

```
1 void schedule() {
2     out.compute_root()
3         .reorder(slope, writeIdx, ySquareMp1)
```

```
4           .unroll(slope, 4)
5           .vectorize(ySquareMp1, 64)
6           .parallel(writeIdx);
7   }
```

With this in mind, a schedule has been crafted for `horizontal_drt` and `bar_detector` algorithms for the size of 512×512 and for each processing unit (DSP, GPU and CPU). The averaged execution times for different input sizes are represented in the figure 8 for the best schedules found. The tile_size is fixed to size/64, that is 8 for 512, and accordingly for other sizes.

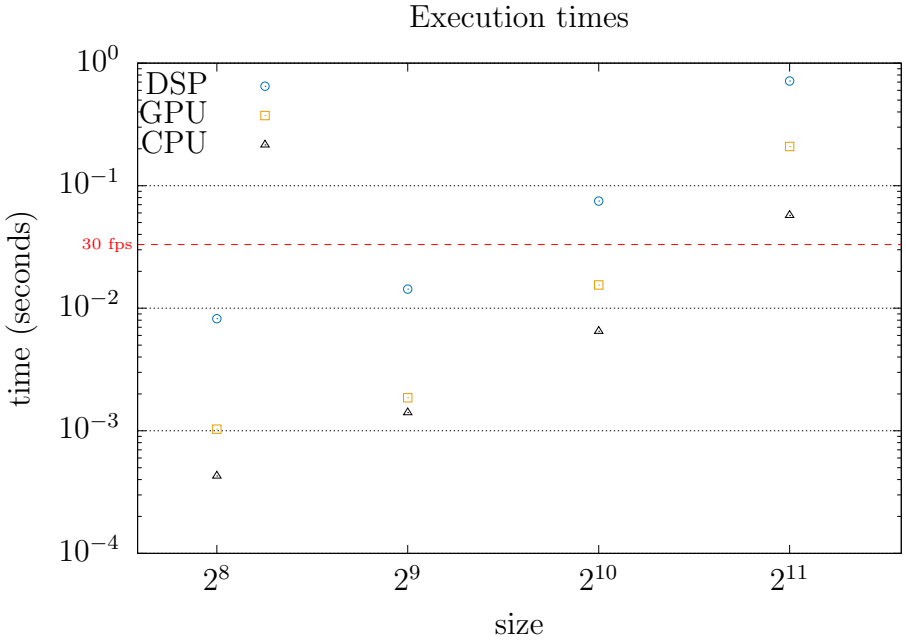

Figure 8: Times in seconds of the execution of `bar_detector` algorithm, which includes two calls to `horizontal_drt`, coded in Halide and run in different targets of the SoC Qualcomm Snapdragon™ 865.

5. COMPARISON WITH OTHER METHODS

As this contribution stops at the bar detection stage we will compare with other barcode detection methods solely on its performance on real-time, as we are currently not performing clustering and decoding stages, which should be measured under other metrics.

The hybridized geometrical+AI approach from Zamberletti *et al.*,[2] took 200ms to perform its localization stage on images of size 640×480 in a 2013 mobile device. Our method takes 1.4ms to process an image of 85% that size, 512×512, on the Snapdragon™ 865 CPU. This reduction in processing times is partially due to the advance in mobile architectures from 2013 to 2019, the year when Snapdragon™ 865 was released. Also because we are exploiting parallelism easily thanks to the Halide language, but more important because our method has been directly

elucidated to perform its task, and does not surge as the combination of preexisting methods to detect borders, then apply Hough transform, then chunk its result and then pass it to a neural network.

The other geometry-based method, that of Creusot *et al.*[3] takes half the time than Zamberletti's, but that is still 100ms, to process images of size 640×480, even if the target platform is a PC.

A 2019 work from Xiao and Ming[14] reports 14.5ms, on an Intel Core i7 3.50 GHz processor, for the 1D barcode region location and rotation angle prediction on images sized 640×480. They use a neural network approach to determine a region of interest and then a geometric approach to determine the bar orientation. Their time of 14.5 ms is highlighted as a milestone given that it is several times faster than the methods[15–17] they review.

As we have discussed our method currently exhibits $O(N^2 \log_2 N)$ complexity and requires just integer arithmetics which are prone to parallelization and SIMD acceleration. Even then, the times obtained in a mobile device exceeded our expectations, as they are over 10x faster than what other authors report in more powerful devices, for an input just 1.17x larger.

6. FUTURE LINES

We expect to address the problem of vanishing detection at the borders of a bar zone, something that should be doable repeating the estimation at a smaller grid, and for less orientations therefore. That is, a multiscale estimator, where just for dubious regions we can look at the intermediate results at stages lesser than the tile size.

Another objective is to eliminate the trade-off between estimation density and estimation precision, which is currently due to the fact that both depend, one directly and the other inversely, on the tile size. This is because we have considered only non-overlapping regions. We already have in mind how, without repeating calculations, to achieve this goal, forcing the partial transforms to be computed at overlapping stripes.

7. CONCLUSIONS

We have developed a method that, based on the partial stages of the DRT for horizontal and vertical lines, manages to estimate the presence and local orientation of bars.

