**TELKOMNIKA Indonesian Journal of Electrical Engineering**
Vol. 14, No. 1, April 2015, pp. 173 ~ 184
DOI: 10.11591/telkomnika.v14i1.7467

# An Efficient Converging Snake Algorithm for Locating Object Boundaries

**Atiqur Rahman*[1], Rashed Mustafa[2]**
Department of Computer Science & Engineering, University of Chittagong, Chittagong, Bangladesh
*Corresponding author, email: atiqcse09@cu.ac.bd[1], bulbul.cse.cu@gmail.com[2]

***Abstract***

*Active contour are now established as a technique for extracting salient contours from an image. A snake is an active contour for representing object contour. Traditional snake algorithms are often used to represent the contour of a single object. A different contour search algorithm is presented in this paper that provides an efficient convergence to the object contours than both the kass et al and greedy snake algorithm (GSA).Our proposed algorithm provides a straightforward approach for snake contour rapid splitting and connection, which usually cannot be gracefully handle by traditional snakes. This algorithm compares with other two conventional approaches is faster according to needed execution time. This paper tells us which one is better by comparing each other. The experimental results of various test sequence images with a single object shown good performance, which proved that the proposed algorithm is faster among those.*

***Keywords:*** *active contour, snake, parametric curve, finite differences, greedy algorithm*

## 1. Introduction

Snakes are mainly used to dynamically locate the contour of an object. In order to answer the question what is an active contour and how does it work and what separates the different active contour methods from each other, a brief review of the main research papers dealing with active contours will be given, followed by a detailed analysis of the theory behind the Kass et al [20], snake and the greedy snake algorithm [14]. The reason for choosing these two algorithms for a closer comparison is that the Kass et al paper was the first paper to suggest energy minimizing snakes for image segmentation. The greedy snake algorithm on the other hand is interesting to examine since it deals with discrete values when minimizing the snake's energy. Most of the remaining snake algorithms are also more or less variations of these two algorithms. To answer the question how would an active contour algorithm be implemented in MATLAB, both the Kass et al. snake and the greedy snake will be implemented in MATLAB. By running a number of experiments, with the two implemented snakes, both on synthetic and real images the questions under which conditions does an active contour perform satisfactory, under which conditions does an active contour perform unsatisfactory will be sought answered. Finally the strengths and weaknesses of each of the three snake algorithms will be discussed. In the process of writing this report and implementing the algorithms I also hope to acquire a deeper understanding of active contours, since this is an area of much interest to our. It should also be noted that to limit the scope of this thesis we shall only deal with parametric active contours and not Geodesic active contours.

The rest of this paper will be organized according to the following structure: section 2 literature review; implementation framework will be illustrated in section 3; section 4 elucidates results; section 5 discussion and section 6 concludes this paper.

## 2. Literature Review

This section will start with a general introduction of what an active contour is used for and how it works. Hereafter a brief literature review is presented which highlights the main differences for some of the most widely known active contour methods.

*Received January 7, 2015; Revised March 13, 2015; Accepted March 28, 2015*

Active contours are used in the domain of image processing to locate the contour of an object [20]. Trying to locate an object contour purely by running a low level image processing task such as Canny edge detection is not particularly successful [17]. Often the edge is not continuous, i.e. there might be holes along the edge, and spurious edges can be present because of noise. Active contours try to improve on this by imposing desirable properties such as continuity and smoothness to the contour of the object. This means that the active contour approach adds a certain degree of prior knowledge for dealing with the problem of finding the object contour.

An active contour is modeled as parametric curve; this curve aims to minimize its internal energy by moving into a local minimum. The position of the snake is given by the parametric curve, in practice the curve is often closed which means that v(0) = v(1). Furthermore the curve is assumed to be parameterized by arc length [15].

A closed parametric snake curve is illustrated in Figure 1. Each point along the curve is under the influence of both internal and external forces, and the snake continuously tries to position itself so that the combined energy of these forces is minimized.

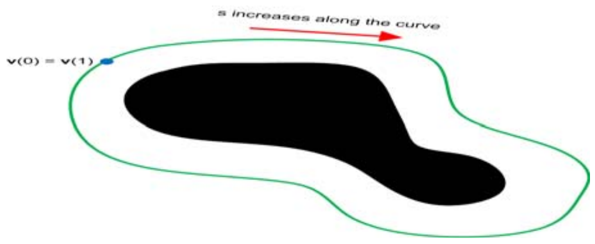

Figure 1. Illustration of a parametric snake curve v(s). The blue dot marks the starting point and end point of the snake curve. Ideally the snake will have minimized its energy when it has positioned itself on the contour of the object

### 2.1. Analysis of the Kass et al. & Greedy snake [20]

In the previous section a short introduction to what an active contour is and how it works was presented, followed up by a review of some of the best know parametric active contour models. In this chapter we will continue with a more thorough analysis of the Kass et al. and the greedy snake algorithm. Thereafter possible extensions and improvements to the two snake algorithms will be examined. However, let us first take a look at the parametric curve and examine why it is used as the basis for the snake model.

