Technical Report

Pattern Recognition and Image Processing Group
Institute of Visual Computing and Human-Centered Technology
TU Wien
Favoritenstrasse 9-11/193-03
A-1040 Vienna AUSTRIA
Phone: +43 (1) 58801 - 18661
Fax: +43 (1) 58801 - 18697
E-mail: sek@prip.tuwien.ac.at
URL: http://www.prip.tuwien.ac.at/

PRIP-TR-150 July 15, 2021

Students' SS 2021 contributions:
186.837 Seminar of Computer Vision and Pattern Recognition

Alice Barbara Tumpach and Andreas Wiesinger

edited by Walter G. Kropatsch and Jiří Hladůvka
Institute of Visual Computing and HCT
Pattern Recognition and Image Processing Group 193-03

Abstract

This technical report presents a selection of papers submitted by students in the course Seminar of Computer Vision and Pattern Recognition (SE 186.837) of the Pattern Recognition and Image Processing group during summer term 2021.

Contents

Canonical parameterizations of $2D$-shapes

Alice Barbara TUMPACH*

Abstract

This paper is devoted to the study of unparameterized simple curves in the plane. We propose diverse canonical parameterization of a 2D-curve. For instance, the arc-length parameterization is canonical, but we consider other natural parameterizations like the parameterization proportionnal to the curvature of the curve. Both aforementionned parameterizations are very natural and correspond to a natural physical movement : the arc-length parameterization corresponds to travelling along the curve at constant speed, whereas parameterization proportionnal to curvature corresponds to a constant-speed moving frame. Since the curvature function of a curve is a geometric invariant of the unparameterized curve, a parameterization using the curvature function is a canonical parameterization. The main idea is that to any physically meaningful stricktly increasing function is associated a natural parameterization of 2D-curves, which gives an optimal sampling, and which can be used to compare unparameterized curves in a efficient and pertinent way. An application to point correspondance in medical imaging is given.

1 Introduction

1.1 Motivation

Curves in $\mathbb{R}^2$ appear in many applications: in shape recognition as outline of an object, in radar detection as the signature of a signal, as trajectories of cars etc... There are two main features of the curve : the route and the speed profil. In this paper, we are only interested in the route drawn by the curve and we will called it the unparameterized curve. An unparameterized curve can be parameterized in multiple ways, and the choosen parameterization selects the speed at which the curve is traversed. Hence a curve can be travelled with many different speed profils, like a car can travel with different speeds (not necessarily constant) along a given road. The choice of a speed profil is called a parameterization of the curve. It may be physically meaningful or not. For instance, depending on applications, there may not be any relevant parameterization of the contour of the statue of Liberty depicted in Figure 2. In this paper, we propose diveres very natural parameterization of 2D-curves. They are based on the curvature, which together with the arc-length measure form a complete set of geometric invariants or descriptors of the unparameterized curves.

1.2 Past work

In this section, we look at the geometry of the space of 2D-curves in the plane, and give an overview of past work on the analysis of contours, comprising comparison and interpolation tasks.

When we want to interpolate between two 2D-contours like the two red contours of a ballerina depicted in Figure 1, there are a couple of things to keep in mind.

First, one should not take two parameterizations of the contours at random and interpolate linearly between them. This gives usually very bad results, even when one starts the parameterization at points that should correspond. In Figure 1, we start the parameterization at the top of the head of the ballerina, travel the contour counterclockwise with a speed profil that is illustrated by the sampling of the curves : on portions of the curve where points accumulate the speed is small, whereas on portions of the curve where points are very far apart, the curve is travelled at high speed (in order to travel between two successive points, the same amount of time is needed). The resulting interpolation is depicted on the first line of Fig. 1. One sees that this interpolation procedure does not give good results.

*`alice-barbora.tumpach@univ-lille.fr` is with Univ. Lille, CNRS, UMR 8524 - Laboratoire Paul Painlevé, F-59000 Lille, France and with Pauli Institut, Vienna, Austria.

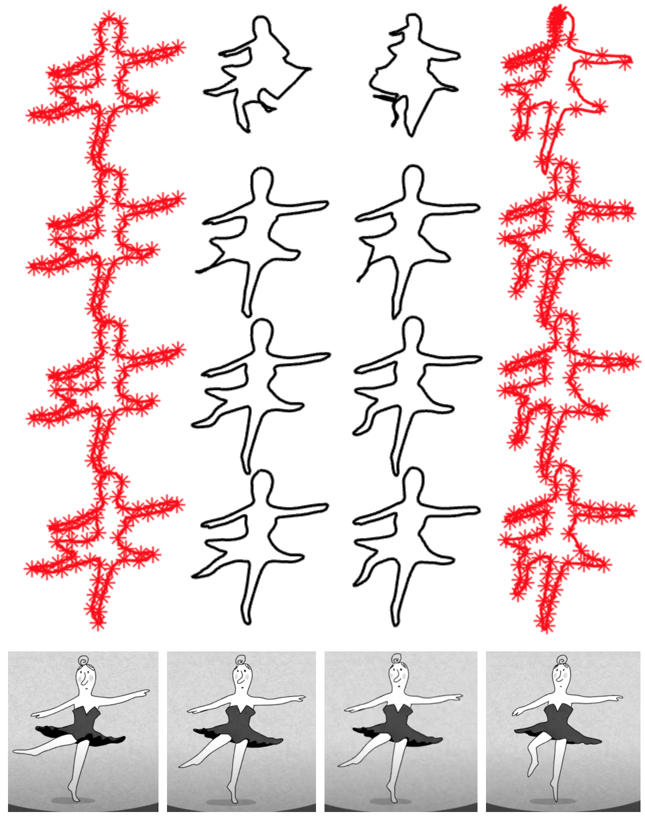


Figure 1 – First line : linear interpolation between some parameterized ballerinas, second line : linear interpolation between arc-length parameterized ballerinas, third line : linear interpolation between two registered ballerinas using [1], and [2], fourth line: linear interpolation between two registered ballerinas using a function of arc-length and curvature, fifth line : reference movement taken from [3].

Second, picking up a preferred parameterization of the curves, for instance the arc-length parameterization, and interpolating linearly between the parameterized curves may also lead to bad results. In the second row of Figure 1, the two ballerinas are parameterized proportionally to arc-length hence the resulting samplings are uniform. The linear interpolation between the two contours parameterized proportional to arc-length is depicted at the second row in Figure 1. One can see that the deformation shrinks the moving leg and therefor appears unnatural. However, in some applications, where the routes to compare are very similar, the result may be satisfactory and no fancy shape analysis is needed.

One can distinguish two tasks in the comparison of curves :

1. the registration or correspondance, which consists in choosing parameterizations of two curves so that features of the curves that should correspond are associated to the same value of the parameter,
2. the measurement of the discrepancy between the two curves and the generation of a deformation of one curve into the other.

The recent use of differential geometry in shape analysis has allowed to take on these two tasks in the same framework. A traditional strategy to generate deformations between unparameterized curves is the following : the space of parameterized curves is endowed with a parameterization-equivariant Riemannian metric which allows to compute preferred deformations between curves, called geodesics, which are minimal for the corresponding variational problem. Then, given two unparameterized curves, one chooses the parameterization of one curve and, to each parameterization of the second curve, one compute the geodesic (if it exists!) between the two parameterized curves. At last, one has to solve an optimization problem consisting in singeling out the parameterization of the second curve (if it exists!) that achieve the infimum of the cost function among all possible parameterizations of the second curve. The geodesic between the two unparameterized curves is then given by the geodesic between the first curve (with its arbitrarily choosen parameterization) and the second curve with the parameterization minimizing the cost function. The discrepancy between the two curves is measured as the length of this geodesic. This procedure endow the shape space with a Riemannian structure called the quotient Riemannian metric. One can mention the following problems encountered when one pursue this strategy :

