
Template Matching

If Edison had a needle to find in a haystack, he would proceed at once with the diligence of the bee to examine straw after straw until he found the object of his search.

A little theory and calculation would have saved him ninety per cent of his labor.

NIKOLA TESLA

8.1 Introduction

Template Matching is a set of machine vision techniques that allow to find objects within an image (or multiple images) that match the given template.

Template Matching algorithms are relatively straightforward to use and allow to find template occurrences regardless of their orientation. Their applicability is limited to tasks in which the object that is to be found has well-defined shape that we know in advance.

We will start with a demonstration of a naive Template Matching method, which is insufficient for real-life applications, but illustrates the concept from which other Template Matching algorithms stem from (**Naive Method**).

In the next two sections we will explain how this idea is improved in more advanced algorithms (**Grayscale-Based Matching**, **Edge-Based Matching**). We will conclude this chapter with a presentation of the Template Matching filters available in Adaptive Vision Studio (**Filter Toolset**).

8.2 Naive Method

Template Matching techniques are expected to address the following need: provided a reference image of an object (**template image**) and an image to be inspected (**input image**) we want to identify all input image locations in which the object from the template image is present.

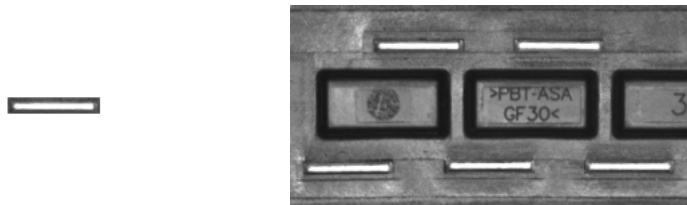


Figure 8.1: Example of a Template Matching task: find the template depicted in the first image in the second image.

Imagine that we are going to solve the inspection task demonstrated in **Figure 8.1** - our goal is to find the pins of a plug. We are provided with a **template image** representing a single pin an **input image** representing a plug to be inspected.

We will perform the actual search in a straightforward way - we will position the template over the input image at every possible localization and each time we will compute some numeric measure of similarity between the template and the image segment it currently overlaps. Finally we will identify the positions that yield the best similarity measures as probable template positions.

Image correlation

One of the subproblems that occur in the specification above is calculating a **similarity measure** of the aligned template image and the overlapped segment of the input image; which is equivalent to calculating a similarity measure of two images of equal dimensions. This is a classic task, and a numeric measure of image similarity is usually called **image correlation**.

Cross-Correlation

The fundamental method of calculating the image correlation is so called cross-correlation, which essentially is a simple sum of pairwise multiplications of corresponding pixel values of the images. Formally:

$$CC(\text{Image1}, \text{Image2}) = \sum_{x,y} \text{Image1}(x, y) \times \text{Image2}(x, y)$$







Image1	Image2	CC
		19404780
		23316890
		24715810

Table 8.1: Cross-correlation values for example image pairs.

Though we may notice that the correlation value indeed seems to reflect the similarity of the images being compared, cross-correlation method is far from being robust. Its main drawback is that it is biased by changes in global brightness of the images - brightening of an image effectively increases its cross-correlation with another image, even if the second image is not at all similar.

Normalized Cross-Correlation

Normalized Cross-Correlation is an enhanced version of the classic cross-correlation method that introduces two improvements over the original one:

- The results are invariant to the global brightness changes, i.e. consistent brightening or darkening of either image has no effect on the result (this is accomplished by subtracting the mean image brightness from each pixel value).
- The final correlation value is scaled to $[-1, 1]$ range, so that NCC of two identical images equals 1.0, while NCC of an image and its negation equals -1.0.







Image1	Image2	NCC
		-0.417
		0.553
		0.844

Table 8.2: Normalized Cross-Correlation values for example image pairs.

$$\text{NCC}(\text{Image1}, \text{Image2}) = \frac{1}{N\sigma_1\sigma_2} \sum_{x,y} (\text{Image1}(x,y) - \overline{\text{Image1}}) \times (\text{Image2}(x,y) - \overline{\text{Image2}})$$

Exercise: Verify that Normalized Cross Correlation of an image and its negation equals -1.0 using filters: [NegateImage](#), [ImageCorrelation](#).