The most basic way of representing a curve in two dimensions is the explicit form:

$$y=f(x) \tag{1}$$

Nearly all simple curves can be represented using the explicit form, but for more complex curves with two or more y-values for a single corresponding x-value the explicit form fails [20]. Moreover the explicit form has problems with modeling curves parallel to the y-axis, since the slope for such a curve tends towards infinity [20]. These shortcomings excludes the use of the explicit form for curves such as circles, ellipses and other conics. The two dimensional implicit curve equation can be used to represent any curve, thus avoiding the limitations of the explicit curve. Unfortunately the inherent form of the implicit curve equation does not allow us to compute points on the curve directly, often requiring the solution of a non-linear equation for each point. Furthermore both the explicit and implicit form requires that the underlying function is known, and in practice when dealing with complex deforming curves such as snakes this is not the case. This leads us to consider the parametric form of a curve which, in vector form, is specified as:

$$f(x,y)=0 \tag{2}$$

$$\mathbf{v}(s) = \begin{bmatrix} x(s) \\ y(s) \end{bmatrix}$$

The parametric curve only has one independent parameter s which is varied over a certain interval, usually [0; 1]. By using the parametric representation we avoid the problems that both the explicit and implicit forms have. For instance we can have multiple y-values for a single x-value, this is easy to see in the parametric form of a unit circle with center at origin.

$$\mathrm{v}(s) = \begin{bmatrix} x(s) \\ y(s) \end{bmatrix} = \begin{bmatrix} \cos s \\ \sin s \end{bmatrix} \quad \text{with} \quad s \in [0, 2\pi].$$

Finally using the parametric representation also enables the curve to be independent of the particular coordinate system used [18].

## 3. Implementation Framework

This section will contain a brief introduction to how the implemented snakes are run, together with an explanation of some of the implementation details.

The two snake algorithms have been developed and tested using MATLAB R2010a running on windows 7 version Home premium. The program is run from the MATLAB command line and takes 3 command line arguments. All 3 input arguments are strings. The first argument can either be Kass or greedy specifying whether to run the Kass et al. or greedy snake algorithm. The next argument can be either user or circle, this decides whether the snake should be manually input by the user or input as a circle. The last argument specifies whether scale space continuation should be on or off. So for instance if the user wishes to run the greedy snake, input the snake control points manually and have scale space continuation turned on he should input the command line of the MATLAB console below:

```
main('greedy', 'user', 'off')
```

The file main.m runs the snake program, it also contains nearly all of the parameters and thresholds used. So if the input image should be changed or the parameter β adjusted it can be done by editing this file. If the user has chosen to input the snake manually the image will be shown and the user then clicks in the position of the image where a snake control point should be inserted. Once finished the user should click somewhere outside the image to start the snake algorithm.

The snake program is dependent on the Image Processing Toolbox for MATLAB being installed, as functions from the toolbox are used for doing convolution. For approximating the directional gradients $G_x$and $G_y$of the image function I(x; y), we use convolution with Sobel filters.

$$G_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} * \mathrm{I}(x,y) \quad , \quad G_y = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} * \mathrm{I}(x,y).$$

The gradient magnitude used in the image energy terms is then computed as:

$$\|\nabla I\ (x,y)\| = \sqrt{G_x^2 + G_y^2} \qquad (3)$$

In the Kass et al. snake implementation the value h used in the finite differences is set to h = 1. This speeds up the computation and in practice the snake still evolves satisfactory.

Below in Figure 2 a data-flow diagram is show. This diagram gives an overview of all the MATLAB files and how the functions invoke each other.

The main function in main.m contains most of the parameters and thresholds used, furthermore it also handles drawing the snake evolution in the image. The Kass Snake function calculates how the snake moves, given the initial position and the image energy, based on the Kass et al. snake model. In the Greedy Snake function the evolution os the snake is calculated on the basis of the greedy snake algorithm. The getAvgDist function returns the average distance between all the snake control points in the snake. The getModulo function is used extensively by the Greedy Snake function. This is because we want to make sure that when a loop is applied to iterate through all the snake control points the first and last point will be the same. Finally the snake Resample function tries to resample the snake control points so that they are equally spaced along the snake curve.