- The choice of a Riemannian metric on the space of parameterized curves is usually not easy. As was first highlighted in [4], a badly chosen Riemannian metric can lead to vanishing geodesic distance, ruling out any effort to use geodesic distance to measure discrepancy between curves. For this reason, a large mathematical literature developped in order to propose Riemannian metrics with good mathematical properties: mention Sobolev metrics in [5], curvature weighted metrics in [6], almost local metrics in [7], metrics mesuring the deformations of the interiors of shapes in [8].
- The geodesic between two parameterized curves with respect to a given Riemannian metric are usually hard to compute, and one has to use algorithms like the path-straightening method or the shooting method to approximate them ([9]). These algorithms are time-consuming. To overcome this difficulty and speed up the comparison of curves, some metrics have been proposed where the geodesic on the space of parameterized curves are explicit, like in [10], [11], or [1] and [2]. The framework of [2] has recently been adapted to general manifolds in [12] and homogeneous spaces in [13].
- The optimization problem over all parameterizations of a given curve raise mathematical as well as practical difficulties : first the set of all parameterizations of a given curve is an orbit of an infinite-dimensional Fréchet Lie group, the group of diffeomorphisms, with a lot of pathological properties. There is in general no guarantee that this mathematical problem can be solved. Second, the algorithms used to approximate the solution of this optimization problem are based on dynamical programming (see for instance [14]) with the drawback that in practise only a finite number of reparameterizations distributed mainly around the identity map are considered. For this reason, a gauge invariant framework has been proposed in [15] (see also [16]) in the context of shape analysis of surfaces, where this optimization step is avoided by the use of a Riemannian metric which degenerates along the orbit of the reparameterization group. Another idea to avoid this minimization problem was proposed in [17], where the quotient elastic metric introduced in [2] is expressed as a metric on the section of arc-length parameterized curves. Nevertheless, since the geodesics on shape space are not explicit in any of the previously mentionned works, shape comparison is not really efficient.
- In [18], the optimization step is solved for piecewise linear curves (polygons) under the elastic metric of [19]: the precise matching minimizing the geodesic distance is given between two piecewise linear curves. The only lack in this work is that it relies on the Euclidean geometry of $\mathbb{R}^n$ and may not be adapted to general manifolds or homogeneous spaces.

2 Different parameterizations of 2D-shapes

2.1 Arc-length parameterization and Signed curvature

By 2D-shape, we mean the shape drawn by a parameterized curve in the plane. It is the ordered set of points visited by the curve. The shapes of two curves are identical if one can reparameterize one curve into the other (using a continuous increasing function). Any rectifiable planar curve admits a canonical parameterization, its *arc-length parameterization*, which draws the same shape, but with constant unit speed. The set of 2D-shapes can be therefore identified with the set of arc-length parameterized curves, which is not a vector subspace, but rather an infinite-dimensional submanifold of the space of parameterized curves (see [17]).

It may be difficult to compute an explicit formula of the arc-length parameterization of a given rectifiable curve. Fortunately, when working with a computer, one do not need it. One neither need a concrete parameterization of the curve to depict it, a sample of points on the curve suffises. To draw the statue of Liberty as in Figure 2, left, one just need a finite ordered set of points (the red stars). The discrete version of an arc-length parameterized curve is a uniformly sampled curve, i.e. an ordered set of equally distant points (for the euclidean metric). Resampling a curve uniformly is immediate using some appropriate interpolation function like the matlab function *spline* (the second picture in Figure 2 shows a uniform resampling of the statue of liberty).

Consider the set of 2D simple closed curves, such as the contour of Elie Cartan's head in Figure 3. After the choice of a starting point and a direction, there is a unique way to travel the curve at unit

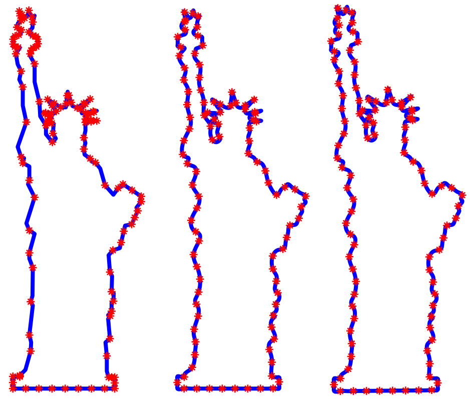


Figure 2 – The statue of Liberty (left), a uniform resampling using Matlab function spline (middle), a reconstruction of the statue using its discrete curvature (right).

speed. In Figure 3, we have drawn the velocity vector near the glasses of Elie Cartan, as well as the unit normal vector which is obtained from the unit tangent vector by a rotation of $+\frac{\pi}{2}$. These two vectors form an orthonornal basis, i.e. an element (modulo the choice of a basis of $\mathbb{R}^2$) of the Lie group SO(2), which is characterized by a rotation angle. The rate of variation of this rotation angle is called the signed curvature of the curve. For instance, when moving along the external outline of the glasses, this curvature equals the inverse of the radius of the glasses.

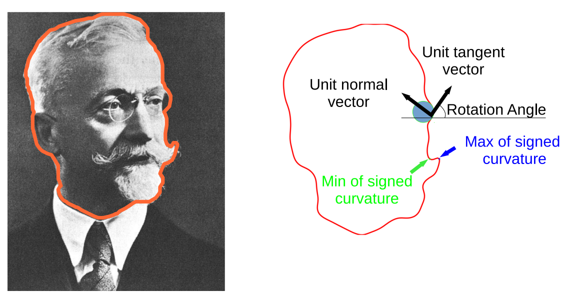


Figure 3 – Elie Cartan and the moving frame associated to the contour of his head.

There is a little difference in the construction of the moving frame for 2D curves in comparison to the moving frame for 3D curves. Indeed in the 2D case, we don't need the second and third derivative of the curve to construct the frame. Just the first derivative is enough. In fact we are using the knowledge that the curve stays in the plane to construct the normal at each point of the curve. In other words, we are using additional geometric properties of the ambient space (in this case the complex structure of the plane). The consequence of this is that the moving frame can be defined even for 2D curves with flat pieces (zero curvature sections) like the statue of Liberty which has a long flat piece at its base (see Figure 2). The corresponding curvature is therefore signed, with positive sign when the moving frame is turning clockwise, negative sign when the moving frame is turning counterclockwise, and zero along flat pieces. We have depicted the curvature function κ of Elie Cartan's head in Figure 4, first line, when the parameter $s \in [0;1]$ on the horizontal axis is proportional to arc-length, and such that the entire contour of Elie Cartan's head is travelled when the parameter reaches 1. Its corresponds to a uniform sampling of the contour.

A discrete version of an arc-length parameterized curve is an equilateral polygon. To draw an equilateral polygon, one just need to know the length of the edges, the position of the first edge, and the angles between two successive edges. The sequence of turning-angles is the discrete version of the curvature and defines a equilateral polygon modulo scaling, rotation and translation. In Figure 2 , right, we have reconstructed the statue of Liberty using the discrete curvature function.

In order to interpolate between two parameterized curves, it is easier when the domains of the parameter coincide. For this reason we will always consider curves parameterized with a parameter in $[0;1]$. A natural parameterization is then the parameterization proportional to arc-length. It is obtain from the parameterization by arc-length by dividing the arc-length parameter by the length of the curve

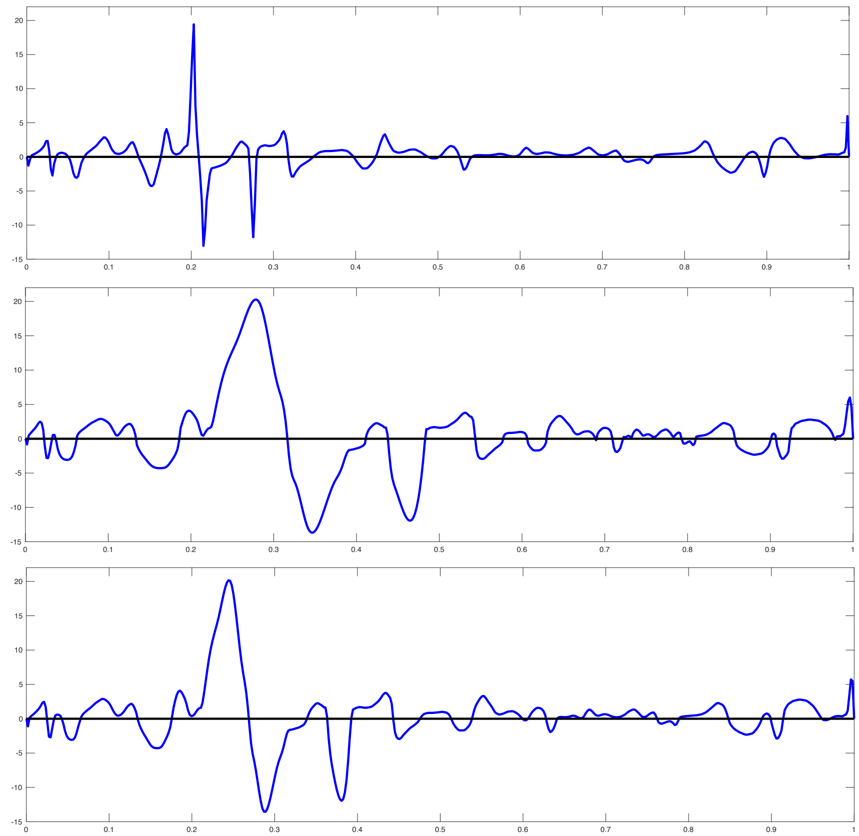


Figure 4 – Signed curvature of Elie Cartan's head for the parameterization proportional to arc-length (first line), proportional to the curvature-length (second line), and proportional to the curvarc length (third line).