Template Correlation Image

Let us get back to the problem at hand. Having introduced **Normalized Cross-Correlation** - robust measure of image similarity, we are now able to determine how well the template fits in each of the possible positions. We may represent the results in form of an image, where brightness of each pixels represents the NCC value of template positioned over this pixel (black color representing the minimal correlation of -1.0, white color representing the maximal correlation of 1.0).

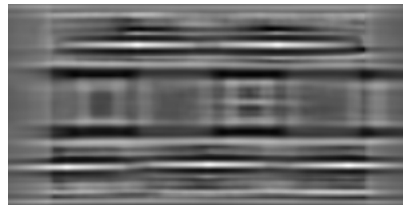


Figure 8.2: Each pixel of the Template Correlation Image represents the similarity between the template and the input image at given location.

Identification of matches

All that needs to be done at this point is to decide which points of the template correlation image are good enough to be considered actual matches. Usually we identify as matches the positions that (simultaneously) represent the template correlation:

- stronger than some predefined threshold value (i.e. stronger than 0.5)
- locally maximal (stronger than the template correlation in the neighboring pixels)

Summary

It is quite easy to express the described method in Adaptive Vision Studio - we will need just two built-in filters. We will compute the template correlation image using the **ImageCorrelationImage** filter, and then identify the

8. TEMPLATE MATCHING

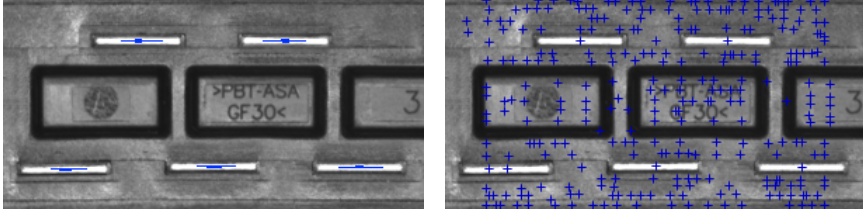


Figure 8.3: Areas of template correlation above 0.75 and points of locally maximal template correlation

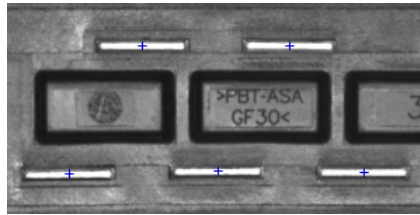


Figure 8.4: Intersection of two candidate sets - points of locally maximal template correlation of value at least 0.75.

matches using [ImageLocalMaxima_Regions](#) - we just need to set the **inMinValue** parameter that will cut-off the weak local maxima from the results, as discussed in previous section.



Figure 8.5: Naive Matching method implemented in Adaptive Vision Studio.

Exercise: Prototype a program that will find the pins of a plug using Naive Template Matching method. Use filters: **ImageCorrelation-Image**, **ImageLocalMaxima_Regions** and the input files: `pin.png`, `plug.png`.

Though the introduced technique was sufficient to solve the problem being considered, we may notice its important drawbacks:

- Pattern occurrences have to preserve the orientation of the reference template image.
- The method is inefficient, as calculating the template correlation image for medium to large images is time consuming.

In the next sections we will discuss how these issues are being addressed in advanced template matching techniques: **Grayscale-Based Matching** and **Edge-Based Matching**.

8.3 Grayscale-Based Matching

Grayscale-Based Matching is an advanced Template Matching algorithm that brings major improvements to the idea of correlation-based template detection, enhancing its efficiency and allowing to search for template occurrences regardless of their orientations.

Advantages of Grayscale-Based Matching come as a consequence of efficient schema of computation called **Pyramid Processing**. In this section we will introduce this concept and explain how is it employed in Grayscale-Based Matching.

Image Pyramid

Image Pyramid (see **Figure 8.6** for an example) for a given image is a series of images, each image being a result of downsampling¹ of the previous element. First image of the pyramid is the initial image. Conventionally we call elements of the pyramid **pyramid levels**, the initial image being at the lowest level 0.

¹scaling down, by the factor of two in this case

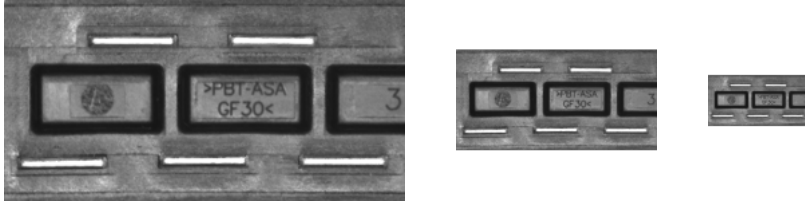


Figure 8.6: Example image pyramid.

