Pattern Recognition in High-Throughput Zebrafish Imaging

Nezhinsky, A.E.

2 Segmentation and Pattern Recognition in Image Space

Based on:

A. Nezhinsky & F.J. Verbeek: Pattern Recognition for High Throughput Zebrafish Imaging using Genetic Algorithm Optimization. In: 5th IAPR Conference on Pattern Recognition in BioInformatics (PRIB 2010), Lecture Notes in Bioinformatics 6282: 302–312, Springer (2010)

and

A. Nezhinsky & F.J. Verbeek: Efficient and Robust Shape Retrieval from Deformable Templates. In: Leveraging Applications of Formal Methods, Verification and Validation. Applications and Case Studies (ISoLA 2012), Lecture Notes in Computer Science 7610: 42–55, Springer (2012)

2.1 Introduction

Retrieving location and contour of a shape is crucial for the analysis of large image datasets in many fields, such as optical character recognition (OCR) and bio–imaging. Segmentation is the decomposition of an image into foreground objects and background. Foreground objects are preferably the (biological) structures of interest, while other objects and noise are the background. In most cases objects of interest are homogenous regions and therefore most segmentation methods are based on homogeneity. Homogeneity of an object can be based on different characteristics, this can be color, texture or other similarity measures.

2.1.1 Related Work

Most segmentation techniques use a characteristic object feature in order to separate objects from the background. These features can be based on object shape or on color or texture and are attempting to mimic human visual system behavior. A successful segmentation turns a grayscale or color image into a binary representation in which one value represents the object of interest and the other the background.

The first step in many applications is to segment a meaningful biological structure from images and remove the background, despite its position, rotation and scale. Images in biology are acquired under different conditions which affects quality, brightness, contrast, lighting conditions and the amount of noise. Consequently, extraction as well as processing of shapes from the images might be hampered.

We can separate the segmentation techniques into two groups: bottom up segmentation and top down segmentation [4]. Bottom up segmentation uses image criteria in order to define image regions that are likely to be the structure of interest. In this case no machine learning or pattern recognition is involved.

The top-down approach on the other hand uses pattern recognition techniques that use predefined object representation in order to approximate the object location. Top-down methods are suggested to mimic human vision behavior and are becoming more successful in retrieving an object based on its predefined shape. On the other hand bottom-up techniques can get more precise local image boundaries [31]. Recent work [31, 4, 68, 22] stresses the importance of combining both methods exploiting their advantages.

Bottom-up Segmentation We first give a short overview of some bottom-up segmentation methods. Color and texture based approaches are intuitive and most common during low level segmentation.

Color based segmentation can be based either on gray scale images or on color. The common approach based on color value is *thresholding* an image with a single value. The thresholding method converts a gray scale image into a binary representation by a threshold value. The value is chosen either manually by the scientist or automatically [46]. During thresholding, pixels that have a grayscale value above the threshold value are marked as foreground. All other pixels are then marked as background. A number of variations exist on this approach. Thresholding can be done in an inverted manner,

such that pixel values below the threshold value are treated as foreground, the rest as background. Additionally multi thresholding methods exist, where a pixel is set as foreground if its value is in between two threshold values.

For color images these values first must be converted to gray scale or separated into different channels (RGB or HSV [21]) and then thresholded. This approach works well under the assumption that the object of interest is of a characteristic color that is significantly different from the color of the background or other objects in the image.

Another approach is to look for region similarity by applying region growing techniques within the *region-based* methods. A state of the art color region based method is Mean Shift discontinuity preserving altering [59]. It first determines the clusters in the image by merging regions iteratively based on color similarity, then it assigns class labels to formed clusters. If the Mean Shift is used with a generalized kernel it can be seen as a nonparametric algorithm. Another widely used and adapted region-based method is the watershed segmentation [12].

Besides color characteristics objects can be characterized by their edges. An edge location within each image is efficiently represented by the magnitude of the gradient. For each (pixel) position (x, y) in the image we have $\mid \nabla f(x, y) \mid$ as the magnitude of the gradient by usage of Sobel, Previt or Roberts filter [21] on the edge map. A binary representation can be also found directly by applying a Canny edge detector. An edge map represents shape boundaries. Alternatively edges can be found by considering zero-crossings of the Laplacian-of-Gaussian [53]. After the edges are identified the image can be separated into objects by using connected components labeling [50] combined with thresholding. These edge-based methods can only perform well if edges are clearly present in the image and no objects are overlapping one another.

A more elaborate list of (low level) segmentation techniques is given in [38].

Top-down Segmentation (Pattern Recognition) An important problem in image analysis field is the content retrieval of objects from images. While the bottom up segmentation techniques can be used to separate foreground from the background, it is not enough for annotation of objects within the foreground. To accomplish that certain pattern recognition based on a shape model are required.

Most widely used methods are: active shape models (ASM) [9], active snakes (or contours) [26], the Hough Transform [52], deformable templates [65] and level-set based methods (LSM) [51]. Most model based segmentation methods are concerned with the retrieval of certain predefined shapes from the images.

A large number of context based image retrieval systems have been described [70]. These can be divided into the *free-form* and the *parametric* approaches [24].

Free Form Free-form class models have no global shape limitation, but focus mainly on attraction towards certain image features [24]. Free form models need a correct global initialization inside an image and optimize the local shape. A popular *free-form* approach for shape retrieval is the Active Contour (or active snake) [26]. Shape edges need to be connected together for an active snake to perform correctly.

Parametric In papers dealing with recognition of objects with known shapes, [18, 44, 16] the authors stress the use of deformable templates. Deformable templates are part of the *parametric class* models as they make use of a predefined set of parameters. The representation of the parameters might differ, but often a template is used, which consists of a set of contour points to which we approximate the basic shape outline. A deformable template [24] approach gives the possibility to represent an object that can be subject to certain deformations in a compact manner. In that case object localization should be performed by a process of matching the deformable template to the object shape in the input image. In [9, 27, 45] a deformable template representation of a shape is a set of points approximating the outline as obtained from a priori knowledge.

Existing template matching systems are known for handling grayscale or binary images. Some systems work directly on the grayscale input image. In this case often shape descriptors like SIFT [37] are used to describe shape or texture characteristic. Other systems assume a *bottom-up* segmentation step like thresholding [21, 46] already converted the image to a binary representation. In [33] the authors propose a deformable template based region merging system. As input their method uses both *over segmentation* with region color and edge detection and is based on splitting and merging regions based on certain perceptual criteria.

2.1.2 Overview of Basic Approaches for Zebrafish Embryo Segmentation

We have tried to apply different bottom up segmentation methods in order to separate the zebrafish shapes from the background. Note, that this concerns solely segmentation, not recognition. This means the separation of foreground (the space taken up by zebrafish embryos) and the background of an image.

The motivation to apply color or grayscale based segmentation is based on the suggestion that the zebrafish embryos have a different color than the background. In order to determine the color that defines a zebrafish embryo automatically, we first attempt to determine the color of the background. This can be done under the assumption that a zebrafish embryo is never located on or intersecting the edge of the image. This means that all the pixels on the edge of the image are part of the background. All image objects that are not of that color are then treated as zebrafish.