L. The corresponding curvature function is also defined on $[0;1]$ and is obtained from the curvature function parameterized by arc-length by compressing the x-axis by a factor L. To recover the initial curve from the curvature function associated to the parameter $s \in [0;1]$ proportional to arc-length, one only need to know the length of the curve.

2.2 Parameterization proportional to curvature-length

In the same spirit as the scale space of T. Lindeberg ([20]), and the curvature scale space of Mackworth and Farcin Mokhtarian ([21]), we now define another very natural parameterization space of 2D curves. Its relies on the fact that the integral of the absolute value of the curvature κ is an increasing function on the interval $[0;1]$, stricktly increasing when there are no flat pieces. In that case the function

$$r(s) = \frac{\int_0^s |\kappa(s)| ds}{\int_0^1 |\kappa(s)| ds} \tag{1}$$

(where κ denotes the curvature of the curve) belongs to the group of orientation preserving diffeomorphisms of the parameter space $[0;1]$, denoted by $\mathrm{Diff}^+([0;1])$. Note that its inverse $s(r)$ can be computed graphically using the fact that its graph is the symmetric of the graph of $r(s)$ with respect to $y = x$. The contour of Elie Cartan's head can be reparameterized using the parameter $r \in [0;1]$ instead of the parameter $s \in [0;1]$. In Figure 5 upper left, we have depicted the graph of the function $s \mapsto r(s)$. A uniform sampling with respect to the parameter r is obtain by uniformly sampling the vertical-axis (this is materialized by the green equidistributed horizontal lines) and resampling Elie Cartan's head at the sequence of values of the s-parameter given by the abscissa of the corresponding points on the graph of r (where a green line hits the graph of r a red vertical line materializes the corresponding abscissa). One sees that this reparameterization naturally increases the number of points where the 2D contour is the most curved, and decreases the number of points on nearly flat pieces of the contour. For a given number of points, it gives an optimal way to store the information contained in the contour.

The quantity

$$C = L \int_0^1 |\kappa(s)| ds, \tag{2}$$

where $s \in [0;1]$ is proportional to arc-length, is called the *total curvature-length* of the curve. It is the length of the curve drawn in SO(2) by the moving frame associated with the arc-length parameterized curve.For this reason we call this parameterization the *parameterization proportional to curvature-length*. In the right picture of Figure 5, we show the corresponding resampling of the contour of Elie Cartan's head.

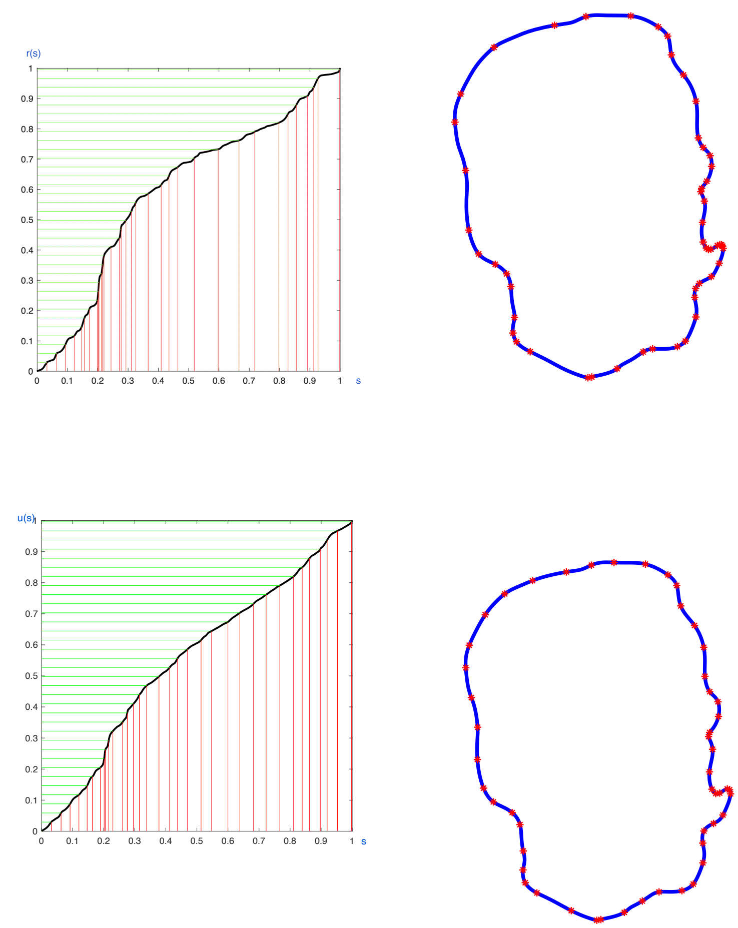


Figure 5 – First line : Integral of the (renormalized) absolute value of the curvature (left), and corresponding resampling of Elie's Cartan head (right). Second line : Integral of the (renormalized) curvarc length (left), and corresponding resampling of Elie's Cartan head (right).

This resampling can naturally be adapted in the case of flat pieces resulting in a sampling where there is no points between two points on the curve joint by a straight line. In the left picture of Figure 6, we have depicted a sampling of the statue of Liberty proportional to curvature-length. Note that there are no points on the base of the statue. The corresponding parameterization has the advantage of concentrating on the pieces of the contour that are very complex, i.e. where there is a lot of curvature, and not distributing points on the flat pieces which are easy to reconstruct (connecting two points by a straight line is easy, but drawing the moustache of Elie Cartan is harder and needs more information).

As in section 2, it is possible to reconstruct a curve from its curvature function parameterized proportionally to curvature-length, provided that we know the length of the curve L and its total curvature-length C, and provided that there is no flat piece. Indeed, derivating equation (1), one obtains $dr = \frac{|\kappa(s)|}{C} L ds$, where Lds is the arc-length measure of the curve.

The drawback of using the parameterization proportional to curvature-length is that one can not reconstruct the flat pieces of a shape without knowing their lengths (remember that the parameterization proportional to curvature-length put no point at all on flat pieces). For this reason we propose a parameterization intermediate between arc-length parameterization and curvature-length parameterization. We call it *curvarc-length parameterization*.

2.3 Curvarc-length parameterization

In order to define the curvarc-length parameterization, we consider the triple $(P(s), \vec{v}(s), \vec{n}(s))$, where $P(s)$ is the point of the shape parameterized proportionally to arc-length with $s \in [0;1]$, $\vec{v}(s)$ and $\vec{n}(s)$ the corresponding unit tangent vector and unit normal vector respectively. It defines an element of the group of rigid motions of $\mathbb{R}^2$, called the special Euclidean group and denoted by $\mathrm{SE}(2) := \mathbb{R}^2 \rtimes \mathrm{SO}(2)$. The point $P(s)$ corresponds to the translation part of the rigid motion, it is the vector of translation needed to move the origin to the point of the curve corresponding to the parameter value s. The moving frame $O(s)$ defined by $\vec{v}(s)$ and $\vec{n}(s)$ is the rotation part of the rigid motion. One has the following equations :

$$\frac{dP}{ds} = L\vec{v}(s) \quad \text{and} \quad O(s)^{-1}\frac{d}{ds}O(s) = \begin{pmatrix} 0 & -\kappa(s) \\ \kappa(s) & 0 \end{pmatrix}, \tag{3}$$

where L is the length of the curve. Endow $\mathrm{SE}(2) := \mathbb{R}^2 \rtimes \mathrm{SO}(2)$ with the structure of a Riemannian manifold, product of the plane and the Lie group $\mathrm{SO}(2) \simeq \mathbb{S}^1$. Than the norm of the tangent vector to the curve $s \mapsto (P(s), \vec{v}(s), \vec{n}(s))$ is $L + |\kappa(s)|$. Therefore the length of the SE(2)-valued curve is $L + \int_0^1 |\kappa(s)|ds = L + \frac{C}{L}$. We call it the total curvarc-length. It follows that the following function

$$u(s) = \frac{\int_0^s (L + |\kappa(s)|)ds}{\int_0^1 (L + |\kappa(s)|)ds}$$

defines a reparameterization of $[0;1]$. The arc-length parameter of the initial shape is related to the parameter u by