Exercise: Compute three-level pyramid for an image from `plug.png` file. Use `DownsampleImage` filter - you will need more than one instance.

Pyramid Processing

Image pyramids can be applied to enhance the efficiency of the correlation-based template detection. The important observation is that the template depicted in the reference image usually is still discernible after significant down-sampling of the image (though, naturally, fine details are lost in the process).

To save computation time, we can identify match candidates in the down-sampled (and therefore much faster to process) image on the highest level of our pyramid, and then repeat the search on the lower levels of the pyramid, each time considering only the template positions that scored high on the previous level.

At each level of the pyramid we will need appropriately downsampled picture of the reference template, i.e. both input image pyramid and template image pyramid should be computed.



Figure 8.7: Template image pyramid.

Multi-angle Matching

Although in some of the applications the orientation of the objects is uniform and fixed (as we have seen in the plug example), it is often the case that the objects that are to be detected appear rotated, as demonstrated in **Figure 8.8**. In Template Matching algorithms classic pyramid search is adapted to allow **multi-angle** matching, i.e., identification of the rotated instances of the template.

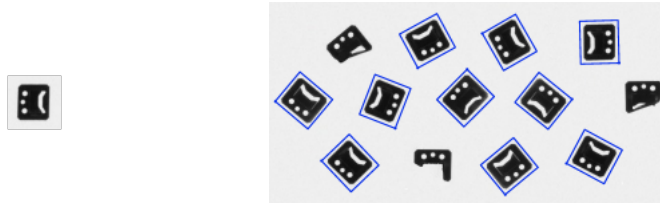


Figure 8.8: Results of multi angle matching of a single mount template.

This is achieved by computing not just one template image pyramid, but a set of pyramids - one for each possible rotation of the template. During the pyramid search on the input image the algorithm identifies the pairs (*template position, template orientation*) rather than sole template positions.

Similarly to the original schema, on each level of the search the algorithm verifies only those (*position, orientation*) pairs that scored well on the previous level (i.e. seemed to match the template in the image of lower resolution).

8.4 Edge-Based Matching

Edge-Based Matching enhances the previously discussed **Grayscale-Based Matching** using one crucial observation - that the shape of any object is defined solely by the shape of its edges. Therefore instead of matching of the whole template, we could extract its edges and match only the nearby pixels, thus avoiding unnecessary computations. In common applications the achieved speed-up is usually significant.

Matching object edges instead of an object as a whole requires slight modification of the original pyramid matching method: imagine we are matching an object of uniform black color positioned over uniform white background. All of object edge pixels would have the same gray intensity and the original algorithm would match the object anywhere wherever there is large enough blob of the appropriate shade of gray, and this is clearly not what we want to achieve.

To resolve this problem, in Edge-Based Matching it is the **gradient direction** of the edge pixels, not their intensity, that is matched.

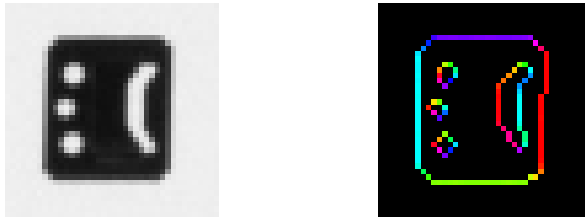


Figure 8.9: Different representation of templates in Template Matching algorithms. Edge-Based Matching template represents a direction as a color in HSV space for illustrative purposes.

Edge-Based Matching algorithm preprocesses both input and template image, extracting their edges and determining the gradient direction of each edge pixel. The results of the preprocessing are then stored implicitly as images² and matched using the same multi-angle pyramid processing method introduced in the Grayscale-Based Matching.

8.5 Filter Toolset

Adaptive Vision Studio provides a set of filters implementing both **Grayscale-Based Matching** and **Edge-Based Matching**.

²Each pixel representing a direction of the gradient vector at the given location, rather than brightness.


As the template image has to be preprocessed before the pyramid matching (we need to calculate the template image pyramids for all possible rotations), the algorithms are split into two parts:


Model Creation - in this step the template image pyramids are calculated and the results are stored in a template model - atomic object representing all the data needed to run the pyramid matching.