First we attempt to do the segmentation based on grayscale thresholding. The most basic approach would be global thresholding. This means that image pixels with an intensity value below a threshold value t are labeled as foreground and the rest of image space is labeled as background. A global threshold method does not work for most of images at hand. This was due to the fact that because of uneven illumination the background intensity differed on different locations in the image and therefore there was no t value that could be set to separate foreground from background. An example of failing global thresholding is shown in Figure 2.1.

We also attempted to interpolate the grayscale background values from the edges to the center of the image. By this we tended to predict the background values as they should be at the location of the image and then subtract them from the original image. A commonly used way of doing it would be using a large Gaussian filter. However, since

Figure 2.1: From left to right: Original image (converted to grayscale) , thresholded grayscale image.

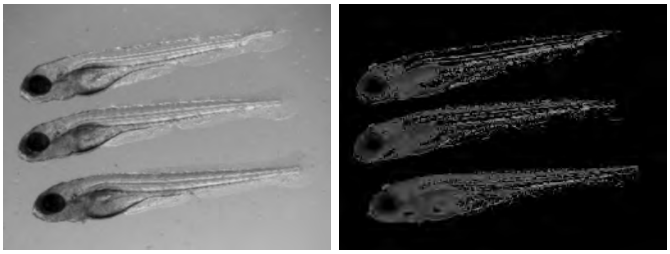

Figure 2.2: From left to right: original image, difference of interpolated image and original.

the zebrafish objects are very large this approach would not work. We applied a *bi-linear interpolation* method to be able to predict the background values of each pixel within the image. Instead of pixels we separated the image into small $a \times a$ rectangular areas and calculated their average intensity. If the difference between this intensity and the interpolated image was larger than a certain threshold t, area was treated as foreground, otherwise as background. Foreground pixels were multiplied with the original image, background area intensity was set to 0. In Figure 2.2 we show some results of this algorithm. While the results seem good there is an amount of variables (t, a) that still need to be set by hand per image before running the algorithm .

Following the grayscale threshold we have tried to separate the image based on its values in the color spaces. Again, we assumed that the edge of the image contains only background values. We have separated the image into HSI space, which is a more standard way to do segmentation on color images than separating into RGB channels. We have followed a typical HSI segmentation method as is presented in [21](page 331). The method is based on thresholding the saturation image S. Then the retrieved mask S_b was multiplied with the hue image H. The saturation threshold value is set based on the edge background values. An example of this method applied to a typical zebrafish image is shown in Figure 2.3. The spatial location of non black pixels in $S_b \times H$ represent the regions of interest. This segmentation is not accurate, since a lot of pixels within the zebrafish and on its border are treated as background.

Figure 2.3: From left to right: original image (converted to grayscale), its saturation component (S), thresholded saturation component multiplied by hue ($S_b \times H$).

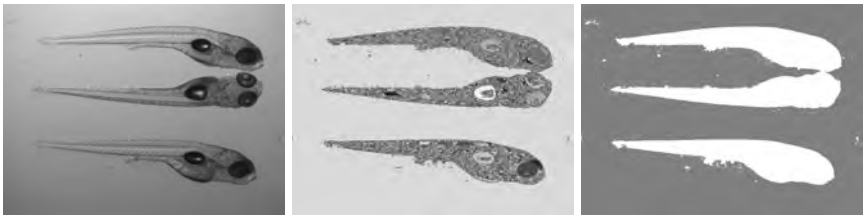

Figure 2.4: From left to right: original image, regions retrieved by mean-shift segmentation, retrieved background region in gray.

Another method based on color is Mean Shift discontinuity preserving altering [59]. The method progresses throughout the image by looking for color similarities. Then the image is labeled into regions. We have applied the method to the zebrafish images by assuming the entire background will be selected as one region. This region can then be identified since we know the background is present at the edge of the image. The result of such a segmentation is shown in Figure 2.4. As can be seen this method gives more accurate results.

In our test case the image background represents a liquid and therefore it is fairly smooth without high local color changes. The zebrafish objects however usually do have strong boundaries. This suggests the use of edge based methods. We have tested our data with several straightforward gradient operators. An operator that directly gives a binary image as output is the Canny edge detector. Its output is shown in Figure 2.5. As can be seen in the figure edge based methods are able to remove the background. However, since the zebrafish embryos have a lot of strong edges present within the fish it still might be difficult to automatically select the zebrafish outline.

Some of the approaches presented here were used as a preprocessing step during the actual pattern recognition process. The choices of these approaches are described in more detail in the following sections. A short summary on the methods as applied to zebrafish embryos is given in Table 2.1.

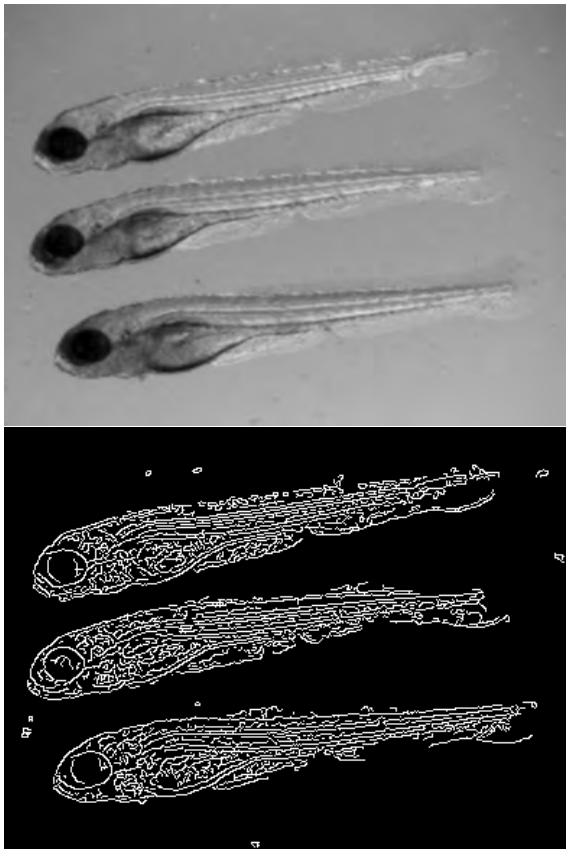

Figure 2.5: From left to right: Original image converted to grayscale, right canny edge detector result.

Segmentation technique	Discussion
binary thresholding	fails at uneven illumination
background interpolation	good, but variables need to be set per case
HSI space based	good, if zebrafish intensity differs from background
mean-shift	good results
edge based	good results, but needs post-processing

Table 2.1: Comparison of different bottom up segmentation methods with respect to zebrafish embryo retrieval.

2.1.3 Deformable Template Matching

The problem at hand is the retrieval of a known basic shape: the basic shape is known but can slightly vary. Besides separating foreground from background also content retrieval is important: we want to know how a shape is located, how many shapes are in the image and where which part of the shape is. This calls for a top down approach. A candidate would be a parametric approach. Template matching is a process of finding a predefined template in an image. The way template similarity as compared to a part of the original image is calculated differs per application [32].

2.2 Deformable Template Matching (Approach 1)

In this section we introduce a slice representation model (SR) of a deformable template. Instead of considering the outline of a shape, we simplify the shape representation by considering only certain characteristic horizontal slices and use these for template matching. The proposed SR model is made advantageous for efficient optimization with a Genetic Algorithm (GA). In [66, 48] authors also propose a GA for low parameter shape templates (circle and ellipse detection).