$$Lds = \frac{L^2 + C}{L + |\kappa(u)|}du.$$

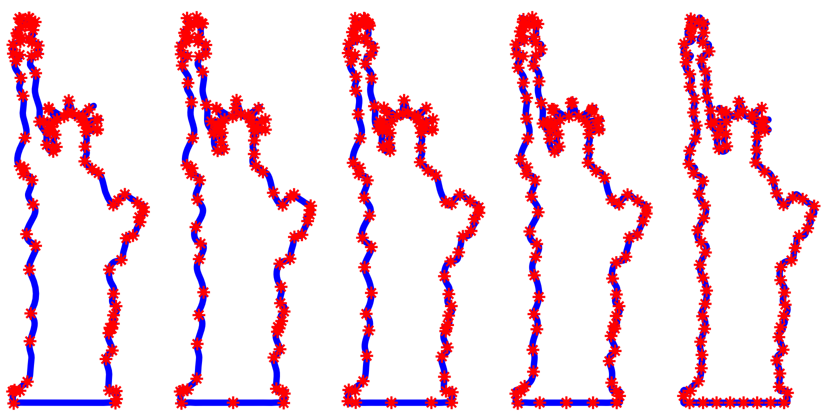


Figure 6 – Resampling of the statue of Liberty proportional to the intergral of λ + curvature, for (from left to right) $\lambda = 0; \lambda = 0.3; \lambda = 1; \lambda = 2; \lambda = 100$.

3 Application to medical imaging : parameterization of bones

In the analysis of diseases like Rheumatoid Arthritis, one uses X-ray scans to evaluate how the disease affectes the bones. One effect of Rheumatoid Arthritis is erosion of bones, another is joint shrinking. In order to measure joint space, one has to solve a point correspondance problem. For this, one uses landmarks along the contours of bones. These landmarks have to be placed at the same anatomical positions for every patient. Below they are placed using a method by Hans Henrik Thodberg ([22]), based on minimum description length which minimizes the description of a PCA model capturing the variability of the landmark positions. For instance in Figure 7 left, the landmark number 56 should

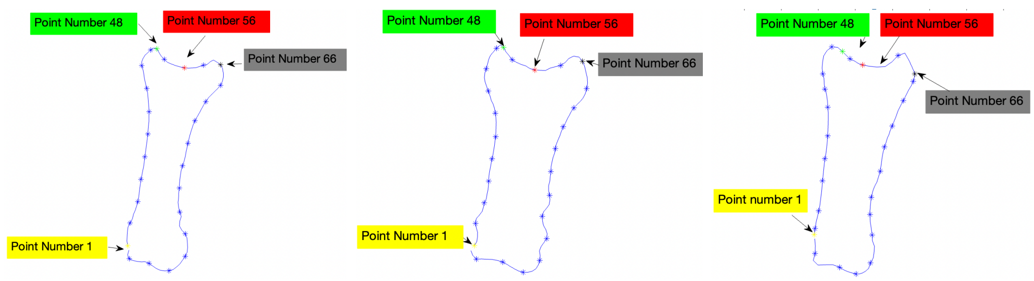


Figure 7 – Point correspondance on 3 different bones using the method of [22]

always be in the middle of the head of the bone because it is used to measure the width between two adjacent bones in order to detect rheumatoid arthritis.

Although the method by Hans Henrik Thodberg gives good results, it is computationally expensive. In this paper we propose to recover similar results with an quicker algorithm. It is based on the fact that any geometrically meaningful parameterization of a contour can be expressed using the arc-length measure and the curvature of the contour, which are the only geometric invariants of a 2D-curve (modulo translation and rotation). It follows that the parameterization calculated by Thodberg's algorithm should be recovered as a parameterization expressed using arc-length and curvature. We investigate a 2 parameter family of parameterizations defined by

$$u(s) = \frac{\int_0^s (c * L + |\kappa(s)|^\lambda) ds}{\int_0^1 (c * L + |\kappa(s)|^\lambda) ds} \tag{4}$$

where c and λ are positive parameters and where L is the length of the curve and κ its curvature function. We recover an analoguous parameterization to the one given by Thodberg's algorithm with $c = 1$ and $\lambda = 7$. The algorithm computes the parameterization of 14 bones in 2.23s on a Mac M1.

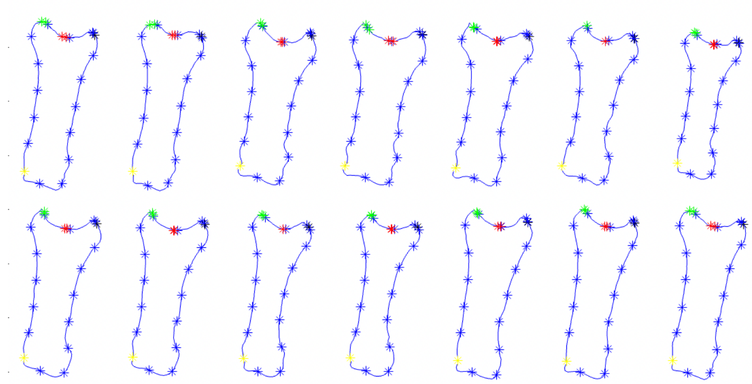


Figure 8 – 14 bones parameterized by Thodberg's algorithm on one hand and the parameterization defined by (4) with $c = 1$ and $\lambda = 7$ on the other hand (the two parameterizations are superposed). The colored points corresponds to points labelled 1, 48, 56, 66. They overlap for the two methods.

4 Conclusion

We proposed diverse canonical parameterization of 2D-contours, which are expressed using arc-length and curvature of curves. The curvature-length parameterization and the curvarc-length parameterization are very natural examples, since they corresponds to a constant-speed moving frame in $SO(2)$ and $SE(2)$. We present an application to the problem of point correspondance in medical imaging consisting of labelling automatically keypoints along the contour of bones. We recover an analoguous parameterization to the one proposed by Thodberg at real-time speed. Having a two-parameter family of parameterizations at our disposal, a fine-tuning can be applied on top of our results in order to improve the point correspondance further.

References

[1] S. H. Joshi, A. Srivastava, E. Klassen, and I. Jermyn, "A novel representation for computing geodesics between n-dimensional elastic curves," in *IEEE Conference on computer Vision and Pattern Recognition (CVPR)*, 2007.

[2] A. Srivastava, E. Klassen, S. Joshi, and I. Jermyn, "Shape analysis of elastic curves in euclidean spaces," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 33, no. 7, pp. 1415–1428, 2011.

[3] A. Sugano, "The physics of the hardest move in ballet," 2017. [Online]. Available: http://ed.ted.com/lessons/the-physics-of-the-hardest-move-in-ballet-arleen-sugano

[4] P. W. Michor and D. Mumford, "Vanishing geodesic distance on spaces of submanifolds and diffeomorphisms," *Doc. Math.*, vol. 10, pp. 217–245, 2005.

[5] M. Bauer, P. Harms, and P. W. Michor, "Sobolev metrics on shape space of surfaces," *Journal of Geometric Mechanics*, vol. 3, no. 4, pp. 389–438, 2011.

[6] ——, "Curvature weighted metrics on shape space of hypersurfaces in n-space," *Differential Geom. Appl.*, vol. 30, pp. 33–41, 2012.

[7] ——, "Almost local metrics on shape space of hypersurfaces in n-space," *SIAM J. Img. Sci.*, vol. 5, no. 1, pp. 244–310, Mar. 2012.

[8] M. Fuchs, B. Jüttler, O. Scherzer, and H. Yang, "Shape metrics based on elastic deformations," *J. Math. Imaging Vis.*, vol. 35, no. 1, pp. 86–102, 2009.

[9] H. Drira, A. Tumpach, and M. Daoudi, "Gauge invariant framework for trajectories analysis," in *Proceedings of the 1st International Workshop on DIFFerential Geometry in Computer Vision for Analysis of Shapes, Images and Trajectories*, 2015.

[10] L. Younes, "Computable elastic distances between shapes," *SIAM J. of Applied Math*, pp. 565–586, 1998.

[11] L. Younes, P. W. Michor, J. Shah, and D. Mumford, "A metric on shape space with explicit geodesics." *Matematica E Applicazioni*, no. 19, pp. 25–57, 2008.