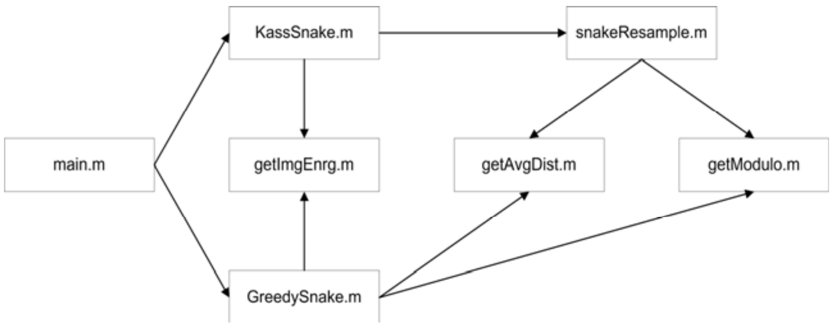

Figure 2. Illustration of the data-flow between the various functions in the implementation

Determination of intersection segments equation is:

$$value=(b_1\cdot b_2 \leq 0 \ \&\& \ b_3\cdot b_4<0) \| (b_1\cdot b_2<0 \ \&\& \ b_1\cdot b_2\leq 0) \quad (4)$$

Splitting and connecting contours equation is:

$$value=(\bar{A}_i\cdot \bar{A}_k) < (\bar{A}_i\cdot \bar{A}_{k-1}) \quad (5)$$

### 3.1. Our Proposed Algorithm of the Boundary Detection of a Objects

The algorithm of the boundary detection of a object is shown as follows:

*Step 1*: Convergence process: calculate the energy functions and minimize the energy terms of snake points. If the iteration reaches the final step, stop. Otherwise, go to step 2.
*Step 2*: The process of determining intersection: if the snake point intersects segment $s_i$ estimated by the equation (3.2), then go to step 3. Otherwise, go to step 1.
*Step 3*: The splitting and connecting process: split the contour by removing the unnecessary Point $v_k$. Snake points, which belong to the same side, are connected by equation(3.3) and then, go to step 4.
*Step 4*: Reorganizing the sequence of the snake point process: a new sequence is formed for each contour. Go to step 1.

## 4. Results

Both the greedy snake, the Kass et al. snake & our approach, that were analyzed and described in chapter 3, have been implemented in MATLAB. In this chapter we will test these three snakes on different types of images. Some of the images will be synthetic while others will be real images captured by a digital camera. All the synthetic images have been made using Photoshop 10.0.

### 4.1. Test number 1; Boundary concavity

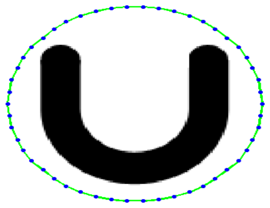

Figure 3. Illustrating an object with a boundary concavity

The initial snake forms a circle around the object and consists of 50 snake points.

The image we will use in the first test is a synthetic image designed to show one of the weaknesses that the Kasset al. snake, greedy snake and many other snakes have. The problem appears when trying to locate the contour of an object which has a boundary concavity. In Figure 3 we see the initial snake and the object of which we wish to find the contour. The object has a large concavity and as we will see the either the Kasset al.snake or the greedy snake is able to move completely into this concavity. The initial snake in Figure 3 has 50 control points and both the two snakes will start evolving from this position.

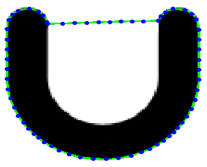

Figure 4. Finals state of the Kasset al.Esnake. Iterations 8000, α= 0:035, β= 0:0005,δ = 3, scale space continuation off and resampling on

In Figure 4 we see the result for the Kasset al. snake. The snake parameters were α= 0:035, β= 0:0005, α= 3 and scale space continuation was off while resampling was on. The snake ran 8000 iterations which is the maximum number of iterations. The reason that the snake did not stop sooner is that new points keep being inserted in the horizontal stretch above the cavity. These new points slowly move out to the sides as they are inserted and the snake never converges according to our stopping criterion, see section 3.3.2. From Figure 3 we clearly see that the snake is not able to move into the cavity. This behavior is caused by a combination of the snakes internal energies and the image energy. The elasticity energy tries to keep the snake from stretching and if the snake is to move down into the cavity the elasticity energy would have to be increased. However if the image energy was somehow pulling the snake downwards into the cavity the elasticity energy could be overcome, but that is not the case. The image energy of 4.1 is only pulling the snake out to the sides of the cavity and not downwards in any way. In Figure 5.3 the final state of the greedy snake is shown. For the greedy snake the final state was reached after 55 iterations. The parameters were α = 1, β= 1, ϒ= 1:2, δ= 3, neighborhood size was 3 ×3and scale space continuation was off. The red snake control points indicate points for which the βvalue has been relaxed by the algorithm, so a corner could develop. Figure 5 clearly shows that also the greedy snake also has problems with moving into the cavity. Actually it does slightly worse than the Kass et. al. snake, since the Kasset al. snake protrudes somewhat deeper into the cavity.