2.2.1 Method

We propose a template representation which captures both boundary and interior of the object and thereby describes the average structure of a predefined shape. The grid of the search space is determined by the discrete pixels in a $K \times L$ image matrix M. We take a binary image as starting point. The binary image is obtained by thresholding. Thereby we assume a background pixel has the value 0 and a foreground pixel value 1. In the following subsections we will describe the template representation in more detail as well as the deformations that a template can undergo.

Slice Representation Model

The prototype template T_0 we propose is represented by the following vector:

$$T_0 = (s_0, s_1, \ldots, s_n, d), \tag{2.2.1}$$

where n is the number of slices; d is the distance between two slices in the horizontal direction. Initially T_0 is represented in a X(horizontal)–Y(vertical) space in the horizontal direction, with slices being parallel to the Y-axis. This is done for ease in the representation. A template slice itself is then represented by the vector:

$$s_i = (w_i, b_i, \varsigma_i), \tag{2.2.2}$$

where w_i is the fixed width in pixels in the horizontal direction, preferably $w_i = 1$; b_i is the fixed width in pixels in the vertical direction of the image, preferably $b_i = 0$ and located below and above the area of w_i. The b_i are required to define that the area surrounding the shape is preferably empty. And ς_i is the seed point of a slice; ς_i is

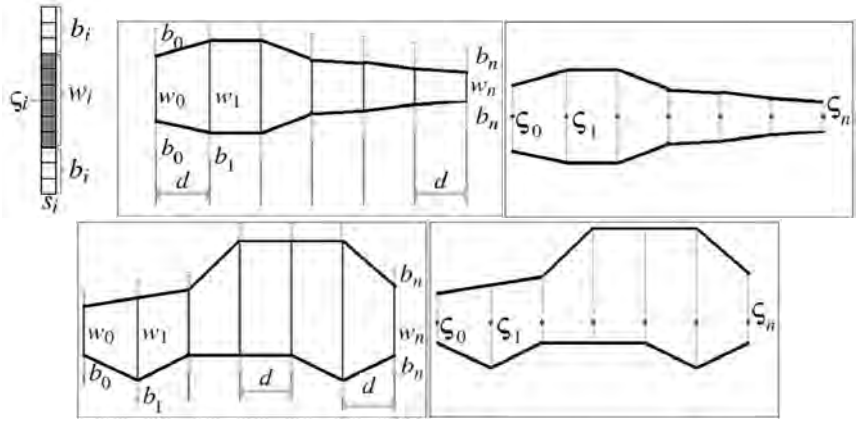

Figure 2.6: a) A single slice. The slice is always parallel to the Y-axis b) Template for a fish shape c) Location of ς_i in a fish shape template d) Template for a car shape e) Location of ς_i in a car shape template

needed to be able to define non symmetrical shapes as input. This point will be used as a reference for allowed slice shifts. In Figure 2.6a an example slice is shown; a slice is a sample of the template always perpendicular to the length axis.
In Figure 2.6b, 2.6c, 2.6d two examples of templates are shown. The template length is fixed according to $\ell(v) = n * d$. For templates with horizontal symmetry axis, ς_i is simply chosen as the center of each slice (cf. Figure 2.6c). For templates having no horizontal symmetry axis ς_i are located on the same horizontal line (cf. Figure 2.6e).

Deformations

This representation was chosen as we assumed the slices will be able to move vertically along the template to match a deformed (slightly shifted) shape. At template shift or slight rotation the global shape is still captured within the template. The total length of the template is fixed in the proposed representation.
A deformed template T is derived from the prototype template T_0 and is represented as $T(T_0, I)$. A deformation I will then be encoded by a state-sequence as follows:

$$I = (x, y + \delta_0, y + \delta_1, \ldots, y + \delta_n), \tag{2.2.3}$$

where x is the shift in the X-axis direction of ς_0; $y + \delta_i$ is the translation measured in the Y-axis direction of the ς_i of slice i.
We assume the maximal vertical deformation between two slices is 45 degrees, cf. Figure 2.7a. As a result of this constraint $|\delta_{i-1} - d| < |\delta_i| < |\delta_{i-1} + d|$ applies for two starting points of consecutive slices i and $i + 1$. Then, each ς_i can be shifted vertically within the following boundaries:

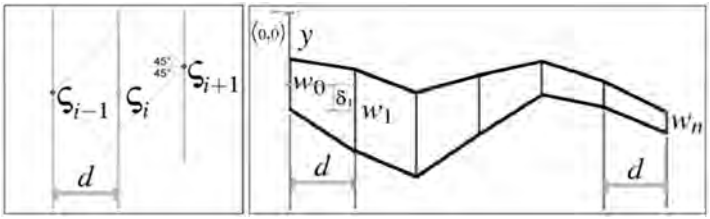

Figure 2.7: a) Vertical shift margins allowed for slice starting point ς_i, relative to ς_{i-1} and ς_{i+1}. b) A possible deformation for a fish template.

$$\max(\varsigma_{i-1} - d, \varsigma_{i+1} - d) < \varsigma_i < \min(\varsigma_{i-1} + d, \varsigma_{i+1} + d). \tag{2.2.4}$$

In Figure 2.7b we demonstrate a possible deformation of a template representing a fish.

Energy Function

The energy function Φ is a fitness function that specifies to what extent an identified shape in the image matches the deformed template. In the literature the probability of a deformed template being located over a shape is referred to as the likelihood [24].
Let us now introduce the details of the computation of Φ. Consider the fitness ϕ_i of a single slice i. In order to find the optimum we wish to maximize matching values (0 or 1) between pixels covered by the slice, i.e., pixels with value 1 contained in w_i and pixels (at top and bottom of the slice) with value 0 in b_i. We then have: $S_i[j]$ represents the j-th element (pixel) of slice S_i, as counted from the top of the slice.

$$\phi_i = \frac{1}{w_i + 2b_i} \sum_{j=0}^{w_i+2b_i} H_j\,,\; H_j = \begin{cases} 1, & \text{if } M(x, j + y + \delta_i) = S_i[j] \\ 0, & \text{otherwise} \end{cases} \tag{2.2.5}$$

In Figure 2.8 an example of a matching is given. The location of the proposed matching is denoted in pattern. For this example:

$$\phi_i = \frac{1+0+0+1+1+1+1+1+0+1+1+1}{12} = 0.75$$

The deformation I consists of multiple slice shifts. The total fitness of all n slices is given by:

$$\Phi = \frac{1}{n+1} \sum_{i=0}^{n} \phi_i. \tag{2.2.6}$$

2.2.2 Optimization by a Genetic Algorithm

In this section we discuss the major components of a Genetic Algorithm (GA), i.e., population, evaluation, selection, crossover and mutation respectively. In a GA, a population

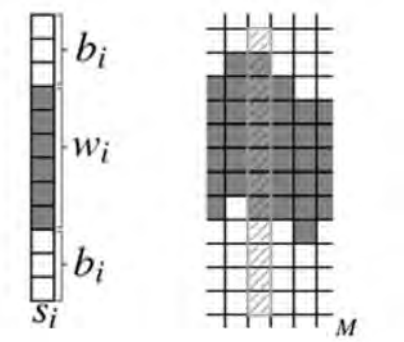

Figure 2.8: Example of matching a slice S_i on to a location in matrix M

P (of size m) of candidate solutions is evolved toward better solutions by introducing computer analogues for recombination, mutation and selection [1].