[12] A. L. Brigant, "Computing distances and geodesics between manifold-valued curves in the srv framework," 2017. [Online]. Available: arXiv:1601.02358v4

[13] E. Celledoni, S. Eidnes, and A. Schmeding, "Shape analysis on homogeneous spaces," 2017. [Online]. Available: arXiv:1704.01471v1

[14] W. Mio, A. Srivastava, and S. H. Joshi, "On shape of plane elastic curves," *International Journal of Computer Vision*, vol. 73, no. 3, pp. 307–324, 2007.

[15] A. B. Tumpach, H. Drira, M. Daoudi, and A. Srivastava, "Gauge invariant framework for shape analysis of surfaces," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 38, no. 1, pp. 46–59, 2016.

[16] A. B. Tumpach, "Gauge invariance of degenerate Riemannian metrics," *Notices of the American Mathematical Society*, vol. 63, no. 4, pp. 342–350, 2016.

[17] A. B. Tumpach and S. C. Preston, "Quotient elastic metrics on the manifold of arc-length parameterized plane curves," *Journal of Geometric Mechanics*, vol. 9, no. 2, pp. 227–256, 2017.

[18] S. Lahiri, D. Robinson, and E. Klassen, "Precise matching of PL curves in $\mathbb{R}^n$ in the square root velocity framework," pp. 133–186, 2015. [Online]. Available: arXiv:1501.00577

[19] A. Srivastava, E. Klassen, S. H. Joshi, and I. H. Jermyn, "Shape analysis of elastic curves in euclidean spaces," *IEEE Trans. PAMI*, no. 33, pp. 1415–1428, 2011.

[20] T. Lindeberg, "Image matching using generalized scale-space interest points," *Journal of Mathematical Imaging and Vision*, vol. 52, no. 1, pp. 3–36, 2015.

[21] F. Mokhtarian and A. K. Mackworth, "A theory of multiscale, curvature-based shape representation for planar curves," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 14, no. 8, pp. 789–805, 1992.

[22] H. Thodberg, "Minimum description length shape and appearance models," *Proceedings of Information Processing in Medical Imaging (IPMI 2003)*, 2003.

 **Alice Barbora Tumpach** is an Associate Professor in Mathematics (University Lille 1, France) and member of the Laboratoire Painlevé (Lille 1/CNRS UMR 8524), since 2007. She received a Ph.D degree in Mathematics in 2005 at the Ecole Polytechnique, Palaiseau, France. She spent two years at the Ecole Polytechnique Fédérale de Lausanne as a Post-Doc. Her research interests lie in the area of infinite-dimensional Geometry, Lie Groups and Functional Analysis. She is author and co- author of several publications in international journals (TPAMI, Communications in Mathematical Physics, Journal of Functional Analysis, Annales de l'Institut Fourier) in the above fields. She served as reviewer for many journals, including TPAMI, Mathematische Annalen, Journal of Mathematical Physics, Journal of Mathematical Analysis and Appplication, Journal of Differential Geometry, Annales de l'Institut Fourier...

About the Evolution of Soft Robots

186.837: Seminar in Computer Vision and Pattern Recognition
SS 2021

Andreas Wiesinger

Abstract

The evolution of artificial neural networks has inspired a plethora of methods and encodings for the evolution of artificial lifeforms like soft voxel-based robots over the past 20 years. Such encodings aim to reach complexity of phenotypes in natural scale or even higher, possibly opening pandora's box and unleashing creatures like shown in the Terminator movies. We review major milestones in the field of artificial evolution and their successors, including Neuroevolution of Augmenting Topologies (NEAT), Compositional pattern producing networks (CPPNs), HyperNEAT and newer encodings. Soft robots are particularly interesting in the context of those encodings as they increase the search space and complexity for the possible outcome and pose a tremendous challenge for many of the surveyed techniques. Novel encodings aim to solve perennial problems in this field, even though major scalability difficulties have been addressed in the past. We investigate whether a novel combination of two innovations can provide a solution to remaining problems.

1 Introduction

We intend to provide a short review for some novel approaches to evolve and encode artificial lifeforms *and* their controllers. It turns out that Neural Networks (NNs) provide elegant means to evolve complex creatures with an efficient, indirect encoding with high abstraction of natural (chemical) evolution processes. Such encodings can be used to automate the design-process of robots for various tasks like explained in the following subsections.

1.1 Motivation

There are many occasions where autonomous robots would be useful in unstructured environments[1]. However, robots are usually designed and pre-programmed by humans for specific structured environments, e.g. factories [2].

Not only the automatic design and autonomous behaviour are important in the robotics field, but also softness for gripping and human-robot interaction, e.g. medical use, endoscopic surgery and search and rescue [8, 16, 22, 31, 32]. Hod Lipson [22] and Tomoya Kimura [19] state in their papers that compatibility[2] is a critical aspect for the safety and comfort which users perceive during the interaction with robots. Softness[3] offers a great opportunity to do that. Lipson also mentions another purpose of robotics in general: To study the behavior and performance of humans and animals. To emulate natural systems, since many materials in nature are soft, is hence easier with soft robots than conventional ones. The latter is also the focus of Francesco Corucci et al. [10] in their study of intelligence and adaptive behavior in animals and plants.

These studies show that it is of interest to find methods for automating the design-process of (soft) robots. There are different approaches, most of which are based on Evolutionary Computation (EC) [8, 36, 39] (and for interested readers we want to refer to additional literature: [2, 7, 9, 10, 17, 19, 22, 23, 27, 28, 30, 31, 38, 40, 42]).

The motivation of this state-of-the-art review is to summarize commonly used methods like NEAT-based Genetic Algorithms (GAs) [28, 39] as well as some alternative and older approaches like Cellular Encodings (or Cell Chemistry Encodings) (CEs) [14, 40], Cell Chemistry (CC) [40, 41] or Artificial Embryogeny (AE) [6, 11, 18, 20, 40], as well as grammatical approaches [18, 21, 40], Genetic Regulatory Networks (GRNs) [2, 4–6, 13, 17] and meta-graphs [2, 35]. The main focus of this paper, however, will be on the latest NEAT-based encodings and a possible novel combination of two of them to address some of the main issues which are discussed in section 1.2.

1.2 Problem Specification

Several different issues arise when we try to evolve morphologies and controller configurations for specimen of robots for complex tasks in unstructured environments:

Auerbach and Bongard [2] as well as Rieffel [31] point out, that the brain, the body and the environment are the three main components which define the search space[4] and

[1] Unstructured environments: The term was used by Auerbach [2] and means in this context: The environments in which the robots will perform their tasks. In this case, *unstructured* means environments like nature, offices or homes, where neither the premises nor its furnishing can be foreseen by the developers of the robot: Places which vary for every task. In comparison to structured environments like factories, where the robot likely works in a specific, dedicated, planned location.

[2] Compatibility between users and robots, both, in terms of mental perception by users as well as physical interaction.

[3] Robots with soft parts in contrast to conventional robots do not only have stiff parts. Softness does not necessarily mean that a robot is completely soft, but rather consists of both, soft, elastic parts like inflatable rubber and stiff parts.

[4] The *search space* is the space in which the weights and the connections and (new) nodes (i.e. the topology) of a network is evolved by "searching" for weights and connections which result in the

evolution of a phenotype[5]. This for once brings up the requirement that the controller (the brain) and the morphology (the body) must usually be co-evolved, i.e. let them evolve simultaneously, because they are interdependent. Furthermore, for the special case of *soft* robots, which we mainly focus on in this paper, the *material* is a third component in addition to the body and the brain which needs to be co-evolved with the latter two [31]. Consequently, the parameter-space increases and leads to difficulties in the scalability of many if not most available encodings concerning the amount of voxels for a phenotype as well as the complexity of its behavior [8, 19]. Last but not least, Lipson [22], Cheney [8] and Medvet [24] as well as Rieffel [31] agree on the fact that the evolved robots are difficult to simulate realistically due to the issue that elastic, soft materials can and will always behave different in the real world than in a virtual, mathematical, approximated environment.

We want to focus solely on the scalability issue in this paper, therefore the rest of the problems shall be mentioned for the sake of completeness. However, the scalability issue has been investigated by many authors previously and can be found especially in NEAT- and CPPN-based methods [36, 38–40] (and for interested readers we want to refer to additional literature: [8, 12, 15, 19, 27, 28, 37, 43]). Because CPPN and NEAT are simply NNs, the issue from above translates to scalable evolution of NNs.