If we wish to correctly segment an object with boundary concavities using a snake we must make sure that the initial snake is placed inside the concavity. This way the snake will be caught by the image energy and stick to the contour.

The gradient vector flow snake which was briefly described in the literature review section should, by itself, be able to move into boundary concavities, but this snake has not been implemented and will therefore not be tested.

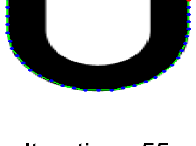

Figure 5. Final state of the greedy snake. Iterations 55,α= 1, β = 1, ϒ= 1:2, δ = 3, neighborhood size 3 × 3 and scale space continuation off

### 4.2. Test number 2; Noisy Image

This test is designed to examine the performance of the two snakes on noisy images, both with and without using scale space continuation. The image seen in Figure 6 shows a black square on a white background. A high degree of Gaussian noise has been added to the image. The noise was added in MATLAB with the command noisyImg = imnoise(img,'gaussian',0.5,2); where 0.5 is the mean and 2 is the variance of the Gaussian. As in the last test the initial snake curve consists of 50 control points placed in a circle around the object we with to find the contour of.

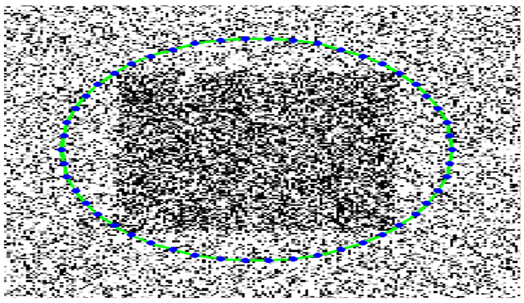

Figure 6. Image of a black square on a white background with a high degree of noise. Initial snake consists of 50 snake control points

In the first two test runs we will run the snakes with scale space continuation turned off. When scale space continuation is turned off we only blur the image once with a Gaussian of $^{\delta}$= 3 and then let the snake run its course on the blurred image. Figure 7 shows the final state of the Kass et. al. snake. We used the parameters α= 0:05, β= 0:0005, α= 3 and had resampling on. Unfortunately the snake was not able to converge according to our stopping criterion and ran all 8000 iterations. Even though the snake runs 8000 iterations it actually moves very little, since it quickly gets stuck on the artifacts that the noise creates in the image energy.

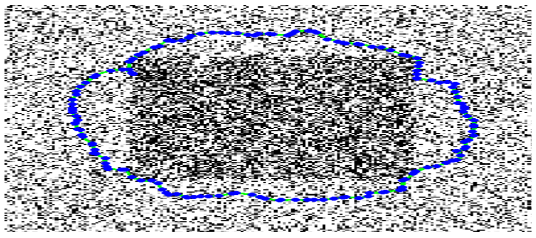

Figure 7. Final state of the Kasset al. snake. Iterations 8000, α = 0:05, β = 0:0005,ϒ= 3, scale space continuation off and resampling on

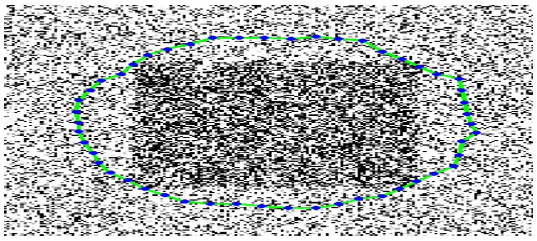

Figure 8. Final state of the greedy snake. Iterations 5, α = 1, β = 1, ϒ= 1:2, δ = 3, neighborhood size 5$^{x}$5 and scale space continuation off

Figure 8 shows the final state of the greedy snake on the same image. For the greedy snake the parameters were set at α = 1, β= 1, ϒ= 1:2, δ= 3 and a neighborhood of size 5$^{x}$5. Again we observe that the snake quickly gets stuck because of the noise. Actually the greedy snake stops after only 5 iterations in this example. From these results it is clear that the amount of noise added presents a significant challenge to both of the snake algorithms.

For the next two test runs we will turn scale space continuation on. This should help in making the snakes more robust in the presence of noise. The implemented scale space continuation uses 4 different scales. Starting at a rough scale with δ = 15 and then reducing δ by 4 until we reach a scale of δ = 3. At each individual scale the snake is allowed to run until it converges or until 1/4 of the total iterations have run.

In Figure 9 the final results of the Kasset al. snake, with scale space continuation on, is shown. It took 4300 iterations for the snake to converge to this result, and the parameters were α= 0:05, β= 0:0005, scale space continuation on and resampling on. It is quite evident from looking at Figure 9 that scale space continuation helps make the Kasset al. snake more robust. We can now actually see that the snake has found the contour of the square.