A candidate solution is also referred to as an *individual*. The outline of a generic GA pseudo code reads:

1. $t = 0$
2. Initialize $P(t)$
3. Evaluate $P(t)$
4. **while** not terminate **do**
5. $P'(t) =$ SelectMates $(P(t))$
6. $P''(t) =$ Crossover $(P'(t))$
7. $P'''(t) =$ Mutate $(P''(t))$
8. Evaluate $P(t)$
9. $P(t+1) = P'''(t)$
10. $t = t + 1$
11. **end while**

The termination criterion can differ, depending on the problem at hand. The details of the operators are dealt with in more detail in the following subsections.

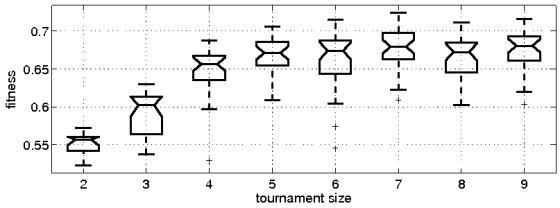

Figure 2.9: Fitness comparison for different tournament values in 30 runs

Representation of Individuals

A shape I based on the template T_0, also referred to as *individual*, is represented in the image as, and consists of the following genomes:

$$I = (x, y + \delta_0, y + \delta_1, \ldots, y + \delta_n). \tag{2.2.7}$$

The individuals are initialized with randomly valued genomes. According to common practice in the GA research, a population is generated with random genome values, covering the entire range of possible solutions. For finding shapes in images, this means random shapes are initialized on random locations in the target image with a random deformation according to Eq. 2.2.7.

Evaluation

The fitness function determines the quality of an individual and depends on the problem at hand. Function Φ (cf. Eq. 2.2.6) will serve as a fitness function for the genome values of an individual; Φ is then used to evaluate the candidate solutions in the selection step.

Selection

For the selection operation, the so-called tournament selection scheme [39] is used in order to prevent premature convergence. Tournament size, i.e., k_{size}, was determined through empirical testing. Consequently, comparison of high and low k_{size} is given in Figure 2.9. We have established $k_{size} = 7$ to be a good value for our experiments.

Crossover

A crossover is applied with single crossover point. A random point $p \in (0, n)$ is chosen where n denotes the length of the genome. After crossover of two individuals I_J and I_K the resulting individual I_L has the following form:

$$I_L = (x_K, y_K, \delta_{0_K}, .., \delta_{p_K}, \delta_{(p+1)_J} + a, .., \delta_{n_J} + a). \tag{2.2.8}$$

Variable a is needed to make sure Eq. 2.2.8 holds and is determined by:

Figure 2.10: Graphical representation of crossover function as derived from the data. Typical example in zebrafish imaging.

$$a = \begin{cases} \delta_{p_K} - d, & \text{if } (\delta_{p_K} - \delta_{(p+1)_J} > d) \\ \delta_{p_K} + d, & \text{if } (\delta_{p_K} - \delta_{(p+1)_J} < -d) \\ 0, & \text{otherwise} \end{cases} \tag{2.2.9}$$

Mutation

For the mutation operator we use standard settings commonly used for GAs. That is a uniform mutation, where each genome g has the probability to mutate: $1/n$ [1], where n is the genome length. For each genome:

$$g\ U(\bar{g}, \underline{g}) \tag{2.2.10}$$

To make sure a uniform mutation is used we allow every slice center to mutate anywhere within the image space. Since the constrain $\max(\varsigma_{i-1} - d, \varsigma_{i+1} - d) < \varsigma_i < \min(\varsigma_{i-1} + d, \varsigma_{i+1} + d)$ holds, subsequent slice centers are not allowed to be more separated than distance d.

2.2.3 Multiple Object Recognition

The shape under study can have multiple slightly different (deformed) instances in one and the same image. The complete search space in this image is denoted by M_0. First the best matching shape S_0 (with highest fitness) is localized. To decrease the search space for finding the next shape instance, we set all the elements contained within $S_0 \in M_0$ to 0 and name the resulting image M_1.

This process is repeated by iteration and in that manner M_i is reduced for each next identified shape i. No shapes are found if the fitness of the found optimum $F(S_i)$ drops drastically, under a predefined threshold value r. The value of r can be determined empirically from a test on similar images (acquired under the same conditions). An example of such drastic fitness drop in fitness growth is depicted in Figure 2.11. This is a fitness plot for an image containing 3 fish shapes (cf. Figure 2.15a).
The pseudo code for finding multiple shapes in an image can be written as:

1. $i = 0$
2. $S_0 = GA(M_0)$
3. **while** $F(S_i) > r$ **do**
4. save S_i as found shape

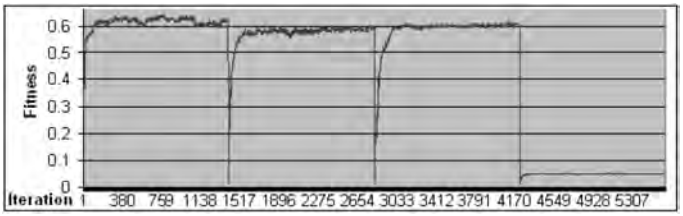

Figure 2.11: Fitness plot for an image containing three zebrafish shapes.

5. $M_i = M_{i-1} \backslash S_i$

6. $i = i + 1$

7. $S_i = GA(M_i)$

8. $t = t + 1$

9. **end while**

2.2.4 Experiments

To evaluate the performance of our template representation and GA optimization we have designed the task of finding objects based on predefined slice templates in images with different type of content. An experiment was performed with templates i.e. in synthetic as well as microscope images. With a simple interface the user selects both input images as well as the template shape for analysis.

Testing with Synthetic Images

We have used 20 synthetic binary images of 388×291 pixels. We have generated the images with different shapes located at different locations in images, slightly rotated, skewed and missing pixel data. Random noise (drawing debris) is introduced. The images randomly contain up to 3 instances of an object.

The first template used for the synthetic shapes is a template of an animal figure presented in Figure 2.12. We have created a synthetic binary image containing one deformed animal shape. The result of the application of the GA optimization is depicted in Figure 2.12.

The second template used for the synthetic shapes is a template of a house figure presented in Figure 2.13.

We have created a synthetic binary image containing two figures that have a deformed house shape. The result of the algorithm (implementation done in Delphi) is shown in Figure 2.13.

Figure 2.12: Very simple template of an animal shape (head, legs and body) and the result of shape localization in a synthetic image containing a simple animal shape instance.

Figure 2.13: Very simple template of an house shape and the result of shape localization in a synthetic image containing two simple house shapes.

