N-Dimensional Path Optimization: The Implementation of a Novel Algorithm in ITK

John Galeotti and George Stetten

Carnegie Mellon University
galeotti+miccai@cs.cmu.edu

Abstract. Using the path framework we previously added to ITK, we implemented a novel algorithm for n-dimensional path optimization, which we call the ND Swath (NDS). NDS uses dynamic programming to globally optimize the placement of a path within an image, subject to several constraints and a user-supplied merit function. The NDS algorithm is presented in this paper along with a description of how it was implemented using ITK.

Introduction

A significant number of segmentation algorithms utilize active contours or other path-type data objects [1-6]. Here we introduce a new algorithm that uses dynamic programming [4-6] to globally optimize a chain-code path with respect to an initial open or closed chain-code path and a local merit function defined over a ND path placed in a ND image. The merit function is arbitrary and is supplied by the user. The global optimization is also subject to several constraints that cause the initial path to act as a sort of prior. We have named the algorithm the ND Swath (NDS). ITK provided the ND framework necessary to implement NDS, which is based on a similar, unpublished 2D algorithm by Stetten.

The original intent was to implement the 2D algorithm in ITK, but the ND nature of ITK encouraged Galeotti to think beyond two dimensions and in the process he was able to remove some of the original algorithms' restrictions, particularly its restrictions on the type of chain-code it could process. Furthermore, the modularity of ITK encouraged the separation of the merit function from the optimization process, producing a generic ND path optimizer suitable for a wide range of tasks, some of which the authors intend to pursue in future research. It is hoped that the generalized NDS will prove useful to other researchers as well.

Background

The combination of dynamic programming and a restricted search-space make global optimization possible in practical systems, provided that reasonably good initializations are provided. Such initializations can be obtained from human interaction, the use of scale-space, or feature-identification algorithms. Depending on the merit

function used, a path could be optimized, for example, to segment a 2D object, trace a 3D surface (in a generally-specified direction or within a 2D subspace), or find the axis (central, skeletal, major, minor, etc.) of an object. Because of their continuous sequential structure, path-based segmentations are able to produce a well-structured continuous boundary estimate, even when the data is noisy. Noise also is less detrimental if a segmentation is globally optimized, because the boundary estimate will not get trapped in a local extrema produced by the noise. Combined with their efficiency at tracking small boundary changes over time, globally optimized paths can provide useful object-segmentation in noisy ultrasound video data [4-6], potentially enabling real-time shape and motion measurements of diverse anatomic structures.

Notation

We will be using and extending the notation presented in [7]. Chain-codes represent a path as a sequence of offsets between adjoining locations on a rectangular lattice. Whether these locations represent voxel-centers or voxel-vertices is unspecified. Both of these interpretations are equally valid, so long as one or the other interpretation is used consistently for the creation, processing, utilization, and visualization of a given chain-code. Chain-codes can, therefore, be used both to segment between voxels and to trace through the centers of voxels. A chain-code with n offset-steps can be denoted as a sequence

$$\mathbf{P} = \left(\mathbf{u}_1, \mathbf{u}_2, \mathbf{u}_3, \ldots \mathbf{u}_n\right) \tag{1}$$

whose elements are the individual step vectors. An arbitrary step $\mathbf{u}_i$ is a vector from a given index to one of its neighbors.

$$\mathbf{u}_i = \begin{pmatrix} \mathbf{u}_{i_1} \\ \vdots \\ \mathbf{u}_{i_N} \end{pmatrix} = \begin{pmatrix} \Delta x_1 \\ \vdots \\ \Delta x_N \end{pmatrix}, \quad \Delta x_i \in \{-1,0,1\} \tag{2}$$

For a 2D chain-code,

$$\mathbf{u}_i \in \left\{ \begin{pmatrix} 0 \\ 1 \end{pmatrix}, \begin{pmatrix} 1 \\ 1 \end{pmatrix}, \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \begin{pmatrix} 1 \\ -1 \end{pmatrix}, \begin{pmatrix} 0 \\ -1 \end{pmatrix}, \begin{pmatrix} -1 \\ -1 \end{pmatrix}, \begin{pmatrix} -1 \\ 0 \end{pmatrix}, \begin{pmatrix} -1 \\ 1 \end{pmatrix} \right\} \tag{3}$$

as illustrated in Figure 1.

