CHAPTER

# 3 1D Edge Detection

Be precise. A lack of precision is dangerous when the margin of error is small.

DONALD RUMSFELD

## 3.1 Introduction

Image edges are locations of sharp change of brightness, i.e. locations of high local contrast. As a consequence of being a contrast-based feature, the presence and position of an edge is not altered by global illumination changes in the image; which contributes to the robustness of Edge Detection-based solutions.

Edge detection techniques come in two variants depicted in **Figure 3.1**. **1D Edge Detection** methods scan the image along a path and locate the points of intersection between image edges and the scan line. **2D Edge Detection** methods locate the entire edge. In this chapter we will inspect the first technique, featuring remarkable performance and sub-pixel precision.

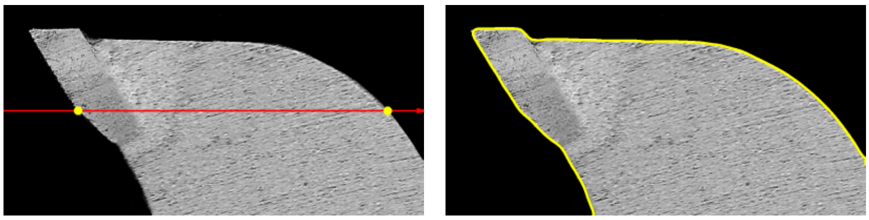


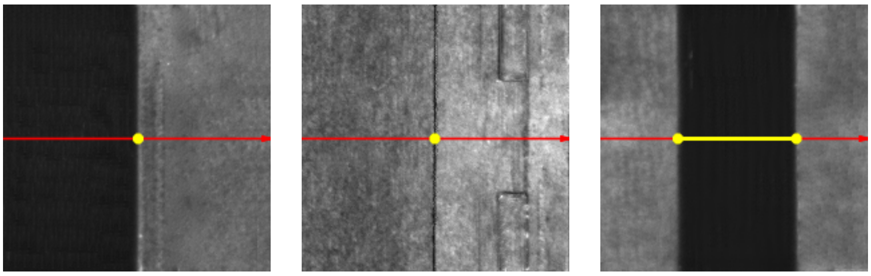
Figure 3.1: **1D Edge Detection**, **2D Edge Detection**

We will start with a short description of the methodology of extracting a 1D image brightness profile along a given path and then proceed to detection of the features present in the profile.

Depending on the nature of the brightness change that constitutes an edge, we distinguish two kinds of edges: **step edges**, occurring between two areas of different intensity, and **ridge edges** (or simply ridges), occurring where image intensity changes briefly and then returns to initial value.

A wide class of visual inspection tasks is focused on the areas bounded by two step edges of opposite polarity, rather than on the edges considered separately. Because of that it is useful to consider such areas as a third, additional type of feature discernible in one-dimensional profile; here called a **stripe**.

Overall, the chapter will cover detection of three kinds of features discernible in 1D profiles, all demonstrated in **Table 3.1**. In each example the feature (step edge, ridge or stripe) is vertical and the image is scanned horizontally to find the points of intersection between the feature and the scan line.



**Step Edge** **Ridge** **Stripe**

Table 3.1: Different kinds of image features extracted from 1D profile.

## 3.2 Profile Extraction

Before we apply any of the **1D Edge Detection** methods, firstly we need to acquire a 1D profile that is to be inspected. Computing a discrete profile of image brightness along a given path is relatively straightforward task.

The first step is to sample the scan path, selecting a set of equidistant (typically with distance of one pixel [7, p. 150]) points of interest along the path. Each of these points will correspond to one element of the constructed profile. The second step is to compute the brightness values *related to* each of the points. **inScanPath**

### Multiple Sampling

We used the expression *related to* rather than *at* on purpose - although we could simply take the image brightness values at each point of interest, it is more prudent to use an average of a series of sampling points perpendicular to the scan line, as demonsted in **Figure 3.2**; thus achieving a simple mean of noise suppression. But what kind of average should we use to compute the result for a single point of interest?

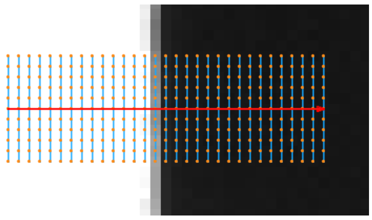


Figure 3.2: Multiple Sampling for 1D Edge Detection.