Application to the Zebrafish Larvae Images

To evaluate the performance of our algorithm in a real-world imaging application we have chosen the task of finding zebrafish embryo shapes. The task at hand concerned using a High Throughput (HT) segmentation technique for retrieving the location and number of zebrafish embryo objects within images [54]. An additional requirement for this application is the need for automatic recognition of head, body and tail of each embryo. A typical binary image as presented for localization is shown in Figure 2.15a. This image was converted from color scale images to binary in a preprocessing step. We use a straightforward gradient operator (Sobel) followed by an iso-data threshold. In that way strong edge pixels were extracted for a binary representation [21].

Each image contains multiple zebrafish embryo shapes. The number of shapes in an image is not known in advance, the shapes might be overlapping. The shapes in all the images are assumed to be located approximately horizontal with a maximum angle of 45 degrees; so our approach could be used. We have applied the algorithm for a database of 100 images with the same settings for the GA ($m = 200$, $k_{size} = 7$, $r = 0.5$). For 87 images the amount of embryos in the image and the approximation of their shape could be retrieved correctly. In the cases where the algorithm failed, it was mostly due to large occlusion overlap or shapes being much longer or shorter than the proposed template. For the small and medium occlusions and overlap the algorithm performed correctly (cf. Figure 2.15). In Figure 2.14b the template of a zebrafish used in these results is

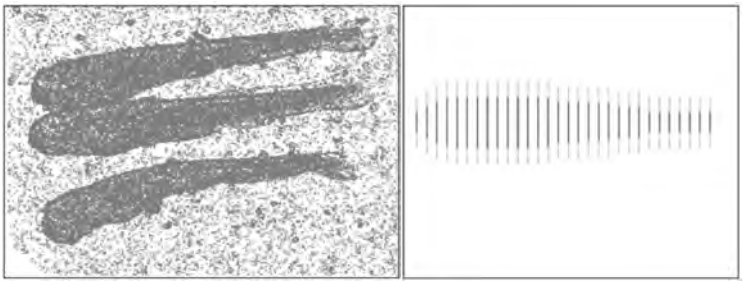

Figure 2.14: a) Typical binary image of zebrafish embryos in a resolution of 388 × 291 pixels. b) The template T_0 in a graphical representation that has been used on 100 tested images. Dark gray lines represent w_i and light gray lines represent b_i within slices. Some results are shown in Figure 2.15.

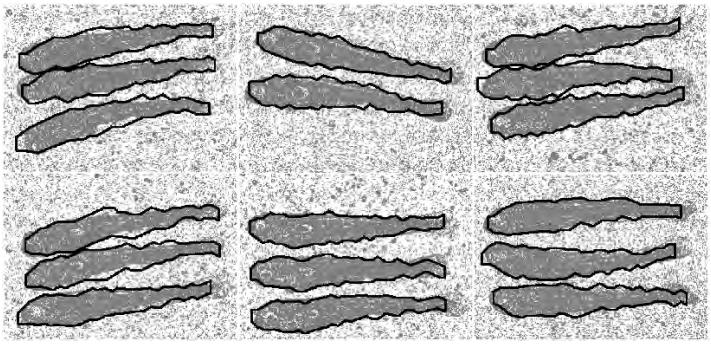

Figure 2.15: Graphical representation of template matching results for 6 images containing zebrafish larvae shapes.

shown in a graphical representation.

2.2.5 Conclusions and Discussion

In this section we have illustrated an application of a GA to optimize a deformable template approach. This approach can be used in different fields as the template can represent different shapes. Our approach was designed for an application in the HT screening of zebrafish embryos [54]. The optimization of template parameters is done through a Genetic Algorithm, which provides the possibility to search for an optimal solution in large search spaces. Our approach also allows to retrieve multiple instances of a certain object in a single image. Results indicate that this approach has a low error rate while computational performance is manageable and fast, which is, of course, very suitable for HT applications. Future work is directed towards a further generalization of the approach and making the template representation scalable.

2.3 Deformable Template Matching (Approach 2)

In the previous section we have illustrated a deformable template approach for the representation of a biological shape in a binary representation. We take this approach as a basis and further investigate the possibilities of a deformable template approach. We consider a representation on the basis silhouettes or boundary representations of prototype templates.

In research dealing with recognition of known shapes [18, 44, 16] the use of deformable templates is emphasized. Deformable templates typically are parametric class models; they start from a set of predefined parameters. The representation of the parameters might differ, but often a template is used, consisting of a set of contour points to which the basic shape outline is approximated. So, if the basic shape that is looked for is known, it still needs to be localized in the images. Therefore the prior knowledge can be exploited by choosing the parametric deformable shape template matching method as a basic approach [24, 70, 7]. Such an approach is used in the segmentation of cells in microscope images [18]. In this application the Hough transform approach is reformulated to be used as a deformable template. However, if the shapes are more complex than a circular shape like object, it is difficult to adapt to this approach. An example of a more complex shape is the segmentation of the masseter on the basis of a predefined template [44]; locally deformed instances of the template can be successfully extracted from input images. On the basis of this approach we propose a further generalization with which it is also possible to deal with multiple objects in one image as well as with global deformations; e.g., bending of the entire object. Based on silhouettes or boundary representations of prototype templates a considerable amount of research has been completed [16]. Usually the silhouettes are defined by contour points and make up the template. These templates can then be deformed by a set of parametric transformations, including both local and global transformations [70]. We have taken this traditional representation as a starting point; however, we have replaced the contour points in the silhouette by a contour area (cf. Figure 2.17). The reason for using this representation is that we would like to allow multiple overlapping instances of the object in one input image and therefore we have to accommodate for missing contour points.

In addition to the template matching, we also address the problem of shape normalization; in particular for applications of biological objects. The combination of shape localization and normalization has been successfully applied for the roundworm, i.e., C. elegans [47]. It is known as the BDB+ method. On the basis of the object boundary a straightening is applied so as to ease the further analysis of the objects. In our application, however, this method cannot be used; it starts from a predefined shape and then straightens the shape assuming the boundary has already been extracted. We want to investigate the recognition and straightening of more complex elongated shapes and, as indicated, account for the presence of multiple instances in one image.

The framework that we have elaborated consists of two steps. First, a preprocessing step including a segmentation of an input image in order to separate the object(s) of interest from the background is applied. Segmentation alone, however, does not give satisfactory results, as we are not only interested in separating background from fore-

ground, but we also want to recognize position and best possible representation of the object. This is realized in the second step consisting of a matching of a deformable template to the segmented image. This step is the main focus of this chapter. Finally, a post-processing step includes shape normalization through straightening of the extracted shape. Such is possible from contextual information about the object in the image that we have gained. Deformations are known and therefore deformations can be normalized according to the template. The framework was implemented in `C++` using the *OpenCV* graphics library (*http://opencv.willowgarage.com*).

2.3.1 Method

The starting point of our algorithm is variation; i.e. a shape has variation, it can be inflicted with noise and it can be deformed or partially occluded. Our framework detects deformed instances of a predefined structure by means of deformable template matching and these instances are subsequently extracted from an input image. In Figure 2.16, an overview of the process is presented. The approach consists of two steps: the preprocessing step and the template matching step. First, during the preprocessing step, the input image is converted to a strict binary representation. In the main process the deformable template matching is applied in the binary image obtained from the preprocessing. This entails looking for the best match of a prototype template in the image. If a match is found the result is annotated according to the prototype template and henceforth, straightened.