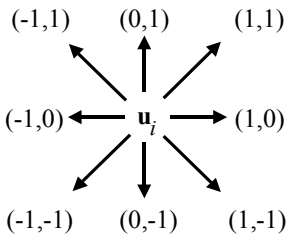

Fig. 1. Step $\mathbf{u}_i$ in a 2D chain-code.

The individual offset vectors (or steps) of a specific chain-code $\mathbf{P}$ may be represented by adding an index to $\mathbf{P}$ as $\mathbf{P}_i = \mathbf{u}_i$, where $\mathbf{u}_i$ is the ith step of $\mathbf{P}$. Note that $\mathbf{P}$ (without an index) is an entire chain-code, but $\mathbf{P}_i$ (with an index) is only one step of $\mathbf{P}$.

The matrix $\mathbf{P}$ and its constituent steps $\mathbf{P}_i$ are relative displacements from a starting index location **s**, resulting in a terminal location **e**.

$$\mathbf{e} = \mathbf{s} + \sum_{i=1}^{n} \mathbf{P}_i \tag{4}$$

The combination of a chain-code $\mathbf{P}$ and a starting location **s** is designated as $\mathbf{P}^{\mathbf{s}}$. $\mathbf{P}_i^{\mathbf{s}}$ designates a specific step of a specific chain-code placed at a specific starting location. Like $\mathbf{P}_i$, $\mathbf{P}_i^{\mathbf{s}}$ is the offset value of a step, but unlike $\mathbf{P}_i$, the location of $\mathbf{P}_i^{\mathbf{s}}$ can be calculated because the starting location of the step's chain-code is known. The absolute location in an image of the chain-code step $\mathbf{P}_i^{\mathbf{s}}$ can be represented as

$$\mathbf{p}_i^{\mathbf{s}} = \mathbf{s} + \sum_{j=1}^{i} \mathbf{P}_j \tag{5}$$

Note that capital **P** is used either by itself to denote an *entire* chain-code or with a subscript to denote the *offset* vector of a single step of a chain-code, while lowercase **p** is used to denote the *location* to which a *single* step of a chain-code points (and therefore requires that the starting location of that chain-code be specified). Because $\mathbf{p}_i^{\mathbf{s}}$ is defined by a sequential iterative process, chain-code location is optimized for sequential access rather than random access.

A chain-code can be *open* or *closed*. If a chain-code is closed, it begins and ends on the same pixel,

$$\mathbf{e} = \mathbf{s} \tag{6}$$

or, put another way, the sum of its steps is the zero-vector,

$$\sum_{i=1}^{n} \mathbf{P}_i = \mathbf{0} \tag{7}$$

Chain-codes may cross themselves, in which case, for some j and k, $j \neq k$, $\mathbf{p}_j^{\mathbf{s}} = \mathbf{p}_k^{\mathbf{s}}$. If a path is closed and does not cross itself, it can be constrained to proceed

in a clockwise or counterclockwise direction, permitting unambiguous determination of inside vs. outside for any given step.

New Notation: Trails

A partial segment of a chain-code, which we designate a trail, may be represented as

$$\mathbf{T}_i = (\mathbf{P}_1, \mathbf{P}_2, ..., \mathbf{P}_i) \tag{8}$$

As with a path, if an arbitrary trail is constrained a-priori such that the trail must start at **s** and the trail's final step i must end at $\mathbf{e}$, then this is designated as $\mathbf{T}_i^{\mathbf{s},\mathbf{e}}$.

Method

NDS makes use of a Markovian chain-code merit function of the form

$$M(\mathbf{P}^{\mathbf{s}}) = \sum_{i=1}^{n} m(\mathbf{p}_i^{\mathbf{s}}, \mathbf{P}_i) \tag{9}$$

where m is an arbitrary local merit function, representing the merit of placing a single chain-code step $\mathbf{P}_i$ at location $\mathbf{p}_i^{\mathbf{s}}$ in the image over which the merit function is defined. It follows that

$$M(\mathbf{T}_i^{\mathbf{s}}) = \begin{cases} M(\mathbf{T}_{i-1}^{\mathbf{s}}) + m(\mathbf{p}_i^{\mathbf{s}}, \mathbf{P}_i), & i > 0 \\ 0, & i = 0 \end{cases} \tag{10}$$

The use of a merit function m allows NDS to regard chain-codes as traveling through either voxel centers or voxel vertices, depending on which interpretation is used within m. This is an important flexibility considering the advantages of each interpretation for certain image processing algorithms. The freedom of defining m also allows NDS to optimize paths over any type of image, whether it be an intensity image, a RGB color image, or even an image whose voxels contain elaborate data structures.