As long as the whole range of sampling points fits within the object being inspected and its features are perpendicular to the scan path, the brightness information collected at each of the sampling points is equally good or bad as the information collected at the other ones. Because of that we may safely use the simplest arithmetic mean.

We will refer to the number of points used to compute a single profile value as *scan width*. The wider the scan, the stronger noise attenuation we get. However, if the 2D feature we are to inspect is not perfectly perpendicular to the scan line, the wide scan area will cause the edge in the resulting 1D profile to be stretched and thus harder to identify precisely. Increasing the scan width will also increase the computation time of the profile extraction, which depends linearly on the number of sampling points.

**Figure 3.3** depicts an example scan path along with the area of pixels taken into account as well as the extracted brightness profile.

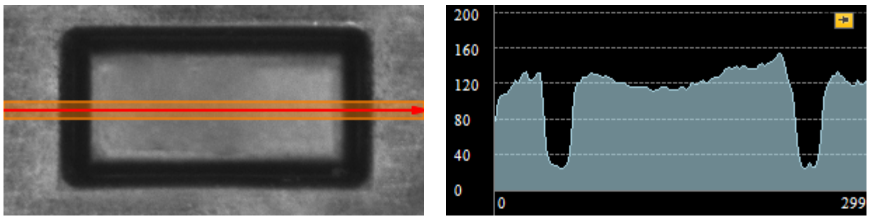


Figure 3.3: Scan area and the extracted brightness profile.

## Refinement

Even though reasonably wide scan area suppresses the noise significantly at the extraction level, we need to keep in mind that only random noise can be suppressed in this way. Real pictures contain small features[1] of image texture irrelevant to the inspection task that need to be attenuated before further processing of the profile.

Selecting the smoothing filter to refine the extracted profile is not an obvious choice. On one hand we want to suppress the noise present in the profile, so that irrelevant intensity changes are not identified as edge points, on the other hand we want to achieve high precision of the edge localization.

These two criteria cannot be considered independently - smoothing of the profile suppresses the noise, but also lowers the precision. Canny determined[8] that the Gaussian smoothing operator yields the optimal trade-off between achieved noise reduction and introduced lose of precision.

**inSmoothing-StdDev**

**Standard deviation** determining the degree of actual smoothing performed by the Gaussian operator is a parameter of the algorithm. Accurate adjustment of this value will contribute to the robustness of the computation. We should pick a value that is high enough to eliminate noise that could introduce irrelevant extrema to the profile derivative, but low enough to preserve the actual edges we are to detect.

**diagProfile**

The standard deviation parameter should be adjusted by hand, through interactive experimentation on representative sample data. We should inspect the results of the profile extraction and refinement looking for optimal trade-off between fidelity and noise attenuation.

**Figure 3.4** demonstrates profiles indicating too low and too high standard deviation values - in the first case the profile contains fine noise, in the second case significant edges are attenuated. **Figure 3.5** demonstrates an appropriate smoothing - fine noise is lost, while significant edges are preserved.

[1]Possibly perpendicular to the scan line and thus not affected by averaging the sampling points.

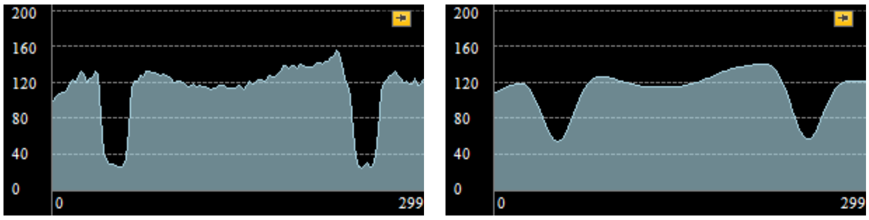


Figure 3.4: Noisy or flattened profile indicates that standard deviation is sub-optimal.

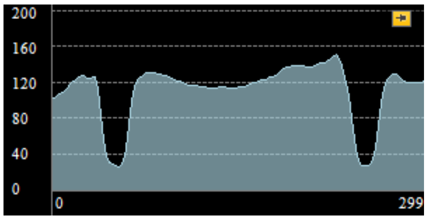


Figure 3.5: Profile indicating optimal standard deviation value.

## 3.3 Step Edges

Once the brightness profile is extracted and smoothed, we can proceed to detection of its features. First type of features that we will inspect is **step edge**. Step edges occur between two areas of different intensity and are represented as an abrupt intensity change in the 1D profile.