Matching - in this step the template model is used to match the template in the input image.


Such organization of the processing allows to compute the template model once and reuse it multiple times. For each Template Matching methods there are two filters, one for each step of the algorithm.

Grayscale-Based Matching

CreateGrayModel		
inImage		inPyramidHeight
inTemplateRegion		inMinAngle
inPointOfReference		inMaxAngle
diagPatternPyramid		inAnglePrecision
outModelPosition		
outGrayModel		

MatchGrayModel_NCC		
inImage		inMinScore
inSearchRegion		inMinDistance
inGrayModel		
diagImagePyramid		outAngles
diagMatchPyramid		outScores
outMatches		
outPoints		

Edge-Based Matching

CreateEdgeModel		
inImage		inPyramidHeight
inTemplateRegion		inLowerEdgeThreshold
inPointOfReference		inHigherEdgeThreshold
diagEdges		inMinAngle
diagEdgePyramid		inMaxAngle
outModelPosition		inAnglePrecision
outEdgeModel		


MatchEdgeModel		
inImage		inMinScore
inSearchRegion		inMinDistance
inEdgeModel		inEdgeThreshold
diagEdgePyramid		inIgnoreEdgePolarity
diagMatchPyramid		outAngles
outMatches		outScores
outPoints		

Table 8.3: Template Matching filters available in Adaptive Vision Studio.

Application Schema

The main challenge of applying the Template Matching technique lies in careful adjustment of filter parameters, rather than designing the program structure. Actually, as far as the simple scenario (given object reference image match its

8. TEMPLATE MATCHING

occurrences) is concerned, all Template Matching-based solutions are expected to follow the classic schema described below.

It is advisable to implement each of the (Model Creation, Matching) steps in a separate program. First program will apply the model creation filter and store the result on disk, while the other one will load the model and perform the actual matching. Doing so we will avoid the heavy yet one-off computation of the model in the matching program.

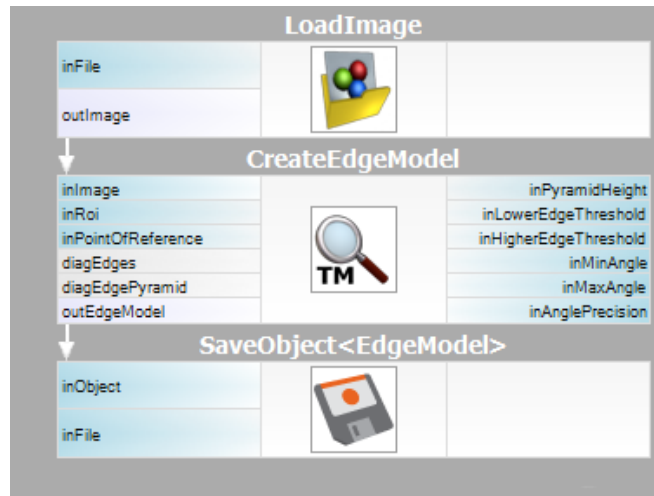


Figure 8.10: Model Creation phase of Edge-Based Matching solution.

After setting up the schema presented in **Figure 8.10** and **Figure 8.11** all that remains to be done is careful setup of the filter parameters. If we do everything right, the positions of matched object occurrences will be available at `outMatches` output of the matching filter.

Model Creation Parameters

This section explains how to adjust the parameters of Model Creation filters: `CreateEdgeModel` and `CreateGrayModel_NCC`.

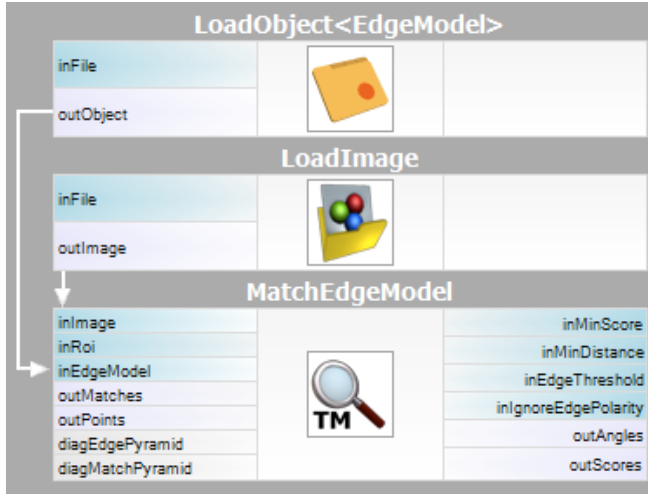


Figure 8.11: Matching phase of Edge-Based Matching solution.