A simple but useful merit function defined over a 2D intensity image is one that rewards a strong unsigned partial spatial derivative of image intensity evaluated at $\mathbf{p}_i^{\mathbf{s}}$ orthogonal to $\mathbf{P}_i$. This merit function, designated as $m_{\partial\perp}$, works well for chain-codes that step between voxel vertices. (If it is known a priori which side of **P** should be lighter or darker, e.g. in closed 2D paths that proceed in a clockwise direction, then a signed partial could be used.)

Another type of merit function is one that performs some form of template correlation, rewarding specific voxel values at $\mathbf{p}_i^{\mathbf{s}}$ and/or in specific neighborhoods relative to $\mathbf{p}_i^{\mathbf{s}}$. One such merit function useful for some 2D segmentations rewards steps where the image value at $\mathbf{p}_i^{\mathbf{s}}$ is consistent with image pixels lying outside the target object, and particular neighbors of $\mathbf{p}_i^{\mathbf{s}}$ have image values consistent with pixels lying inside

the target object. Such a merit function, designated as m_r (r is the standard symbol for correlation), would optimize a closed chain-code to trace through all of the exterior neighbors of a target object, in effect identifying target object pixels as those which lie strictly interior to the resulting closed chain-code.

Given a particular merit function, NDS uses dynamic programming to efficiently find the globally optimal chain-code path $\mathbf{Q}$ within a restricted search space. The initial path must be a "reasonable" approximation of the optimal path, where "reasonable" is a function of the NDS parameters used; a less restricted NDS can find an optimal path that is more divergent from an initial path, but will require more time. The search space is restricted by requiring that each step in the optimal path be based on its corresponding step in the initial path:

$$\mathbf{Q}_i = f(\mathbf{P}_i,\ldots) \tag{11}$$

This restriction has three components:

- Preservation of number of steps
- Limitation of step rotation
- Limitation of step translation

The first restriction is that the optimal chain-code must consist of the same number of steps as the initial chain-code. This restriction does not, however, require that the physical length of the initial and optimal chain-codes be the same because not all $\mathbf{u}_i$ are of the same length. The length (vector magnitude) of a diagonal step that spans three dimensions is physically longer than another step which spans only two, and so on:

$$\sqrt{(\pm 1)^2 + (\pm 1)^2 + (\pm 1)^2} \;>\; \sqrt{(\pm 1)^2 + (\pm 1)^2 + (0)^2} \;>\; \sqrt{(\pm 1)^2 + (0)^2 + (0)^2} \tag{12}$$

The second restriction limits step rotation based on the assumption that the basic shape of the initial path will be correct. Therefore, a path should only be allowed to expand and contract orthogonal to its direction of traversal. This requires that the initial and optimal steps have a positive inner product. The set of possible steps from which $\mathbf{Q}_i$ can be chosen is designated by the set $\mathbf{W}_i$ which contains steps $\mathbf{w}_i$ such that

$$\langle \mathbf{w}_i, \mathbf{P}_i \rangle > 0, \quad \mathbf{w}_i \in \mathbf{W}_i \tag{13}$$

The $\mathbf{W}_i$ for two different 2D $\mathbf{P}_i$ and two different 3D $\mathbf{P}_i$ are shown in Figure 2.

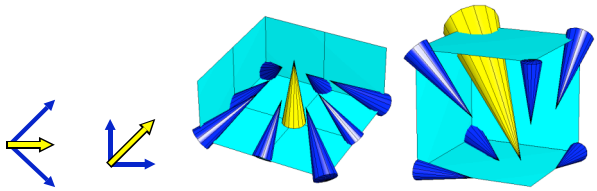

Fig. 2. The set of steps $\mathbf{W}_i$ (blue) that can possibly replace an original step $\mathbf{P}_i$ (yellow)

The final restriction limits the ND translation distance between an optimal step and its corresponding initial step. This limitation is expressed in terms of a hypercube of radius r_f termed a fovea. If a fovea is centered at $\mathbf{p}_i^{\mathbf{s}}$, then $\mathbf{q}_i^{\mathbf{s}}$ must lie within that fovea. A fovea centered at $\mathbf{p}$ can be denoted as $F^{\mathbf{p}}$, and the set of locations within $F^{\mathbf{p}}$ can be denoted as $\mathbf{f}^{\mathbf{p}}$. Increasing r_f allows NDS to find paths increasingly divergent from the initial path but at the cost of drastically increased search time. It is typically much more efficient to use scale space with NDS than it is to increase r_f beyond three or four. Moving the fovea down the length of the initial path sweeps out the ND region in which the optimal path must lie. This region is referred to as the swath of the initial chain-code.