### Edge Operator

Finding the step edges in a profile requires an edge operator - an operator that produces high output for locations representing sharp change of brightness and low output for the signal plateaus. One such operator is **derivative** of a function - elementary concept of calculus. This is also the operator suggested by Canny in already mentioned work [8]. But how do we actually compute the derivative?

In case of a continuous signal its derivative is well defined. As both image and (consequently) its brightness profile are discreet, we are left with partial differences - discreet approximations of the signal first derivative.

The simplest way to compute the partial difference of a discreet signal $S$ is to subtract each value from its successor, i.e.:

$$D[i] = S[i+1] - S[i]$$

This operator, called **Forward Difference**, has a slight drawback - the resulting approximation $D[i]$ actually corresponds to the domain value in between $i+1$ and $i$ (i.e to $i + \frac{1}{2}$). To achieve *stable* approximation of the first derivative we can compute the value $D'[i]$ as a mean of $D[i]$ and $D[i-1]$ (**Central Difference**):

$$\begin{aligned} D'[i] &= \frac{1}{2}(D[i] + D[i-1]) \\ &= \frac{1}{2}(S[i+1] - S[i] + S[i] - S[i-1]) \\ &= \frac{1}{2}(S[i+1] - S[i-1]) \end{aligned}$$

Both equations are feasible for our application, however we need to remember about the $\frac{1}{2}px$ shift introduced by Forward Difference operator and translate the edge points accordingly on the very end. **Figure 3.6** demonstrates an example finite difference profile.

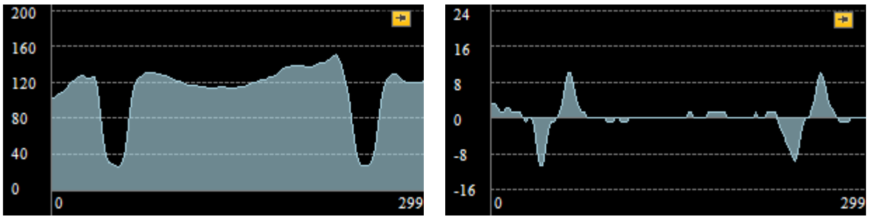


Figure 3.6: Forward difference of an example profile.

## Edge Points

Once we have computed the derivative we can identify the edge points of the original profile. There are two criteria that a profile value has to meet to be considered an edge point:

1. Significant magnitude, i.e. magnitude larger than some predefined threshold.

2. Locally maximal magnitude.

Both conditions are necessary - first ensures that only significant brightness changes are identified as edge points, second (called Non-Maximum Suppression) ensures that a significant but stretched edge yields only one edge point.

**inMinStrength**
**diagDerivative**

The value of the minimum magnitude threshold in each case should be adjusted after inspection of derivative profile of sample data. **Figure 3.7** demonstrates an example in which the extrema corresponding to object edges vary in magnitude from 11 to 13, while the magnitude of other extrema is lower than 3. Therefore any threshold value in range $(4, 10)$ will be appropriate. A value in the middle of that range would be a prudent choice.

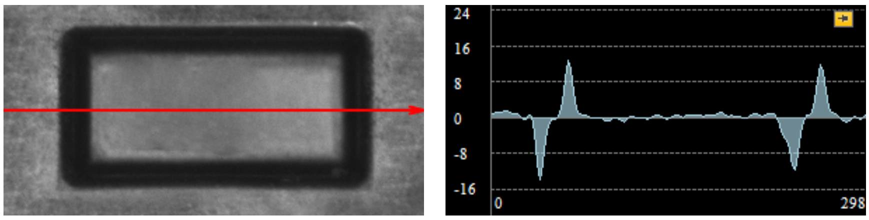


Figure 3.7: Derivative profile of example data should be inspected to determine the minimal edge point magnitude.

Profile locations meeting the magnitude criteria directly translate to edge points in the original image. An example set of extracted edge points is demonstrated in **Figure 3.8**.

## Sub-pixel precision

Even though both the image being inspected and the extracted brightness profile are discreet with pixel-precision, we can compute the local extrema of the derivative profile with sub-pixel precision thus achieving sub-pixel precision of the entire method.

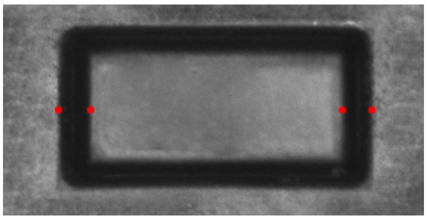


Figure 3.8: Result of **1D Edge Detection** - a list of edge points along the scan path.