Pyramid Height

The **inPyramidHeight** parameter determines the number of levels of the pyramid matching and should be set to the largest number for which the template is still recognizable on the highest pyramid level. This value should be selected through interactive experimentation using the diagnostic output **diagPatternPyramid** (Grayscale-Based Matching) or **diagEdgePyramid** (Edge-Based Matching).

In the example presented in **Table 8.4** the **inPyramidHeight** value of 4 would be too high (for both methods), as the structure of the template is entirely lost on this level of the pyramid. Also the value of 3 seem a bit excessive (especially in case of Edge-Based Matching) while the value of 2 would definitely be a safe choice.

Angle Range

The **inMinAngle**, **inMaxAngle** parameters determine the range of template orientations that will be considered in the matching process. For instance (values in brackets represent the pairs of **inMinAngle**, **inMaxAngle** values):

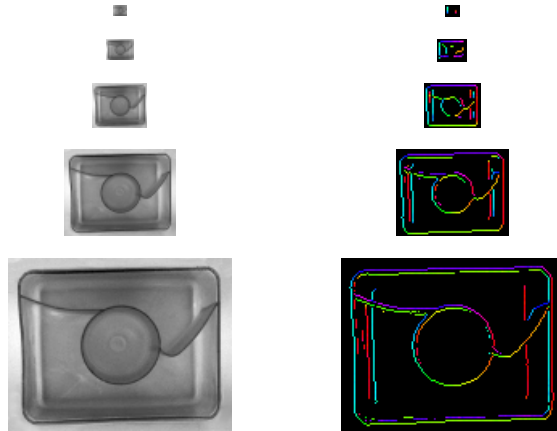
Grayscale-Based Matching Edge-Based Matching

Table 8.4: Image pyramids of the capsule template in both Template Matching algorithms.

- (0.0, 360.0): all rotations are considered (default value)
- (-15.0, 15.0): the template occurrences are allowed to deviate from the reference template orientation at most by 15.0 degrees (in each direction)
- (0.0, 0.0): the template occurrences are expected to preserve the reference template orientation

Wide range of possible orientations introduces significant amount of additional computation and increases memory usage, so it is advisable to limit the range whenever possible.

This parameter should be set through interactive experimentation to a value low enough to assure that all correct matches will be returned, but not much lower, as too low value slows the algorithm down and may cause false matches to appear in the results.

Edge Detection Threshold

The **inLowerEdgeThreshold**, **inHigherEdgeThreshold** parameters of **CreateEdgeModel** filter determines the settings of the hysteresis thresholding used to detect edges in the template image. The lower the values, the more edges will be detected in the template image.

These parameters should be set so that all the significant edges of the template are detected and the amount of redundant edges (noise) in the result is as limited as possible.

Similarly to the pyramid height, edge detection thresholds should be selected through interactive experimentation using the diagnostic output **diagEdgePyramid** - this time we need to look only at the picture at the lowest level. **Table 8.5** table presents an example analysis.

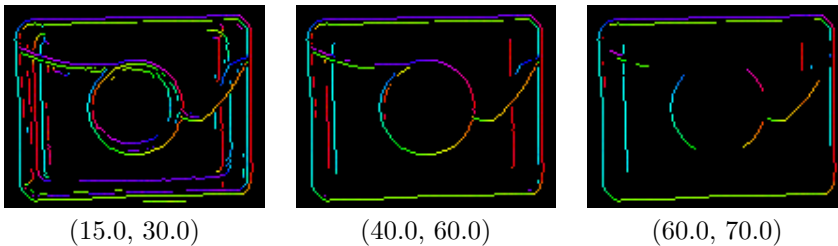


Table 8.5: Example results of model creation with various edge detection threshold values. The results in the middle seem optimal: left image exhibits excessive amount of noise, while in the image on the right significant edges were not found.

The **CreateEdgeModel** filter will not allow to create a model in which no edges were detected at the top of the pyramid (which means not only some significant edges were lost, but all of them), yielding an error in such case. Whenever that happens, the height of the pyramid, or the edge thresholds, or both, should be reduced.