The corresponding result for the greedy snake is shown in Figure 10. The parameters were α = 1, β = 1, ϒ= 1:2, δ = 3, neighborhood size 5x5 andthe number of iterations was 76. Again we see that scale space continuation helps significantly, which was to be expected also for the greedy snake.

The two snakes both find the contour quite well considering the amount of noise in the image. However it seems the Kasset al. snake performs slightly better than the greedy snake. The greedy snake does not find the lower left corner of the square in Figure 10.

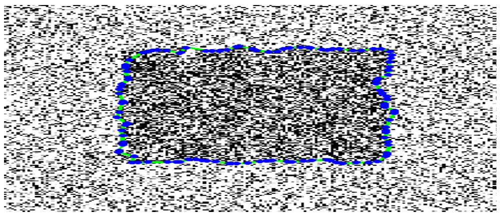
Figure 9. Final state of the Kasset al. snake with scale space continuation on. Iterations 4300, α= 0:05, β= 0:0005 and resampling on

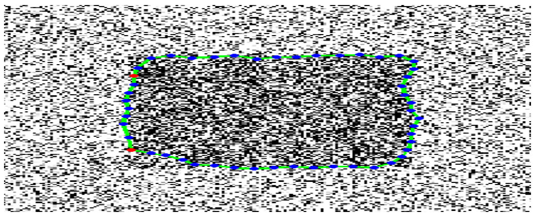
Figure 10. Final state of the greedy snake with scale space continuation on. Iterations 76, α = 1, β = 1, ϒ= 1:2 and neighborhood size 5 x5

### 4.3. Test number 3; Image of a Leaf

Now that we have tested the snakes on two synthetic images we will try to apply them to a real image. The image is cut out of a larger image showing a number of leafs lying on a table.

The initial snake has been initialized manually for this test. This was done to illustrate how a manually initialized snake should be placed, but also because placing the snake in a circle around the leaf gave bad results. In Figure 11 the manually placed initial snake can be seen. There is 78 snake control points in the initial snake.

In Figure 12 we see the final result for the Kasset al. snake. The parameters were α = 0:035, β = 0:0005, resampling on and scale spacecontinuation on. The snake converged after 6161 iterations. We see that the snake has found the general shape of the leaf contour quite well. There is however some problems with the upper right corner and lower left corner of the leaf. This can be attributed to the fact the the image is quite blurry in this part so the exact transition between the table and the leaf is hard to make out.

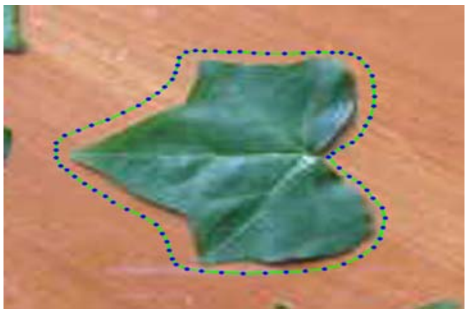
Figure 11. Image showing a leaf laying on a table. Initial snake consists of 78 snake control points

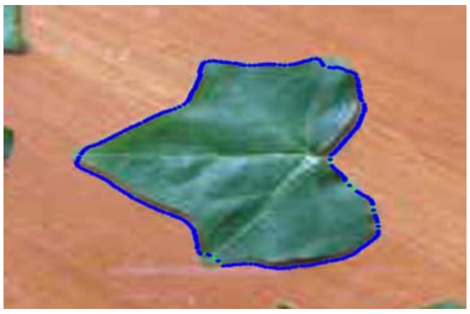
Figure 12. Final state of the Kasset al. snake. Iterations 6161, α = 0:035, β = 0:0005, resampling on and scale space continuation on

The final position of the greedy snake is shown in Figure 13. The parameters were α = 1, β = 1, ϒ= 1:2, neighborhood size 5 ×5 and scale space continuation on. While the number of iterations runs were 128.

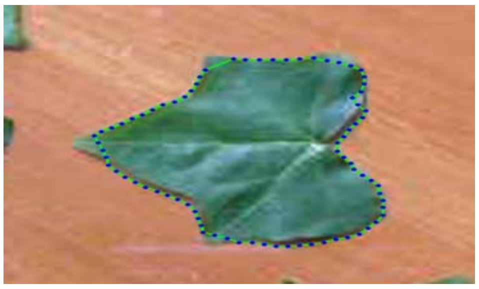
Figure 13. Final state of the greedy snake. Iterations 128, α = 1, β= 1, ϒ= 1:2, neighborhood size 5 ×5 and scale space continuation on