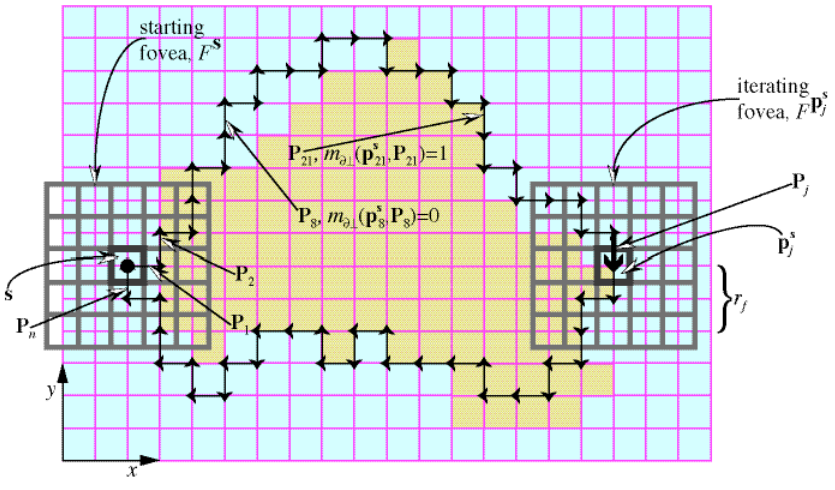

Fig. 3. Fovea iterating along initial path in a 2D image

NDS searches the restricted search space by centering a fovea at $\mathbf{s}$, and then iterating another fovea along $\mathbf{P}^{\mathbf{s}}$ as shown in Figure 3. For each step of the path NDS maintains a list of the set of optimal trails from each location in the starting fovea to each location in the iterating fovea:

$$\left\{\mathbf{T}_i^{\mathbf{s}',\mathbf{e}'}\right\}, \quad \forall \mathbf{s}' \in \mathbf{f}^{\mathbf{s}}, \forall \mathbf{e}' \in \mathbf{f}^{\mathbf{p}_i^{\mathbf{s}}} \tag{14}$$

Each optimal trail of length n is chosen from the set of all concatenations of all length n-1 trails $\mathbf{T}_{i-1}$ with all possible steps in $\mathbf{W}_i$ that result in the desired $\mathbf{s}'$ and $\mathbf{e}'$:

$$\mathbf{T}_i^{\mathbf{s}',\mathbf{e}'} \in \left(\mathbf{T}_{i-1}^{\mathbf{s}',\mathbf{e}'-\mathbf{w}_i}, \mathbf{w}_i\right) \quad \ni \quad \mathbf{w}_i \in \mathbf{W}_i \ \wedge \ \left(\mathbf{e}'-\mathbf{w}_i\right) \in \mathbf{f}^{\mathbf{p}_{i-1}^{\mathbf{s}}} \tag{15}$$

Note that the necessity of subtracting $\mathbf{w}_i$ from $\mathbf{e}'$ will result in some $\mathbf{T}_{i-1}$ with ending locations outside of the swath. Because such trails would not lie within the restricted search space, it is important that these cases be eliminated from the list of possible trails.

Because the merit of a trail is recursively defined, all that is needed to determine the merit of a trail is local merit of the new trail's final step and the already calculated merit of the rest of the trail:

$$M\left(\mathbf{T}_i^{\mathbf{s}',\mathbf{e}'}\right) = M\left(\mathbf{T}_{i-1}^{\mathbf{s}',\mathbf{e}'-\mathbf{w}_i}\right) + m\left(\mathbf{e}',\mathbf{w}_i\right) \tag{16}$$

Once the iterating fovea reaches the end of the initial chain-code **P**, the set of optimal trails from every location in $\mathbf{f}^{\mathbf{s}}$ to every location in $\mathbf{f}^{\mathbf{e}}$ has been found. The optimum path **Q** is then the $\mathbf{T}_n^{\mathbf{s}',\mathbf{e}'}$ with the largest merit:

$$\mathbf{Q} = \mathbf{T}_n^{\mathbf{s}'',\mathbf{e}''} \quad \ni \quad M\left(\mathbf{T}_n^{\mathbf{s}'',\mathbf{e}''}\right) = \max_{(\mathbf{s}',\mathbf{e}')} M\left(\mathbf{T}_n^{\mathbf{s}',\mathbf{e}'}\right) \tag{17}$$