Pre-processing

The first step in the analysis is retrieving foreground and background: i.e., an operation that converts an input image to a binary representation by marking the pixels which belong to foreground objects 1 and the background pixels 0. Different binarization methods are described in the literature; i.e., based on the usage of global or adaptive threshold methods, color or edge based segmentation. The choice of the method depends on the input image at hand, its properties and quality [21]. In the cases where prior spatial information is known this given can be exploited and the threshold value set can be based on this given.

After separation of foreground and background in the image, the contextual information still needs to be retrieved in order to recognize the objects of interest.

Prototype Template

The initial contour sketch of the object of interest is defined by the prototype template T_0. The construction of T_0 is based on prior knowledge and is an approximate representation of how a typical object contour should look like and represents a contour location area. A few examples are shown in Figure 2.17.

A contour location area represents the region of interest within which the local template contour can exist and change. By doing so, the local deformations become limited

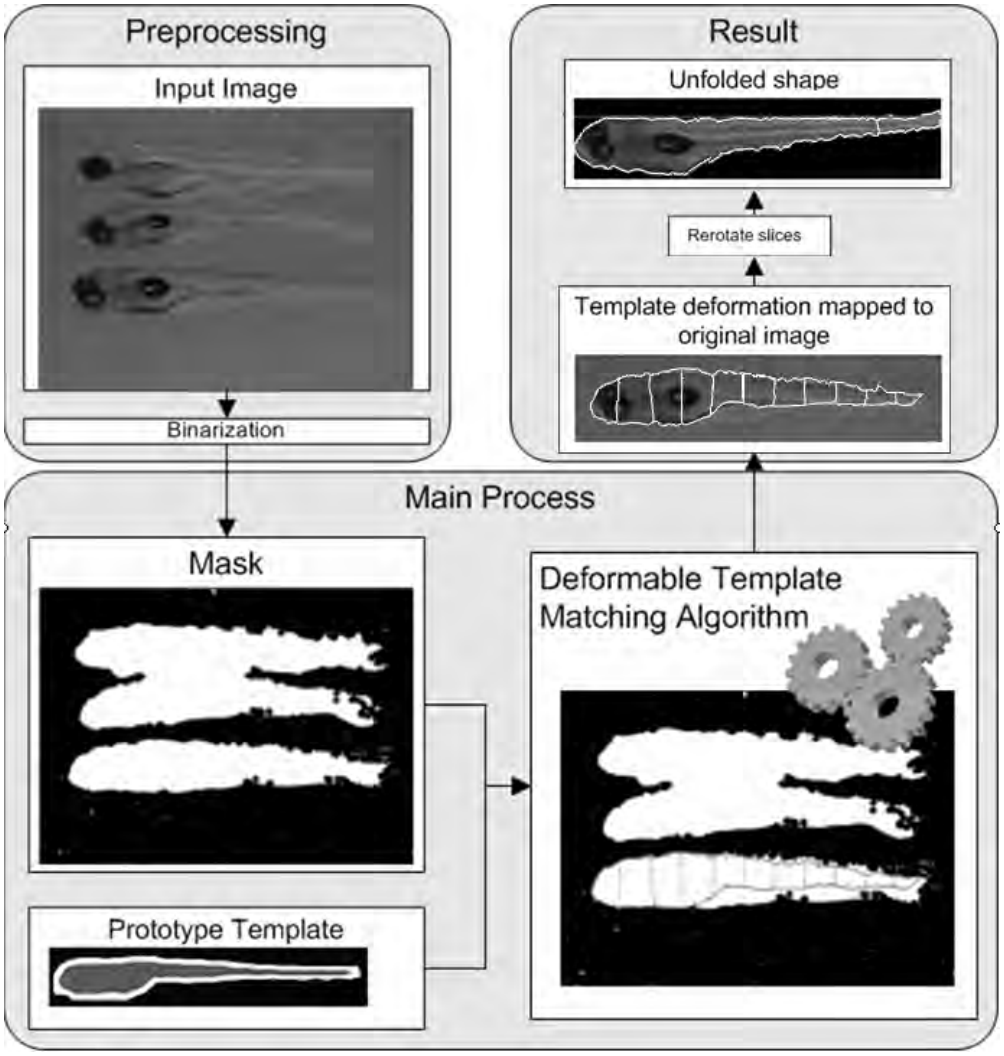

Figure 2.16: Proposed framework for automatic shape retrieval and straightening.

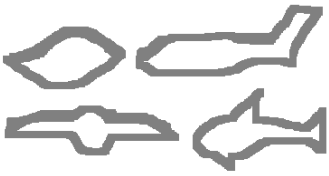

Figure 2.17: Some examples of prototype templates of different objects. Gray area represents the space where connected component boundaries might be located.

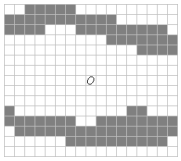

Figure 2.18: An example of matrix representing a template slice $t(i, j)$. Fields with value 1 are marked with gray color. Other fields have the value 0.

Figure 2.19: A prototype template as a chain of slices and a deformed instance. All shape boundaries that fit in the grey area fit the template. Black lines represent two example shapes that fit these templates.

by looking only at the pixels that are located within the contour location area (depicted in gray in Figure 2.17). Limiting the template boundary location of the deformed template is necessary to predict image boundary location in cases where it is missing, incomplete or overlapping.

As a result of this representation only global deformations are left, which will be described in the next subsection.

Parametric Transformation

Biological shape instances are often bended and rotated. To cope with these global deformations T_0 is distributed into n smaller sub-templates, so called slices t_i:

$$T_0 = t_0, t_1, \ldots, t_n \tag{2.3.1}$$

A single slice can be seen as a rectangular matrix $t(i, j)$, consisting of binary values. This is shown in Figure 2.20.

The horizontal medial axis of the slice $O_{horizontal}$, is defined at $[i/2, *]$, and the slice origin as O at $[i/2, j/2]$.

Within the whole template the slices can rotate around their origin O to allow matching against rotated shape, but the origins are linked together as a chain (Figure 2.19); at any deformation of the total shape the distance between sequential slice origins remains the same.

A deformed template T' is derived from T_0 and is represented as $T' = (T_0, \underline{\xi})$. A deformation ξ of the slice chain is encoded by the following state-sequence:

$$\underline{\xi} = (x, y, \alpha_0, \alpha_1, \ldots, \alpha_n) \tag{2.3.2}$$

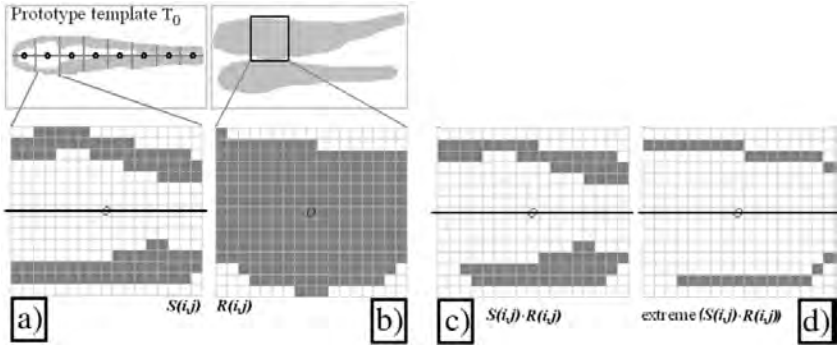

Figure 2.20: An example of template matching of a template slice $S(i, j)$ and a region $R(i, j)$.