Although we have not performed an exhaustive study on the accuracy of the detection, given the way it works by simultaneously considering sets of integrals in orthogonal directions, it seems to be less dependent on noise than other local methods that would be based on the same idea of gradients with preferential orientation over a whole area.

Moreover, while realizing a local method based on gradients but combining at scales larger than the pixel can be expensive, our method exhibits quasilinear time complexity, and can be expressed exclusively on (short)integers arithmetic.

We have formulated the simultaneous two-quadrant DRT in Halide; and the estimator from the outputs of the partial DRT in horizontal and vertical. We have managed to run both on the 3 computing platforms provided on a mobile device SoC.

For size 512×512, and a tile size of 8×8, it is not that we are below the 33ms barrier that determine the processing at 30fps, it is that we are an order of magnitude below, in the 2ms range. From this result we have more than enough margin to address in future work augmented reality applications.

On the other hand, in GPU and DSP we are obtaining slightly worse results than in CPU, but we expect to lower their times with more experience expressing low-level algorithms on these platforms.

Disclosures

The authors declare that there is no conflict of interest.

Acknowledgements

This work has been partially supported by the project ProID2020010066 of 'Estrategia de Especialización inteligente de Canarias RIS-3', cofunded by Government of the Canary Islands and European Regional Development Fund (ERDF). J.G.M.-H. has been partially supported by 'Convenio de investigación Wooptix-ULL: Asesoramiento técnico sobre *consumer electronic* con tecnología *lightfield*, 2022'. This work has also been supported by 'Research incentives for promoting open dissemination of results of research excellence' granted by 'Universidad de La Laguna' and 'Consejería de Economía, Conocimiento y Empleo (CEI Canarias-ULL, SD-19/02)'.

Code, Data, and Materials Availability

Code and data samples are available from authors upon reasonable request.

REFERENCES

[1] "GS1 general specifications," standard, GS1, 2022. §5.1-§5.5. https://www.gs1.org/docs/barcodes/GS1_General_Specifications.pdf.

[2] A. Zamberletti, I. Gallo, and A. Simone, "Robust angle invariant 1D barcode detection," 11 2013.

[3] C. Creusot and A. Munawar, "Real-time barcode detection in the wild," in *Applications of Computer Vision (WACV), 2015 IEEE Winter Conference on*, pp. 239–245, Jan 2015.

[4] R. O. Duda and P. E. Hart, "Use of the hough transformation to detect lines and curves in pictures," *Commun. ACM* **15**, p. 11–15, jan 1972.

[5] W. Götz and H. Druckmüller, "A fast digital Radon transform—An efficient means for evaluating the Hough transform," *Pattern Recognition* **29**(4), pp. 711–718, 1996.

[6] S. Ando and H. Hontani, "Automatic visual searching and reading of barcodes in 3-d scene," pp. 49 – 54, 02 2001.

[7] J. Radon, "Über die Bestimmung von Funktionen durch ihre Integralwerte längs gewisser Mannigfaltigkeiten," *Akad. Wiss.* **69**, pp. 262–277, 1917.

[8] M. L. Brady, "A fast discrete approximation algorithm for the Radon transform," *SIAM Journal on Computing* **27**(1), pp. 107–119, 1998.

[9] A. Brandt and J. Dym, "Fast calculation of multiple line integrals," *SIAM Journal on Scientific Computing* **20**(4), pp. 1417–1429, 1999.

[10] O. Gómez-Cárdenes, R. Oliva-García, G. A. Rodríguez-Abreu, and J. G. Marichal-Hernández, "Exposing parallelism of discrete radon transform," in *Proceedings of the 3rd International Conference on Telecommunications and Communication Engineering, ICTCE '19*, p. 136–140, Association for Computing Machinery, (New York, NY, USA), 2019.

[11] J. Ragan-Kelley, C. Barnes, A. Adams, S. Paris, F. Durand, and S. Amarasinghe, "Halide: A language and compiler for optimizing parallelism, locality, and recomputation in image processing pipelines," *SIGPLAN Not.* **48**, p. 519–530, jun 2013.

[12] A. Adams, K. Ma, L. Anderson, R. Baghdadi, T.-M. Li, M. Gharbi, B. Steiner, S. Johnson, K. Fatahalian, F. Durand, and J. Ragan-Kelley, "Learning to optimize halide with tree search and random programs," *ACM Trans. Graph.* **38**, jul 2019.

[13] "Lantronix - Snapdragon™ 865 mobile HDK," 2019. https://www.lantronix.com/products/snapdragon-865-mobile-hdk/#tab-techspecs.

[14] Y. Xiao and Z. Ming, "1D barcode detection via integrated deep-learning and geometric approach," *Applied Sciences* **9**(16), 2019.

[15] A. Namane and M. Arezki, "Fast real time 1d barcode detection from webcam images using the bars detection method," in *Proceedings of the World Congress on Engineering (WCE)*, **1**, 2017.

[16] C. Creusot and A. Munawar, "Low-computation egocentric barcode detector for the blind," in *2016 IEEE International Conference on Image Processing (ICIP)*, pp. 2856–2860, IEEE, 2016.

[17] D. K. Hansen, K. Nasrollahi, C. B. Rasmussen, and T. B. Moeslund, "Real-time barcode detection and classification using deep learning," in *International Joint Conference on Computational Intelligence*, pp. 321–327, SCITEPRESS Digital Library, 2017.