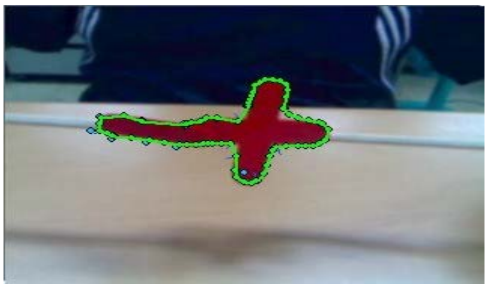
Figure 14. Final state of our approach

The greedy snake also seems to capture the general contour of the leaf. However there is a few more errors made, compared to the Kass et. al. snake. As with the Kass et. al. snake the greedy snake also has problems with the upper right corner and the lower left corner of the leaf. Furthermore the greedy snake also has some problems with the upper left corner and the tip of the leaf [13].

## 5. Discussion

### 5.1. Comparing Computational Speed

We have observed how well both of the three snakes find different object contours. Now we will briefly examine the computational speed. In the table the time it takes each snake to run 100 iterations is shown.

Table 1. Comparing the running time of the Kass et al. & greedy snake to the running time of Our approach

| Snake | Running time for 100 iterations |
|---|---|
| Kass | 0.893994 seconds |
| Greedy | 4.111357 seconds |
| Our approach | 0.5123 seconds |

The test was run on a HP G62 Notebook PC 2.00 GHz. with 4 GB ram. The shark tooth image was used with the following parameters. Kass et al.: α =0:05, β= 0:0005, re sampling on and scale space continuation on. Greedy: α = 1:2, β = 1, ϒ= 1:2, neighborhood size 5×5 and scale space continuation on. The Kass et al. snake is seen to be significantly faster than the greedy snake when it comes to running a hundred iterations. In practice however the Kass et al. snake also has to complete a far greater number of iterations to converge. Thus in reality the greedy snake actually converges faster than the Kass et al. snake. My approach snake is seen to be significantly faster than the Kass et al. snake& the greedy snake when it comes to running a hundred iterations.

Our experimental results show that snakes can be used to segment a variety of different shapes. It was also established that scale space continuation greatly reduced the influence of noise in the images.

Comparing the results from both the Kass et al. snake and the greedy snake with my approach leads us to the conclusion that my approach snake gives slightly better results overall.

One of the reasons for the better results is of course that our re sampling method increases the number of control points which enables the snake to find the contour of finer details. However if we try to increase the number of points in the greedy snake, the snake point neighborhoods can start to overlap. This leads to undesired effects such as the snake curve overlapping in some places, or even that the distance between snake points begins to increase. Therefore we recommend to use around 50 snake control points in the greedy snake, this number can of course be increased if high resolution images are used.

We also recommend that if the object to be segmented contains boundary concavities and noise that the initial snake is manually initialized. The closer the initial snake is to the desired contour the greater the probability is of the snake also converging to the desired contour. Having to initialize the snake close to the desired contour can of course present some problems if a completely automatic system for contour detection is sought after. The problem of needing to find a good initial position to increase the chance of finding the desired contour is actually one of the inherent problems with snakes. This problem has lead to much research into how to best initialize the snake curve.

Another problem with snakes is the fact that the parameter of the snake often has to be adjusted for each new kind of image in order to give the best results. Finding suitable values for the parameters relies on manual tuning based on trial and error. This process can often be time consuming. In the above test runs the parameters were adjusts for each snake in each image to give the best possible result.

### 5.2. Extensions and Improvements

In this section extensions to the original algorithms will be presented along with the improvements we have made in order to better the algorithms.

#### 5.2.1. Stopping Criterion for the Kass et al. Snake

In the original paper by Kass et al. no indication was given as to when the snake evolution should be stopped. In the greedy algorithm we would stop the iterations when either the maximum number of iterations had been exceeded or when the number of snake points moved in the last iteration was below a threshold. Setting an upper limit on the number of iterations run, can of course also be used for the Kass et. al. snake. However we cannot use the number of point moved as a stopping criterion for the Kass et. al. snake. This is because most of the points in the snake keep moving even when the snake has reached its minimum, the movement at the minimum is however very small. Knowing that the movement of the snake points at the minimum will be quite small, we suggest to stop the snake algorithm when the following term drops below a given threshold

$$\frac{\|\mathbf{v}(s)^t - \mathbf{v}(s)^{t-1}\|}{n}.$$

Where the vector v(s)t contains the indices to the snake points at time step t and v(s)t–1 contains the snake points at time step t –1. We divide by n which is the total number of control points in the snake. Usually a greater number of control points will lead to the term $\| v(s)^t – v(s)^{t-1} \|$ taking on larger values, which is why we divide by n.

This improvement of the Kasset al. snake has been incorporated into our implementation.