Given a local extremum of a profile $P$ at location $i$ we can fit a parabola through three consecutive profile values: $P[i-1]$, $P[i]$, $P[i+1]$ and use the x-coordinate of its peak as the location of the extremum.

### Edge polarity filtering

It is often useful to filter the extracted edge points depending on the transition they represent - that is, depending on whether the intensity changes from bright to dark, or from bright to dark.

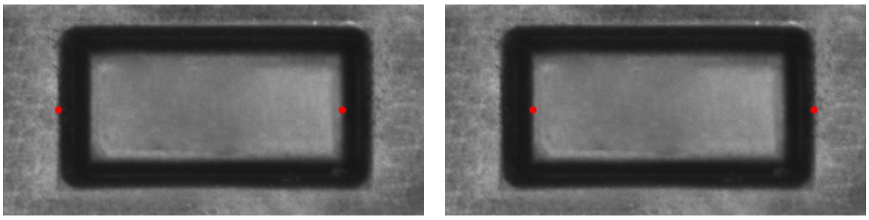


Figure 3.9: **inTransition** = BrightToDark, **inTransition** = DarkToBright

## Post-processing and Available Filters

Once we have extracted the list of relevant edge points in the image we are nearly done. Depending on the use scenario, it may be useful to perform additional filtering of the extracted points on the very end of computation. Three methods of post-processing are particularly useful.

### Multiple Edges

This method essentially performs no post-processing at all, simply returning all of the extracted edge points. We should stay with this variant whenever we want to detect the number of edges present in the image.

In **Adaptive Vision Studio 2.5**1D Edge Detection concluded with this method of post-processing is available in the **ScanMultipleEdges** filter. As opposed to the other variants this filters always returns a non-Nil result - an (possibly empty) array of extracted edge points.

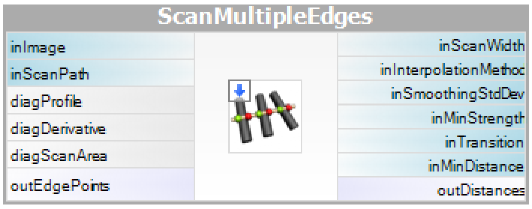


Figure 3.10: The ScanMultipleEdges filter.

### Exactly N Edges

This method selects a set of edge points of desired cardinality yielding the best sum of edge points magnitudes. If we know the number of edges in advance, this method allows us to disregard the adjustment of minimum magnitude threshold - we can simply set it to zero and expect that the actual edges will still be correctly located.

That being said, it is often useful to adjust minimum magnitude threshold anyway, so that in case of an error such as the object not being present in the image, the computation will explicitly yield null result rather than selecting irrelevant weak edges.

In **Adaptive Vision Studio 2.5**1D Edge Detection concluded with this method of post-processing is available in the **ScanExactlyNEdges** filter. The filter returns an array of exactly N edge points or Nil if no correct set of edge points of such cardinality was found.

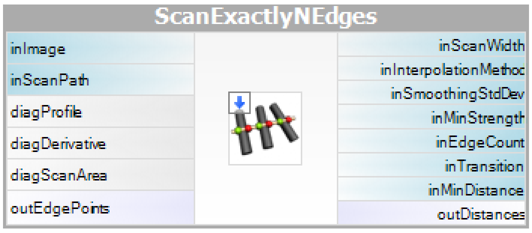


Figure 3.11: The ScanExactlyNEdges filter.

## 3.4 Ridges

Ridges are brief bright or dark impulses on contrasting background, example demonstrated in **Figure 3.12**. Differing from step edges in their definition, they also require slightly different method of extraction. We will start the description from the point in which we have just extracted and refined the 1D profile of image brightness.

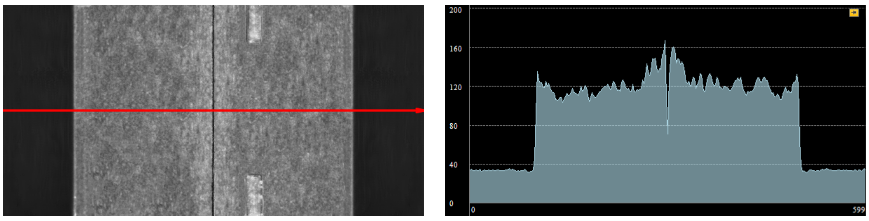


Figure 3.12: An example ridge and brightness profile of a horizontal scan.