Matching Parameters

Adjusting the parameters of the matching filters: **MatchEdgeModel** and **MatchGrayModel** is usually easier than the Model Creation setup. The most important parameter is the minimum score of accepted matches.

Minimum Score

The **inMinScore** parameter determines how permissive the algorithm will be in verification of the match candidates - the higher the value the less results will be returned.

8.6 Tips and best practices

How to select a method?

For vast majority of applications the **Edge-Based Matching** method will be both more robust and more efficient than **Grayscale-Based Matching**. The latter should be considered only if the pattern being considered has smooth color transition areas that are not defined by discernible edges, but still should be matched.

How to efficiently work in a loop?

If we are to run the matching repeatedly on a series of images (e.g. on image stream from a camera), we should ensure that the pattern model is loaded from disk only once.

To do so, we should encapsulate the part of the program that needs to be repeated for each image (the matching itself and processing of the results) in a macrofilter and keep the LoadObject filter that loads the model out of it. This issue is illustrated in **Figure 8.12** and **Figure 8.13**.

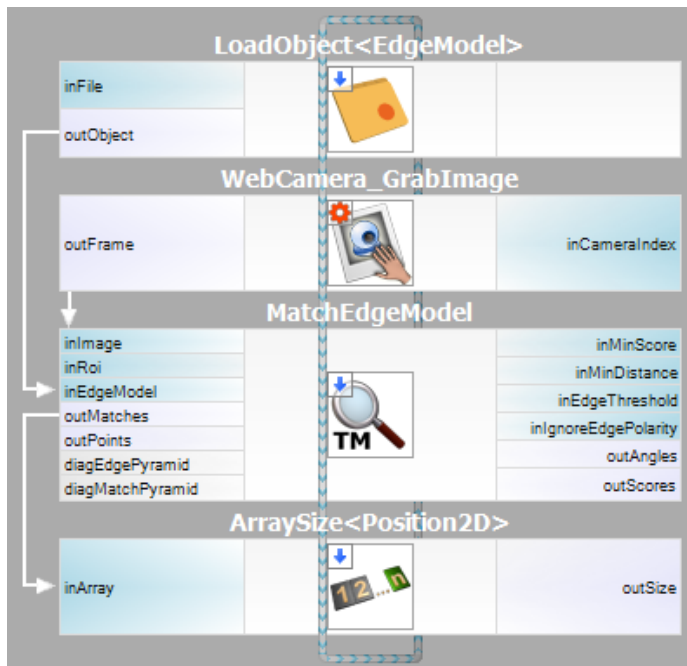


Figure 8.12: Inefficient schema - the model is loaded from disk in each iteration.

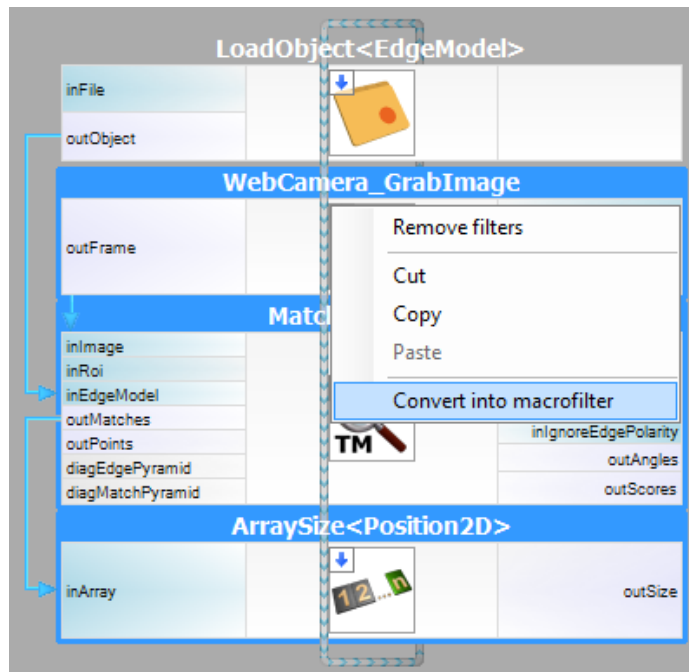


Figure 8.13: Solution - encapsulate the part to be repeated for each frame in a macrofilter.