2 Background

In *artificial evolution*, for systems such as soft robots, the goal is to reach the level of complexity found in natural biological lifeforms, or even outperform nature in this regard [40]. For this sake, different types of encodings have been proposed in the past [28, 39, 40]. The major terms and concepts in this field are summarized below:

2.1 Encoding Types: Direct vs. Indirect Encodings

The two major classes of Topology and Weight Evolving Artificial Neural Networks (TWEANNs) are *direct encodings* and *indirect encodings*. The difference is that direct encodings are easier to implement but are always equally as complex as the evolved phenotype because the genotype is a one to one mapping and contains the exact information about every connection and every node in the phenotype.

Indirect encodings on the other hand aim to drastically lower the complexity of the genotype but still be able to produce very complex phenotypes. The genotype in indirect encodings does not map every part of the phenotype, but utilizes rules which implicitly lead to the final phenotype. Hence, sometimes also the terms *implicit* vs. *explicit* encodings are used [39, 40].

desired network outputs. The specimen we want to evolve, which should be able to fulfill specific tasks, is hence represented by the network outputs.

[5] *Phenotype*: The set of observable characteristics of a living organism, which is influenced by its genotype[6] and the environment. In the context of EC, the evolved weights and connections of a NN are also often referred to as phenotype [15, 40].

[6] *Genotype*: The genetic code which encodes (describes) the phenotype.

Some examples for direct encodings are Structured Genetic Algorithms (SGAs) which belong to Binary Encodings (BEs), Parallel Distributed Genetic Programming (PDGP), which uses Graph Encoding and GeNeralized Acquisition of Recurrent Links (GNARL), which also uses Graph Encoding, but does not implement *crossover* mechanics, which is discussed in Section 2.2. An example for direct encodings is NEAT by Stanley and Miikkulainen [39].

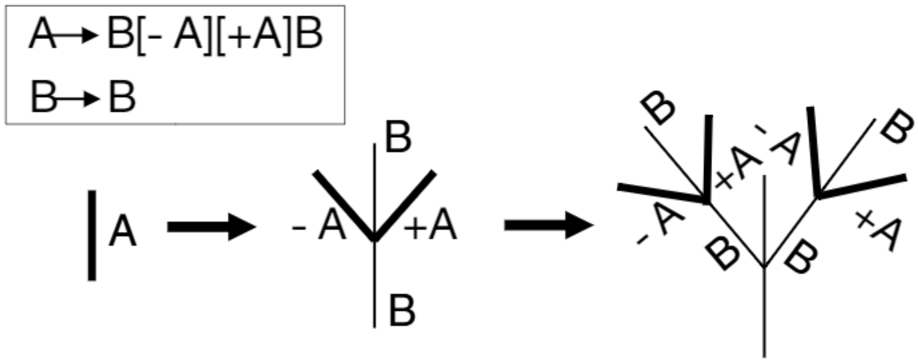


Figure 1: Grammatical approach example, taken from Stanley and Miikkulainen [40]. This illustration shows how one of the first indirect encodings works. The L-rewrite-system by Lindenmayer [21] models a tree-like morphology by simply defining a rule (in this case the two rewrite rules in the box) and a starting symbol. In this case, the symbol A is replaced with an expression defined by the rules in the box until there is no further replacement possible or a defined amount of generations is reached. $-$ and $+$ define relative angles for the next generation for a specific branch.

Almost 20 years later, many indirect encodings have been published, CPPNs being one of them, also by Stanley [36]. L-Systems by Lindenmayer [21], from which an example can be seen in Figure 1, were one of the first indirect encodings and are amongst the *grammatical approaches*. Also within the class of grammatical approaches are *grammar trees* [14]. Because the rules are effective on single cells, this system by Gruau is called Cellular Encoding (or Cell Chemistry Encoding).

2.2 Competing Conventions

The Permutations Problem [29, 39] or *Competing Conventions Problem* [25, 33, 39] has been a main problem in Neuroevolution (NE) until Stanley [39] proposed *historical markings* in NEAT. It is about the possible loss of critical information in a genome which can happen when two permutations of genomes are crossed over with each other: Crossover will likely lead to damaged offspring if a weight optimization problem in an NN has multiple possible solutions. Figure 2 shows an example for a possible resulting genome after crossover if the problem is not circumvented properly.

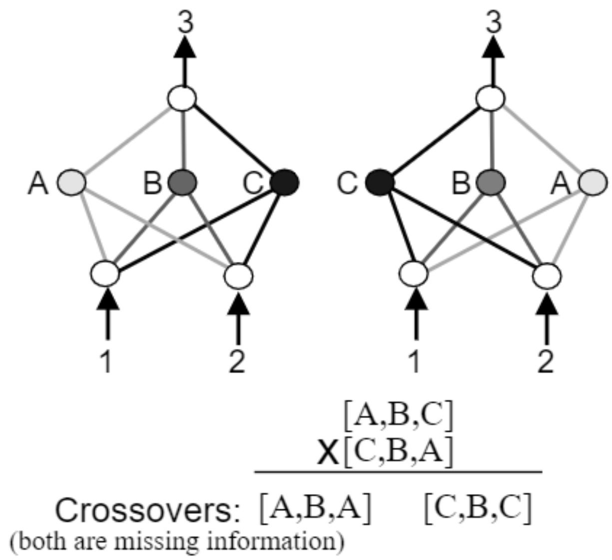


Figure 2: Competing conventions example, taken from Stanley [39]. Even though the order of the hidden nodes differs between the two networks, the final output of them is the same. A crossover would lead to either one solution which would miss a crucial component. Therefore Stanley defines them as being *incompatible* for crossover.

The problem is even more difficult when different topologies and lengths of genomes shall be crossed because TWEANNs can still produce the same (or similar) solutions in this case. In nature, during a process called *gene amplification*, new genes can also not just insert themselves at random positions in the genome [39]. A system where genes can only be crossed with alleles[7] of the same trait, called *homology*, is utilized by nature where homologous genes are lined up before they can be crossed. Stanley discovered that homology is given when the *historical origin* is shared across genes [39].

2.3 Niches: Protecting Innovation within Species

Adding new nodes and connections to TWEANNs brings both, innovation and problems: On the one hand this is the way how new features in a specimen are found in the search space, on the other hand every new node and connection adds a new nonlinearity or redundancy which decreases the fitness of the network. Therefore it is not sufficient to just evolve a phenotype by randomly varying the nodes and connections in the hope of new interesting features, because most of them will rather destroy already evolved features. A successful Evolutionary Algorithm (EA) needs to protect new features in order to be able to optimize the weights to get the best out of it. Stanley explains

[7] One of several possible variants of a gene is called an *allele*.

that such innovations could be protected by isolating different network structures into different *species*, therefore allowing them to optimize their structures within the same species, which is called *speciation* or *niching*. Having a compatibility function which tells the compatibility between networks of different topologies is needed to solve the speciation problem. NEAT implements speciation by *explicit fitness sharing* which groups individuals by their genetic similarity. The historical markings in NEAT do not only provide a solution to the competing conventions problem, but as a consequence also for the speciation problem [39].

2.4 Minimizing the Search Space

Another problem is the initial population and the complexity of the search during evolution. The initial topology was often chosen randomly, where minimal solutions are not targeted specifically. NEAT targets minimal solutions by starting out minimally and by intentionally letting the network grow only if the emerged structure benefits the outcome. NEAT therefore focuses on keeping the dimensionality[8] of the search space as low as possible throughout all generations and not just at the end of the final network. This dramatically increases the performance as well [39].

2.5 Developmental vs. Non-Developmental Encodings

In nature, development can be seen through the growth of a phenotype over time: Different genes can be activated (or deactivated) at different times (and locations), which controls the development of a phenotype and facilitates the reuse of genes. This is not about evolving phenotypes, but about the change of evolved phenotypes during their individual lifetimes. *Developmental encodings* [1, 3, 36, 40] aim to discover phenotypes with a complexity found in nature, by finding the right abstraction of the developmental process found in nature [36]. The reuse of genes allows efficient artificial development because the genotype does not need to be as complex as the phenotype.

Non-Developmental encodings on the other hand skip the process of development and only simulate the final artificial lifeforms such as if there is no temporal dimension. NEAT [39, 40] is one example for non-developmental encodings.

3 State of the Art

Papavasileiou et al. [28] already published a comprehensive, systematic state-of-the-art survey of NEAT-based methods. Our paper differs from theirs that their latest observed NEAT-based methods were published not later than in 2017, but within the last four

[8]The *dimensionality* of the search space is increased by adding more different possible components to it: Adding different materials of which a robot can consist increases the search space by the amount of materials being added. Adding the goal that a robot should also evolve a controller for its morphology increases the search space. Adding a node or connection at the network-level increases the search space as more weights need to be optimized. Last but not least, adding complexity to the task a robot should fulfill, increases the search space.

years, new techniques have emerged. Therefore, we will shortly summarize NEAT, CPPN and HyperNEAT and then some newer NEAT-based methods in contrast to the ones already covered by Papavasileiou et al. Furthermore, in contrast to their paper, we will also discuss a possible combination of some key features of the reviewed methods to hypothesize possible beneficial outcomes. The papers which are already covered by Papavasileiou et al. are not discussed here again, except for those on which our discussion will be based upon.