### Ridge Operator

Ridges can be thought of as pairs of step edges of opposite polarity lying extremely close to each other. We could use this observation to propose a simple ridge detector operator adding together results of Forward Difference and Backward Difference operators:

$$\begin{aligned} R[i] &= (S[i] - S[i-1]) + (S[i] - S[i+1]) \\ &= 2S[i] - S[i-1] - S[i+1] \end{aligned}$$

Such operator would be a discreet equivalent of the ridge operator proposed by Canny[8], it has however two important drawbacks, pointed out by Subirana-Vilanova and Sung[9]:

- The operator has non-zero response for step edges, which can easily lead to false-positive errors.
- The quality of the detection strongly depends on the ridges having exactly 1 pixel in width, while in reality ridges usually appear as at least slightly wider.

Both problems are illustrated by **Figure 3.13** - we can notice high impulse response for step edges on the boundary of the object. Moreover, as the ridge in the original image has three pixels in width, it appears in the resulting profile as a pair of consecutive step edges.

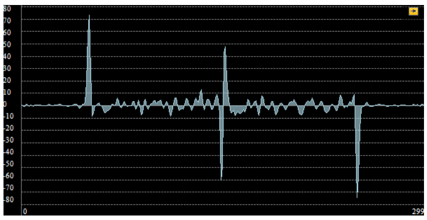


Figure 3.13: Naive ridge detection operator applied to example brightness profile.

The authors of [9] suggest to solve the problem of high response to step edges by applying each half of the ridge operator separately and using the minimum of two responses.

$$R[i] = \min(S[i] - S[i-1], S[i] - S[i+1])$$

Such form of the operator is already feasible for narrow (one pixel wide) ridges. To successfully detect wider ridges we could define a general operator parametrized by the width of the ridge and the width of the reference margin as follows:

$$
\begin{aligned}
R[i] \quad &= \min( \\
&\qquad \overline{S[i..(i+Width)]} - \overline{S[(i-1-Margin)..(i-1)]}, \\
&\qquad \overline{S[i..(i+Width)]} - \overline{S[(i+1)..(i+1+Margin)]}) \\
&)
\end{aligned}
$$

Example results of ridge detection performed using such operator are demonstrated in **Figure 3.14**.

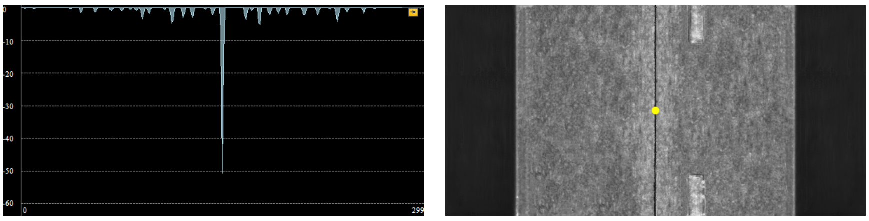


Figure 3.14: .

## Post-processing and Available Filters

All methods of post-processing of the extracted step edge points described in Step Edges section are applicable to ridges. **Adaptive Vision Studio 2.5**contains the following ridge detection filters.

### Multiple Ridges

Similarly to **ScanMultipleEdges**, this method performs the core ridge-detection computation without any filtering of the produced results.

### Exactly N Ridges

Similarly to **ScanExactlyNEdges**, this method firstly extracts the ridges present in the profile and later selects the subset of results of desired cardinality, yielding the best sum of ridge strengths.

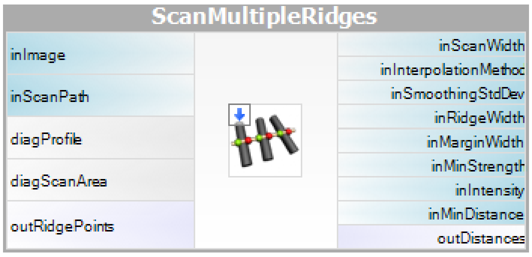


Figure 3.15: The ScanMultipleRidges filter.