Here x is the shift in the X-axis, y the shift in the Y-axis direction of the first slice t_0 and the angle of rotation α_i of each slice t_i. Due to the proposed slice based representation our deformable template approach is very suitable for use with elongated shapes.

Objective Function

The fitness of a template matching of an input image is measured by an objective function [24]. In [47] a parametric representation is used where the algorithm marches along the backbone of C. elegans to calculate the objective function.

A similar approach is applied, by comparing simultaneously the binary slice matrix $S(i, j)$ to a selected image region $R(i, j)$ of the same size (cf. Figure 2.20a,b). First the matrix is considered where both the template and the image region have overlapping foreground pixels, which are the result of $R(i, j) \cdot S(i, j)$ (cf. Figure 2.20c).

Since during this step the shape is a filled binary object, matches that are farthest from the slice center, yet still in the silhouette contour area are assumed to define the object. Then to get the actual border the algorithm marches along the horizontal medial axis of $R(i, j) \cdot S(i, j)$ iterating over 0 till j. Each orthogonal image plane pixel column p_i (with iterating from 0 to i) is compared to the template. It is assumed that the fish silhouette has only one silhouette pixel in the top and the bottom of each column. Therefore, per column the two *extreme* points that are of value 1 (as measured from the horizontal medial axis) remain 1, all other values are set to 0 (Figure 2.20d). The value of $extreme(S(i, j) \cdot R(i, j))$ is then the total amount of overlapping outer shape pixels between $S(i, j)$ and $R(i, j)$. This is treated as intermediate result. The quality (the objective function) of this result is then measured by the length of the retrieved border. The objective function is 1 if all pixels of the silhouette have been retrieved. Therefore the objective function for a slice $S(i, j)$ is:

$$\phi(S(i, j)) = \frac{0.5}{j} extreme(S(i, j) \cdot R(i, j)) \tag{2.3.3}$$

Since a template consists of n slices of the same size, $F(T, \underline{\xi})$ depends on the fitness function of each slice k:

$$F(T, \underline{\xi}) = \frac{1}{n} \sum_{k=0}^{n} \phi_k(S(i, j)) \tag{2.3.4}$$

Matching the Template to the Input Image

To check all possible occurrences T must be transformed, rotated and deformed using all possible parameters. To find the best solution there is a need to retrieve the global maximum of the fitness function for the input image. That is, to compare each $S(i, j)$ to all possible regions $R(i, j)$ within the image. A global search is computationally complex [28], especially when the image search space is large. In the literature Genetic Algorithms and dynamic programming approaches [33, 58] have been used for optimization.

Post-processing: Straightening the Template

After a sequence of slices is found, the shape can be straightened by rotating each found slice back. Since each deformed template T has a deformation defined by $\underline{\xi} = (x, y, \alpha_0, \alpha_1, \ldots, \alpha_n)$ each of its slices is rotated back by the slice angle $-\alpha$. In that way the global deformation of the deformed template is reverted to the prototype template T_0.

2.3.2 Application to the Zebrafish Larvae Images

Pre-processing

Because of uneven illumination, global threshold methods applied to a gray-scale version of the zebrafish images will not produce satisfactory results. We, therefore, employ an edge map based method to the input image. Edges define the boundaries between objects and background without strong dependency on flaws in the illumination. There exist several algorithms to create an edge map. After creation of the edge map a threshold must be set to select the strong edges.

Determining the threshold value of an edge map can be a cumbersome task. To set the threshold automatically without prior spatial relationship knowledge of the image, Otsu's method [46] might be applied.

However, we have prior spatial relationship knowledge of our particular data set and exploit this for our own border based method: the objects in the image are always located at some minimal distance d from image border. We try to exploit this. Thus we can assume that a layer of thickness d on the outside of an image contains only background pixels with some incidental noise. Of this layer we retrieve a number of local maximal pixel values. Of all the collected values we take the median value to determine the threshold value for the edge map.

To select the best preprocessing approach we compare the performance of different edge detectors (Sobel, Roberts) [21] in combination with Otsu's [46] method and our

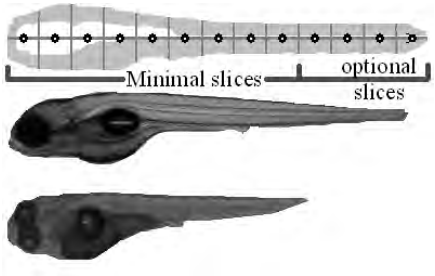

Figure 2.21: Minimal slices and optional slices in a prototype template. The template is compared to an example of a short and a long larvae shape.

border based method on 233 images. We count the number of objects in the segmented image. If the number of objects equals the number of objects in the original image we mark the prediction as correct. Both gradient methods in combination with our border method outperformed thresholding based on Otsu's method. Out of the images incorrectly segmented in 17 cases the zebrafish shapes were touching each other and were thus connected. In all of these cases both edge detectors predicted 1 or 2 fish instead of 3 due to this connection.

After basic segmentation is realized, mathematical morphology operations are applied to get rid of noise and close up unwanted gaps: first the closing operator is applied to interconnect small regions and close holes, then connected components labeling (with filling up holes) is applied to obtain the closed shapes. In order to eliminate remaining noise we use the fact that we know the minimal area covered by a zebrafish. This area size can be automatically retrieved from the template size. Thus we remove all objects smaller then this minimal area. We do not remove objects that are larger than the maximal area, since larger object might be intersecting zebrafish shapes.

Main Process: Deformable Template Matching Prototype Template

Our initial zebrafish template in Figure 2.21 is created from averaging a test set of training shapes [10]. This template is created from averaging a set of 20 zebrafish larvae shapes.

Zebrafish larvae tend to have different lengths. Therefore, the length of the template is not fixed, but can vary between some minimal (min) and maximal (max) number of slices t_{min} and t_{max}. If the number of found slices is smaller than min or larger than max we assume the shape is not found. All the slices t_x with $min < x < max$ are thus optional slices. This is depicted in Figure 2.21. The max and min are set based on the length of the longest and shortest encountered zebrafish.

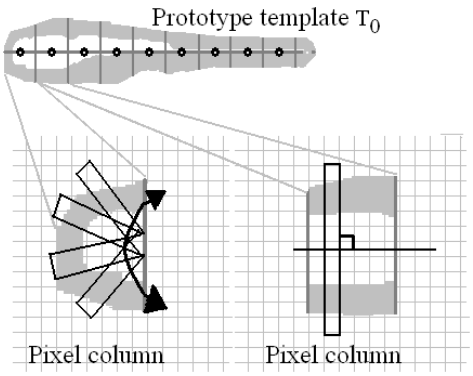

Figure 2.22: Marching direction for the first slice depicted in the image pixel matrix.

Objective Function for the First Slice.

The described objective function is applicable for slices in which the important information is located above and below the slice center. While most of the zebrafish larvae template applies to this condition the very first (head) slice does not. The reason for it is that its shape is close to half circular. This is depicted in Figure 2.22.