#### 5.2.2. Active Resampling of the Kass et al. Snake Curve

In this section we therefore present an improvement to the original Kass et al. snake algorithm which makes sure that the distances between all the snake control points remain more or less equal.

Once the snake has started evolving there is no guarantee that the snake control points are equally spaced, however the snake curve can be dynamically resampled while it evolves. The resampling step that we have added to the original Kass et al. snake first computes the average distance between all the snake control points. Then we iterates through all the snake control points while removing points in parts of the snake where the points are close together compared to the average distance. At the same time the resampling step also inserts new points in parts of the snake where points are far apart compared to the average distance. When a new point is inserted, it is inserted in the middle of the line connecting the two points that are far apart. The resampling is not performed after each iteration of the snake, in the actual implementation we have, in order to reduce computation, set the resampling to be performed every time the snake has iterated 15 times.

Let us take a look at an example and see how adding the resampling changes the result. In Figure 15 the initial snake is show, it consist of 100 snake control points laid out in a circle around the object we wish to segment. Running the Kass et al. snake, with snake resampling turned on, results in the snake converging after 2163 iterations to the state that is seen in Figure 16. We notice that when the snake has converged it hasincreased the number of control points to 195. This is to be expected, since the resampling function is more inclined to add points to the snake rather than remove them. This inclination can however be changed by adjustinga threshold value in the resample function. Adding additional point to the snake actually gives a better fit to the contour, since more points mean that finer details along the contour can be modeled by the curve. Another thing to notice is that the snake in the right half of Figure 16 is not able to move into the concave part of the object. This is a trait that both the greedy snake and the Kasset al. snake exhibits, and we will discuss it in more detail in the experimental results chapter.

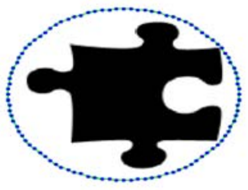
Figure 15. The initial state of the snake

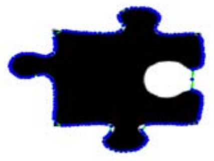
Figure 16. The final state of the snake when using resampling, after 2163 iterations

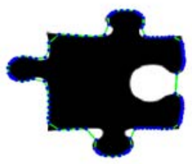
Figure 17. The final state of the snake without resampling. The final state occurred after 2346 iterations

To compare the result we got when having resampling turned on, we ran the Kasset al. snake algorithm in its original form i.e. with resampling turned on the same test image. We set the initial snake to consist of 195 control points to make sure that the end result is directly comparable with the result obtained from having resampling turned on. The converged snake is shown in Figure 17, this time convergence took 2346 iterations.

When comparing the two results we quickly see that not using our resampling method gives worse results for the final segmentation. This is most noticeable in the lower left part of the object in Figure 17 where the snake curve does not follow the contour very well. It is also noticeable that the snake control points are mostly concentrated on the right side of the object in Figure 17 while being much further apart on the left side. We can therefore conclude that using our suggested snake resampling technique not only helps keep the points more evenly spaced, which in turn makes the curvature term more precise. But also directly improves the segmentation of the object in question.

### 5.2.3. Randomizing Snake Points for the Greedy Algorithm

The last improvement that has been implemented is concerned with the greedy snake algorithm. It was noted during initial test runs, when implementing the greedy snake algorithm, that the snake sometimes would have a tendency to rotate. The rotation is particularly noticeable when the image energy is more or less constant throughout the snake. This rotation is due to the fact that the for-loop, which runs through all the snake points computing their new position, does so consecutively. So if one of the control points in the snake is moved closer to another control point, and the image energy and curvature energy is more or less constant thought the snake, it will tend to push the next control point. This is because the elasticity energy for the greedy snake always tries to keep the points evenly spaced. Once one of the control points has pushed another it can set in motion a chain reaction resulting in the whole snake rotating [1].

To avoid this behavior we have made a small change to the greedy snake algorithm. Instead of using a for-loop to consecutively go through each snake control point, we choose the snake control points by random. This approach significantly reduces the tendency for the curve to rotate, as it will be harder for the force of one point pushing the next point to propagate along the curve. It should be noted that the results obtained, when running the same initial snake on the same image, can vary slightly because of the randomization factor [2].

## 6. Conclusion

The main goal of this paper was to compare three different methods within the active contour framework. The active contour methods that were chosen for comparison was the Kass et al. snake, the greedy snake & our approach. The goal of comparing these three methods has been reached. The results of experiments prove a robust, effective, efficient and accurate performance of the proposed method. In the development of the boundary detection of a object, this proposed algorithm can work in a wide range of applications where the classical active contour have failed. We have throughout this paper only been considering snakes that form closed curves. One possible extension could be to extend the implemented program to also deal with open curve snakes. With an open curve snake it would be possible to find contours in the image that does not form closed curves. For instance finding the contour of a line running across the image. Extending the Kass et al. snake into handling open curve contours is done by removing the cyclic boundary conditions. This results in a slightly different form of the coefficient matrix A. More details for developing an open curve snake is found in. The greedy snake could also be extended to work with open curves by removing the use of modulo arithmetic when looping through all the snake control points.