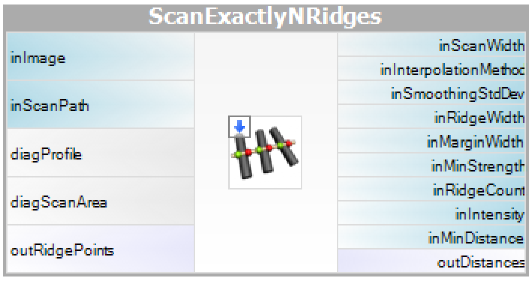


Figure 3.16: The ScanExactlyNRidges filter.

## 3.5 Stripes

Stripes are flat sections of brightness profile bounded by two step edges of opposite polarity. Such definition indicates that problem of stripe detection heavily depends on already discussed detection of step edges.

The concept of stripe is important mostly as a clear and succint mean of formulation for a range of visual inspection tasks, wheareas it does not bring any novelties to the signal-processing aspect of the computation. Indeed, algorithms for stripe extraction firstly find the step edges in the profile (using previously described methods) and then process the results combining the extracted edges into stripes.

Next section summarizes two basic methods of combining the extracted step edges into stripes.

## Edge Processing and Available Filters

### Multiple Stripes

As long as our goal is to maximize the number of constructed stripes, the problem can be solved quite efficiently. It can be easily proven that a simple $O(n)$ algorithm that greedily connects each closing edge with the first opening edge between already constructed stripes and the closing edge itself yields optimal results.

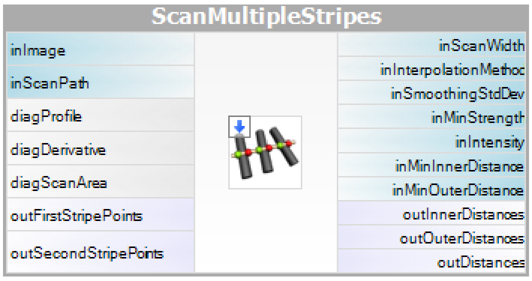


Figure 3.17: The ScanMultipleStripes filter.

### Exactly N Stripes

The task is slightly more complicated if we know the desired number of stripes in advance and aim at maximizing the sum of strengths of step edges constituing the selected stripes. To solve such optimization problem in $O(n^2)$ time we can use a dynamic programming solution.

Let us define a partial solution to the problem as follows:

`Best[Prefix][Count]` - sublist of the first `Prefix` step edges of alternating edge-polarities having `Count` elements that yields the biggest sum of edge strengths.

Having computed the results for `Prefix` $= p$ we can compute the results for `Prefix` $= p + 1$ in linear time - for each `Count` $= c$ we need to consider only two cases, either using the $p + 1$-th step edge to extend the optimal result of `Best[p][count-1]` or not, in which case the result for the subproblem will be equal to `Best[p][count]`.

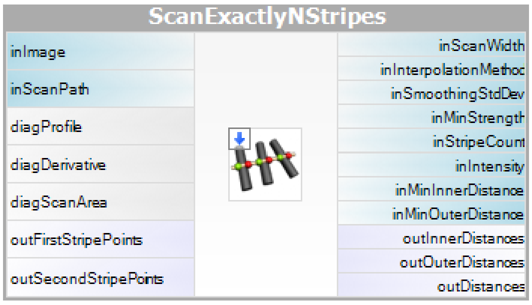


Figure 3.18: The ScanExactlyNStripes filter.

## 3.6 Applications

### Positioning

**1D Edge Detection** methods are commonly employed to determine position of image objects. Let us consider an image of a capsule on production line demonstrated in **Figure 3.19**. We assume that the capsule is aligned with the axis of the image and want to determine the range of x-coordinates occupied by the object.

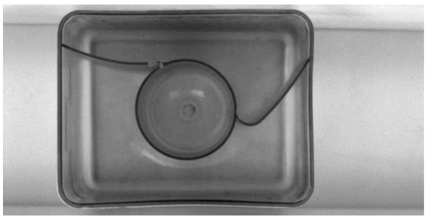


Figure 3.19: An example positioning task.

As long as the background is plain and contrasting with object border, such problem can be solved easily regardless of the inner content of the object. **Figure 3.20** demonstrates the result of running the ScanMultipleEdges filter along a horizontal scan line - the algorithm detects redundant, inner edges, but this does not pose a difficulty, as we only need the first and the last edge point of the returned list.