If **P** was closed, then **Q** must be closed as well, and so $\mathbf{s}' = \mathbf{e}'$ must be enforced:

$$\mathbf{Q} = \mathbf{T}_n^{\mathbf{s}'',\mathbf{s}''} \quad \ni \quad M\left(\mathbf{T}_n^{\mathbf{s}'',\mathbf{s}''}\right) = \max_{\mathbf{s}'} M\left(\mathbf{T}_n^{\mathbf{s}',\mathbf{s}'}\right) \tag{18}$$

Implementation in ITK

ITK provided an excellent framework within which to implement NDS. The ITK iterator and neighborhood iterator constructs were especially helpful in handling the complexity of a ND fovea, and the shaped neighborhood iterator provided an efficient means of dealing with the $\mathbf{W}_i$ for a given $\mathbf{P}_i$.

Trail storage was handled by making a fovea-sized image representative of each trail starting location $\mathbf{f}^{\mathbf{s}}$ in $F^{\mathbf{s}}$, and then having each voxel of that image point to another fovea-sized image-of-steps representative of each trail ending location $\mathbf{f}^{\mathbf{p}_i^{\mathbf{s}}}$ in the iterating fovea $F^{\mathbf{p}_i^{\mathbf{s}}}$. In order to store every *step* of every optimal trail for every length of trail, such an image-of-images-of-steps must be calculated for each step of the initial path, but since the *merit* of a trail of length *i* depends only on the merit of its last step and the merit of a trail of length *i*-1, only the most recent two images-of-images-of-merits must be stored.

The implementation used a fovea-sized neighborhood iterator centered at $\mathbf{p}_i^{\mathbf{s}}$ to iterate through the input image voxel indices used for $\mathbf{e}'$ for each length of trail. The neighborhood was made to follow the course of the initial path, and for each step $\mathbf{P}_i$ of the initial path, the corresponding set $\mathbf{W}_i$ was calculated. Because the merit of each trail feeding each corresponding $\mathbf{w}_i$ would be necessary, an ITK shaped neighborhood was then created to visit only merit values of the trails feeding each $\mathbf{w}_i \in \mathbf{W}_i$. Since the feeding trails' ending locations are offset from the new ending locations by $\mathbf{P}_i$, each $\mathbf{w}_i$ was subtracted from $\mathbf{P}_i$ to transform $\mathbf{W}_i$ into a list of offsets from the beginning of $\mathbf{P}_i$ to the beginnings of the $\mathbf{w}_i$, as was necessary to create the appropriate shaped neighborhood. $\mathbf{W}_i$ could be recovered from the list of shaped neighborhood offsets by simply subtracting the offsets from $\mathbf{P}_i$. The shaped neighborhood was given a constant boundary condition of -∞ to prevent $\mathbf{w}_i$ not completely within the swath from being chosen. In effect, a single shaped neighborhood provided not only $\mathbf{W}_i$, but for each $\mathbf{w}_i$ it also provided the merit value of the corresponding feeder-trail of length *i*-1. Once all the optimal trails had been grown to length *n*, *the* optimal trail $\mathbf{Q}$ was found using a rather straight-forward standard iterative search.

Discussion

NDS uses dynamic programming to efficiently search for a globally optimum path. Dynamic programming is not a new technique for curve detection [5], but it has traditionally been used in relatively small search spaces for 2D images. NDS, however, searches through a high-dimension search space in polynomial time. It is only recently that computational power has become sufficient for NDS to execute in reasonable run time, taking only a few seconds on a typical desktop computer to search for an optimal 100 step chain-code using a fovea of radius 3 (higher search radii can be efficiently achieved by use of scale space).