## References

[1] Delgado-Gonzalo, Ricard, et al. Snakes on a Plane: A perfect snap for bioimage analysis. *Signal Processing Magazine, IEEE.* 2015; 32(1): 41-48.

[2] Bindu VR, KN Ramachandran Nair. Boolean Operations on Free Form Shapes in a Level Set Framework. *Computational Intelligence in Data Mining-Volume 3*. Springer India. 2015: 453-467.

[3] Sakalli, Mustafa, Kin-Man Lam, Hong Yan. A faster converging snake algorithm to locate object boundaries. *Image Processing, IEEE Transactions.* 2006; 15(5): 1182-1191.

[4] Bresson, Xavier, et al. Fast global minimization of the active contour/snake model. *Journal of Mathematical Imaging and vision.* 2007; 28(2): 151-167.

[5] Awadallah, Mahmoud, Lynn Abbott, Sherin Ghannam. *Segmentation of sparse noisy point clouds using active contour models*. Image Processing (ICIP), 2014 IEEE International Conference. 2014.

[6] Fraz, Muhammad Moazam, Sarah A Barman. Computer Vision Algorithms Applied to Retinal Vessel Segmentation and Quantification of Vessel Caliber. *Image Analysis and Modeling in Ophthalmology* 2014: 49.

[7] Amine, Khaldi, Merouani Hayet Farida. Deformable models approach for range image segmentation. *International Journal of Imaging and Robotics.* 2014; 12(1): 39-48.

[8] Delgado-Gonzalo, Ricard, et al. Snakes on a Plane: A perfect snap for bioimage analysis. *Signal Processing Magazine, IEEE.* 2015; 32(1): 41-48.

[9] Duggan, Nóirín, et al. A simple boundary reinforcement technique for segmentation without prior. *Pattern Recognition Letters.* 2014; 46: 27-35.

[10] Farouj Y, et al. *A variational Shearlet-based model for aortic stent detection*. Signal Processing (ICSP), 2014 12th International Conference, IEEE. 2014.

[11] Mishra, Akshaya K, Paul W Fieguth, David A Clausi. Decoupled active contour (DAC) for boundary detection. *Pattern Analysis and Machine Intelligence, IEEE Transactions.* 2011; 33(2): 310-324.

[12] Awadallah, Mahmoud, Lynn Abbott, Sherin Ghannam. *Segmentation of sparse noisy point clouds using active contour models*. Image Processing (ICIP), 2014 IEEE International Conference. 2014.

[13] Fraz, Muhammad Moazam, et al. Blood vessel segmentation methodologies in retinal images–a survey. *Computer methods and programs in biomedicine.* 2012; 108(1): 407-433.

[14] CM Bishop. Pattern Recognition and Machine Learning. First Edition. Springer. 2006.

[15] A Blake, M Isard. Active Contours. First Edition. Springer. 2000.

[16] V Caselles, R Kimmel, G Sapiro. Geodesic active contours. *International Journal of Computer Vision.* 1997; 22(1): 61-79.

[17] LD Cohen. On active contour models and balloons. *Computer Vision, Graphics, and Image Processing. Image Under-standing.* 1991; 53(2): 211-218.
[18] LD Cohen, I Cohen. Finite-element methods for active contour models and balloons for 2-d and 3-d images. *PAMI.* 1993; 15(11): 1131-1147.
[19] MT Heat. Scienti_c Computing, An Introductory Survey. Second edition. McGraw-Hill. 2002.
[20] M Kass, A Witkin, D Terzopoulos. Snakes: Active contour models. *International Journal of Computer Vision.* 1988; 1(4): 321-331.
[21] MS Nixon, AS Aguado. Feature Extraction and Image Processing. First Edition. Newnes. 2002.
[22] H Sagan. Introduction to the Calculus of Variations. First Edition. McGraw-Hill. 1969.
[23] D Salomon. Curves and Surfaces for Computer Graphics. First Edition. Springer. 2006.
[24] JB Waite, WJ Welsh. Head boundary location using snakes. *Br. Telecom Journal.* 1990; 8(3): 127-136.
[25] DJ Williams, M Shah. A fast algorithm for active contours and curvature estimation. *CVGIP: Image Underst*. 1992; 55(1): 14-26.
[26] C Xu, JL Prince. Snakes, shapes, and gradient vector flow. IEEE *Transactions on Image Processing.* 1998; 7(3): 359-369.