3.1 NEAT (2002)

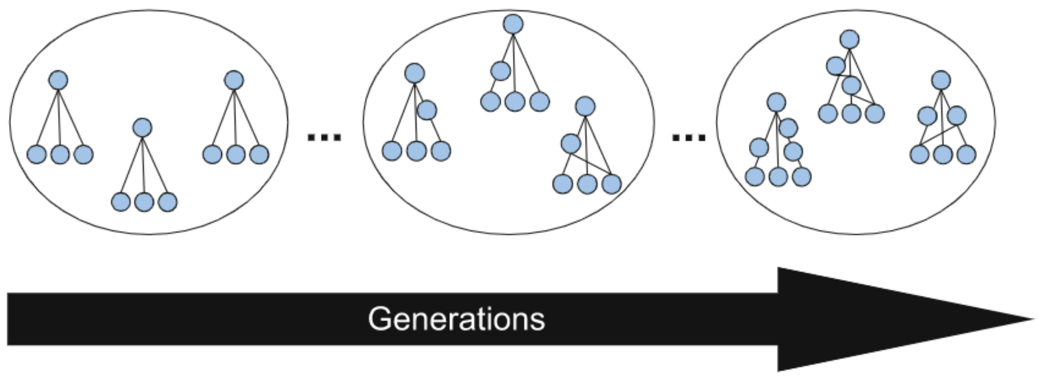


Figure 3: Ambrosio et al. [12] illustrate the process of incrementally complexifying the topology[9]of NNs which are evolved with NEAT after starting out minimally. Additional note: This image illustrates many connected graph-based networks at once, because NEAT evolves many networks independently to have different species and later allow to cross them to get new interesting features through cross-over.

NEAT is a direct encoding made to evolve NNs which aims to solve controlling and decision tasks [39]. The differences between NEAT and other NN-evolving methods are pointed out as follows: NEAT uses historical markings [39], where the generation-number is assigned to each gene of a generation, to keep track of the individually evolved genes (as described in Sections 2.2 and 2.3) which are used to

- determine *how* to combine *which* genes to produce offspring [39],
- allow crossover of different topologies without expensive analysis for the compatibility [39],
- classify individuals into different species [39] (Section 2.2), and therefore

[9]**Topology** was the term used by Ambrosio et al. [12] to describe the **connectivity** of a single graph-based neural network.

- protect structural innovations by speciating the population into niches in which individuals compete during weight-optimization instead of the whole population at once, because adding innovation (by adding new nodes to the topology) often introduces disadvantages at first (see Section 2.3) [39],
- start out with small and simple networks and increase their complexity incrementally as can be seen in Figure 3, without a size-limit for the topology, rather than evolving fixed-topology or random-topology networks (see Section 2.4) [12]

NEAT is a direct encoding on purpose, because Stanley and Miikkulainen wanted to avoid unpredictable biases in their search process, which was likely to happen in indirect encodings at the time of writing their paper due to insufficient knowledge about genetic and neural mechanisms.

3.2 CPPN (2007)

CPPN is an indirect encoding by Stanley [36] which is ultimately just an NN. The name compositional pattern producing network (CPPN) was chosen in order to avoid possible misconceptions about CPPNs, as they are invented for the sake of evolving structures and should not imply that genetic encodings or developing embryos are in effect the same as thinking brains [12, 36]. CPPNs utilize functions with specific characteristics as activation functions in the individual nodes, hence the name *"compositional pattern producing"*. The final output pattern is produced by composing these individual functions in a network together, which is also illustrated in the Figures 4 and 5b. Figure 5 shows how CPPN overcomes the complexity of natural genetic encodings through abstraction of local interaction and temporal (ontogenetic) development (see also Section 2.5): Only the final morphology is represented by the solution of an evolved CPPN. Including development in such a final morphology means that there is just an additional input parameter, namely time, in order to evolve patterns which are changing over time, therefore simulating development.

One can choose the activation functions freely, depending on the desired outcome, however, some core functions for a CPPN proved suitable for a range of goals [12]:

- Sigmoid for irregularities and non-linearities.
- Gaussian or absolute value for symmetry.
- Sine or cosine for repetition.

NEAT is commonly used to evolve CPPNs: NEAT uses a direct encoding, meaning its genotype will have the same size and complexity as the phenotype, but as it is used to evolve an indirect encoding, it will just need the size and complexity of the evolved CPPN and not of the pattern produced by this CPPN. The CPPN is also again the genotype for the phenotype it encodes, which should be taken care of to avoid possible misunderstanding which genotype and phenotype we are talking about.

Finally, some evolved patterns, structures and even soft robots, using CPPNs are shown in the Figures 6 and 7, to provide an overview of its possibilities.

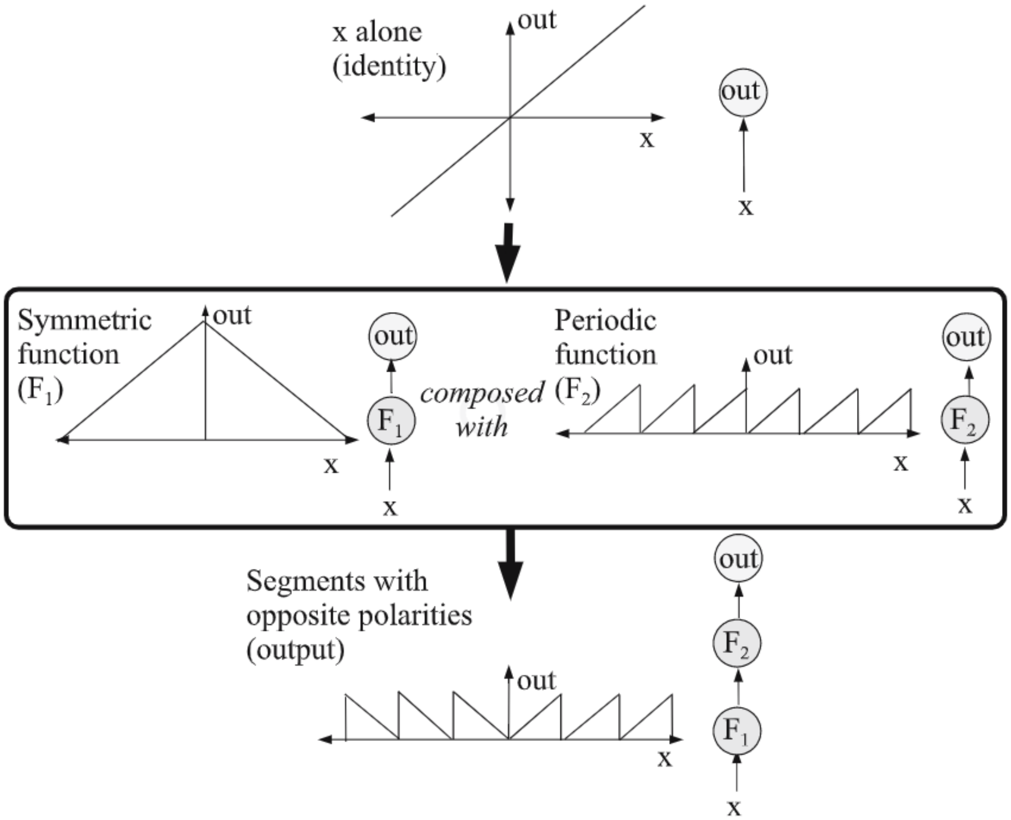


Figure 4: Stanley [36] illustrates the *composition of gradient functions* with this image: Composing an asymmetric gradient x with two functions F_1 and F_2, one being symmetric and the other periodic, results in a new pattern which contains the characteristics from each of them: The gradient from x alone, the symmetry from F_1 and the periodicity from F_2.

3.3 HyperNEAT (2009)

HyperNEAT (from *"hypercube-based NeuroEvolution of Augmenting Topologies"* [12,38]) maps geometric regularities onto a previously defined *substrate*[10]: That is, the CPPN outputs the weights for the connections in that substrate, while that same substrate provides its spatial node placements and connection locations as input to the CPPN. The CPPN is hence called *weight-generator*. A threshold for a minimum weight value determines whether a connection between two nodes in the NN (the substrate) exists at

[10]The *substrate* is the NN for which the connectivity pattern is evolved by the CPPN. This name is chosen to be able to verbally distinguish between the NN of the CPPN and the NN of the final phenotype [12].