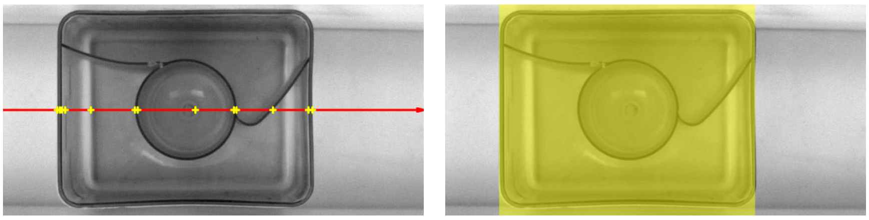


Figure 3.20: Results of the 1D Edge Detection-based positioning.

## Counting

Another popular application of **1D Edge Detection** is counting of the objects positioned along a path. Let us assume that we want to count the blades of a circular presented in **Figure 3.21**.

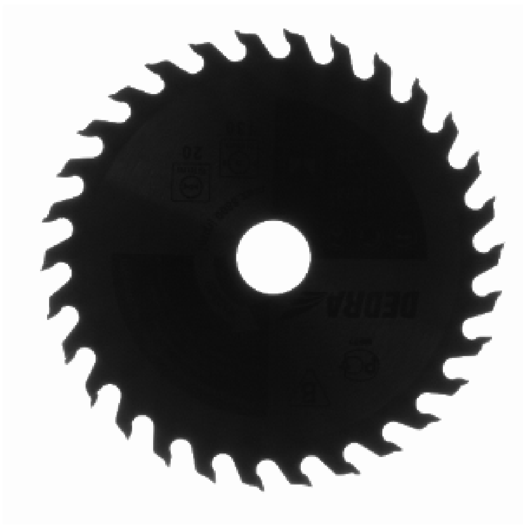


Figure 3.21: A circular saw blade to be inspected.

Such task may be solved by running a single edge detection scan along a circular path intersecting the blades. To produce the scan path we can use a straightforward **CreateCircularPath** filter. The built-in plugin will allow us to point & click the required **inCircle** parameter, as demonstrated in **Figure 3.22**.

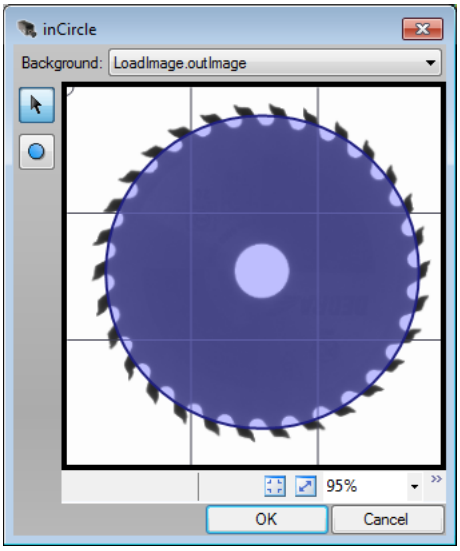


Figure 3.22: Interactive selection of the circle parameter.

The next step will be to pick the specific algorithm for the actual edge detection. As the objects being inspected appear as stripes in one-dimensional brightness profile and we do not know how many of them there are, the **ScanMultipleStripes** would be a right tool for the job.

The program depicted in (...) solves the problem as expected (perhaps after increasing the inSmoothingStdDev from default of 0.6 to bigger value of 1.0 or 2.0) and detects all 30 blades of the saw.

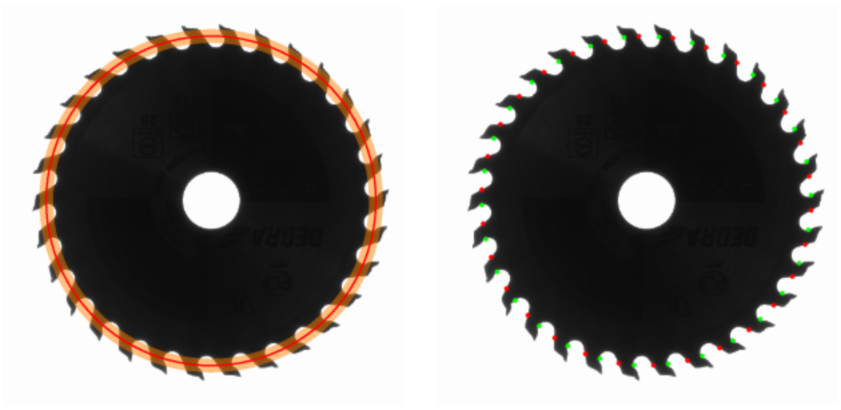


Figure 3.23: Scan area and the extracted stripes.