NDS must store and process a huge amount of data. Let the number of elements across one side of the fovea be denoted by $d_f = 2 * r_f + 1$, and let the number of dimensions of the input image be denoted by D. Then, the number of optimal trails from every element in $F^{\mathbf{s}}$ to every element in $F^{\mathbf{p}_i^{\mathbf{s}}}$ will be d_f^{D*2}, and *n* steps will have to be selected for each of those optimal trails. Selecting a step requires that each $\mathbf{w}_i$ be evaluated, and so the number of steps for which the sum of local and previous merit must be evaluated is

$$d_f^{D*2} * n * |\mathbf{W}_i| \tag{19}$$

Because trails between different fovea elements overlap and because the foveas of neighboring steps overlap, many identical local merit queries will be needed to calculate the different total merit values of each $\mathbf{w}_i$ appended to each trail. Therefore, caching local-merit-values can substantially reduce the number of times the local merit function must be evaluated, in the best case (a straight path) reducing it to approximately

$$d_f^{D-1} * n * |\mathbf{W}_i| \tag{20}$$

$|\mathbf{W}_i|$ varies with D and with the direction of each step $\mathbf{P}_i$. As can be seen from Figure 2, $\mathbf{P}_i$ that span a larger subspace (diagonal steps) can have fewer $\mathbf{w}_i$ than $\mathbf{P}_i$ that span only a few dimensions. Let $D_i' \le D$ represent the dimensionality of the minimum subspace that can contain $\mathbf{P}_i$. For example, if $\mathbf{P}_i = (1 \ 0 \ -1)^{\mathrm{T}}$, then $D_i' = 2$. It works out that

$$|\mathbf{W}_i| = \left(2^{D_i'} - 1\right) * 3^{D-D_i'} \tag{21}$$

Therefore, NDS grows exponentially with image dimension, but it has polynomial growth with respect to fovea size (search radius) and linear growth with respect to path length. Use of scale space to reduce the search radius and merit caching to avoid redundant computation can substantially reduce the large polynomial growth, however.

NDS is a powerful tool for optimization of path-based segmentations. Its only significant shortcoming with regard to quality of result is that it cannot change the number of steps in a chain-code. This is not typically a problem, however, because NDS can "pinch" a chain-code to remove extra steps (as shown at the top-right corner of Figure 4 and such pinched segments can be easily removed from the output of NDS. If significant path growth may be needed, extra steps can be added to the input path without qualitatively changing it by using the path-to-chain-code conversion filter to convert a path to a minimally-connected chain code, all the steps of which are of minimum physical length.

Figure 4 shows an example result of NDS applied to the problem of 2D segmentation. The fovea radius was 3, and the merit function used was m_r which optimized the path to trace around the outside of a dark-gray object. Blurring the square had no effect on the resulting optimal path.

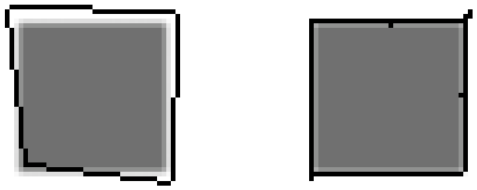

Fig. 4. Input image of a blurred square with the input path (left) and output path (right)

Conclusion

In this paper the novel NDS algorithm for n-dimensional path optimization was developed and implemented in ITK. NDS uses dynamic programming to efficiently search for a globally optimal path, where optimality is defined in terms of an external local merit function. NDS execution time is linear with path length, polynomial with search radius, and exponential with image dimension.

Acknowledgements

The authors would like to thank everyone who has contributed to ITK. This work has been supported in part by a contract with the National Library of Medicine and a NSF Graduate Student Fellowship.

References

1. Chen, C., Huang, T., and Arrot, M., 1994. Modeling, Analysis, and Visualization of Left Ventricle Shape and Motion by Hierarchical Decomposition, PAMI, 16(4), pp. 342-356.
2. Geiger, D., Gupta, A., Costa, L., and Vlontzos, J., 1995. Dynamic Programming for Detecting, Tracking, and Matching Deformable Contours, PAMI 17(3), pp. 294-302.
3. Gunn, S., Nixon, M., 1997. A Robust Snake Implementation: A Dual Active Contour, PAMI 19(1), pp. 63-68.
4. Stetten, G., Drezek, R., 2001. Active Fourier Contour Applied to Real Time 3D Ultrasound of the Heart, International Journal of Image and Graphics, 1(4), pp. 647-658.
5. Montanari, U., 1971. On the optimal detection of curves in noisy pictures. Communications of the ACM, 14(5), pp. 335-345.
6. Pope, Parker, 1984. Dynamic Search Algorithms in Left Ventricular Border Recognition and Analysis of Coronary Arteries. Computers in Cardiology.
7. Galeotti, J., Stetten, G., 2005. Creation and Demonstration of a Framework for Handling Paths in ITK, ISC/NA-MIC/MICCAI Workshop on Open-Source Software, http://hdl.handle.net/1926/40.