To cope with this case, instead of retrieving extreme values above and below the median axis (described in Section 2.3.1) extreme values are retrieved in a circular way as shown in Figure 2.22.

Matching the Template to the Shape

A *top-down* dynamic programming approach is applied with a hash table for intermediate result saving, to obtain a global optimum.

In our test case the larvae shapes are located in an approximately horizontal position in the input image. This given is used to reduce the search space. This is done by assuming that each slice can rotate between -45 to 45 degrees as measured from the image horizontal axis. Additionally a discrete set of deformation angles for each slice is used.

To further reduce the search space a *Multi resolution* algorithm [30] is used. First the solution is located on a low resolution template and a low resolution input image. Then, the solution is used for initialization in a higher resolution.

2.3.3 Experiments and Results

An evaluation of our algorithm is performed on a data set consisting of 233 images. The data set was directly obtained from a running experiment. Out of these images 177 (76%) contained multiple (3) zebrafish larvae, 33 (14%) contained 2 larvae and 23 (10%) 1 larva.

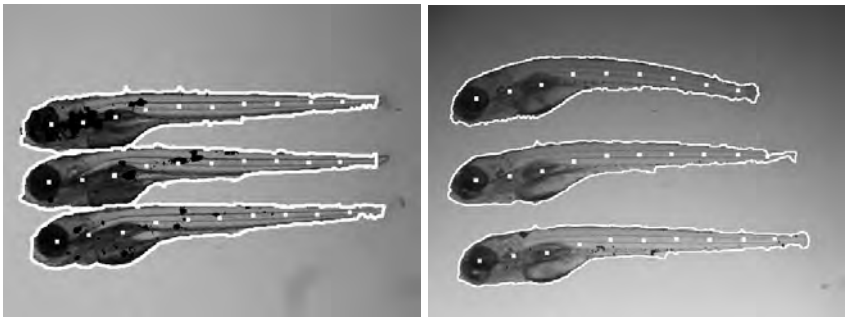

Figure 2.23: Automatic localization of the zebrafish shapes using deformable template matching. The white line defines the shapes as found by the algorithm. White dots represent the found slice centers. In the left image the shapes are touching each other, which makes recognition more difficult. These images were created using the same template (these images were used to determine bacterial infection, black regions, in each larva separately).

A first basic test is to check for how many of the tested images the number of larvae is predicted correctly. The result was that for all images (100%) the number of the larvae shapes (1, 2 or 3 larvae, even if overlapping) within each image was correctly retrieved by our framework.

The automated prediction of the number of shapes in an image looks promising, but only retrieving the number of shapes is not sufficient for a proper analysis of the results. To that end we also test the accuracy of the algorithm, that is, how precise the shapes were retrieved. In Figure 2.23 representative results of retrieved shapes are depicted. We show shapes that are deformed in different ways as well as shapes touching each other. The template in Figure 2.21 was used for the creation of all these images.

We are not aware of other methods that can be used for automatically retrieve predefined shapes from images without an initialization, and therefore we have compared the resulting shapes of each retrieved zebrafish larva shape against ground truth images of the same shapes annotated by experts. In order to have manageable proportions in the evaluation, we reduced our test set to a total of 104 zebrafish shapes (distributed among 35 images, containing up to 3 shapes per image).

Four experts (test-person T1, T2, T3, T4) were asked to delineate the correct outline of the zebrafish larva. Drawing the shapes was realized with a LCD-tablet (*Wacom*) using TDR software [61].

Next, the precision of our approach is compared by applying it to the same input data (algorithm output A). A comparison of human to automatic retrieval was also used in [47] for validation of the results. The accuracy of our shape retrieval algorithm is measured by the equation proposed and explained in [44]:

$$accuracy = 2 \times \left(\frac{N(M_{area} \cap S_{area})}{N(M_{area}) + N(S_{area})} \right) \times 100\% \tag{2.3.5}$$

	A	T1	T2	T3	T4
A	-	96.85	97.19	96.29	96.47
T1	96.85	-	97.61	97.21	96.81
T2	97.19	97.61	-	97.17	97.46
T3	96.29	97.21	97.17	-	96.68
T4	96.47	96.81	97.46	96.68	-

Table 2.2: Accuracy comparison of our algorithm A and the test subjects T1, T2, T3, T4. The matrix is symmetrical, yet we have shown all the values for viewing convenience.

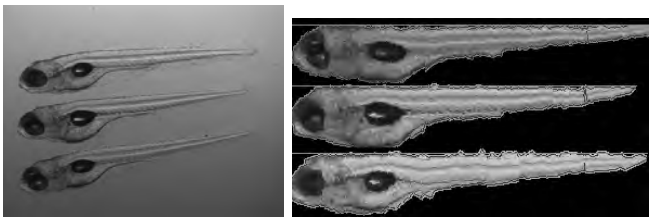

Figure 2.24: Automatic straightening results of zebrafish larvae. Image (left) is the input image. Image (right) is the straightening result.

We have compared the accuracy of our algorithm to T1, T2, T3, T4. The average accuracy was established as 96.71% ($\sigma = 1.27$). Note that this accuracy could not be achieved only by the segmentation step, as 35 (of the 104) shapes used for this test were touching each other and their boundary was derived through the template matching.

Table 2.2 depicts the results of the comparison of the accuracy of our algorithm A with the test persons. In addition the inter-observer variation is analyzed.

As can be seen from the table the accuracy between our algorithm and each test person is as close to the accuracy of the test persons to each other. This indicated that the algorithm retrieves shapes as good as or comparable to manual retrieval.

In the last part of the experiment the objects, i.e., zebrafish, are straightened. This is accomplished using a template with a straight top border in order to align the found slices with their top to a horizontal line. The results are shown in Figure 2.24 and Figure 2.25. To retrieve and straighten a single zebrafish shape from a 2592×1944 image our application needed about 35s of CPU time on an Intel Dual Core 2.66Ghz, 1.00Gb.

2.3.4 Conclusions

In this section we further investigate the possibilities of a deformable template approach. The prototype template proposed in this section is represented as a bitmap and can easily be adapted to the needs of the application while the same algorithm is used. Results can then be compared by shape normalization. The algorithm we propose does not rely on initial localization of the shape and therefore does not require any manual intervention

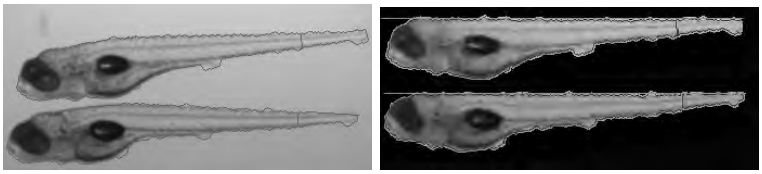

Figure 2.25: Automated straightening results of zebrafish larvae. Image (left) shows the found shapes projected on the input image. Image (right) shows the automatic straightening result.

or analysis. The framework was applied in an experimental set up for HT screening with a readout in images. In the application to zebrafish screening an average accuracy of about 96 percent has been achieved. The framework can be easily adapted to work with other shapes, e.g., in the life sciences or in other fields that require accurate and robust shape retrieval. Further analysis of the object straightening precision is part of the future work.