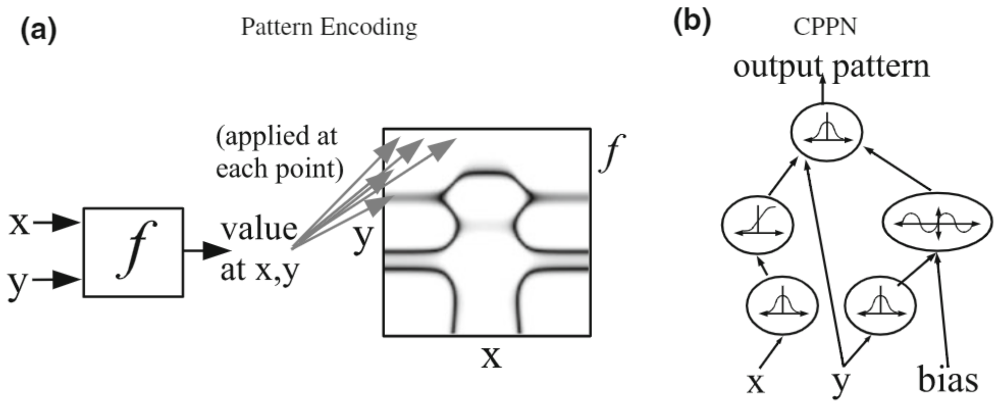


Figure 5: This illustration, taken from Ambrosio and Stanley [12], depicts the individual functions in the CPPN (**b**) which are composed and produce the final output pattern (**a**). The function f in **a** is the network in **b**.
a: The main concept of CPPNs can be seen in this illustration: To provide an abstraction for local interaction and temporal development by making the output of the CPPN solely dependent on the input parameters x and y (and any desired additional dimensions). The evolved pattern is simply a function $f(x, y, ...)$, hence, no interaction between a chosen point and its neighbours is needed to compute the pattern. The key characteristics described in Section 3.2 are the reason why this high abstraction still leads to structures looking similar to those found in nature.
b: The NN-nature of CPPN can be seen in this illustration clearly: The individual nodes have different activation functions which are of either kind of the few main characteristics for the abstraction which Stanley defined.

all. Consequently, the substrate starts out with a *complete graph*[11].

The main advantage of HyperNEAT is, that it is able to express NNs with millions of connections through CPPNs with only dozens [12]. Two more advantages are the scalability and the benefits from the link between geometry and connectivity patterns:

HyperNEAT is able to train the network in a low resolution, then up-scale the substrate to raise the resolution (i.e. add more nodes and connections), without destroying the connectivity-pattern and its already trained weights, because this pattern has a geometric, mathematical nature in a higher-dimensional cube, and the actual connections and nodes can be seen as a kind of "sampling" this pattern in a continuous, spatial space into a discrete and topological space [12, 38].

The latter advantage is that HyperNEAT can exploit the fact that sensor location and effector location correlate in the geometry of the problem domain and can create neural structures which reflect that (geometric) information [12, 38].

[11] A *complete graph* is a graph where every node is connected to all other nodes.

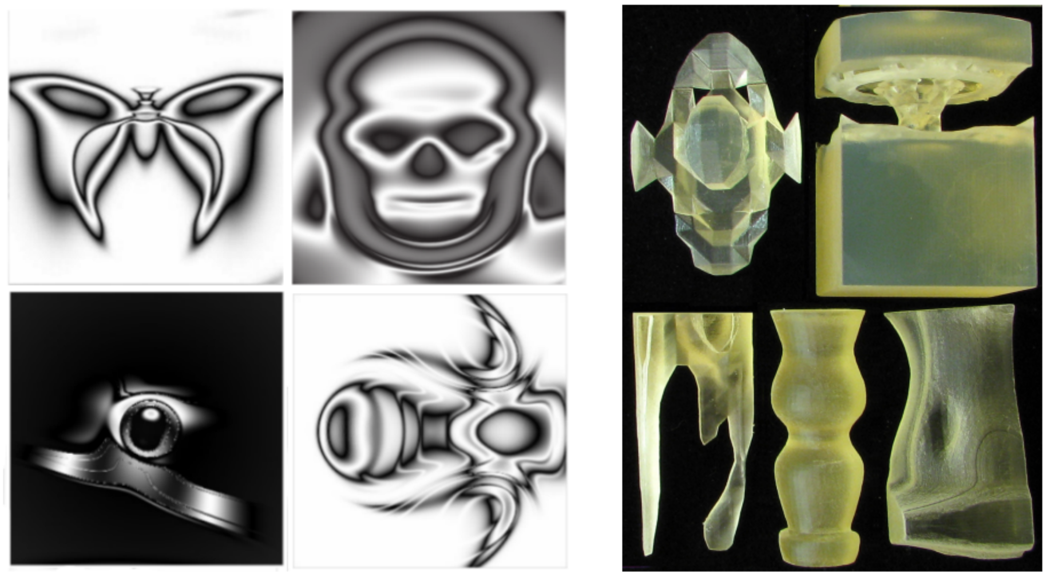


Figure 6: Cheney [8] illustrates 2D (left) and 3D (right) examples of evolved patterns using CPPN-NEAT.

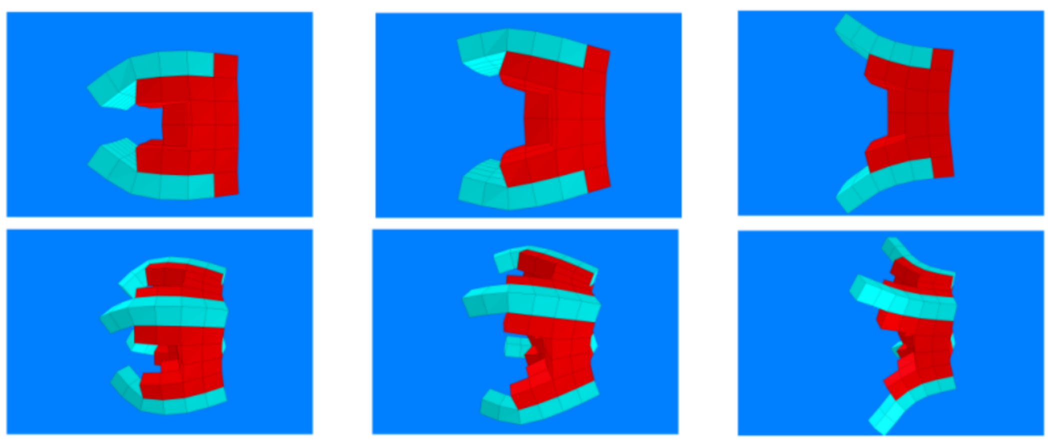


Figure 7: Corucci [9] evolved aquatic creatures (soft voxel-based robots) which look and behave like jellyfish with CPPN-NEAT.

3.4 NEAT-based derivatives

According to the categorization of Papavasileiou et al. [28], x-NEAT methods may be mapped to the following properties. We did not include all of them, as we do only consider the following as relevant for our aims:

- **Support multiple objectives**: The evolved robot should be able to fulfill multiple different tasks. Some of such tasks might be independent from each other (e.g. simply walking towards a static target vs. grabbing objects which are in physical range of the robot) while others might interleave or blend, e.g. keeping track of a moving target via sensory input and adapt the movements accordingly, or grabbing objects while walking. In technical terms, this results in a multiobjective optimization problem, where the evolution algorithm is given a domain where multiple (conflicting) objectives should be optimized simultaneously.

- **Support different node types**: The evolved network should support different activation functions per node in order to allow for a composition of familiar functions to allow complex shapes and patterns as a result as explained previously. CPPNs already support different node types by intention. Not all NEAT-based methods make use of this feature.

- **Modularity**: A modular network contains separately optimizable, independent functional structures which can perform sub-tasks. Our hypothesis is, that new, additional tasks can be added more easily to existing controllers which are already trained and optimized for one or more tasks, if the controller supports modularity. This would enable sequential training instead of (or in addition to) simultaneous training.

- **Support large topologies**: High dimensional input/output space as well as a high-dimensional search space are supported by methods with this property. The evolved networks might also consist of a large amount of nodes and connections. As we aim to evolve robots which support multiple objectives, the search space naturally becomes large. Soft voxel-based robots consist of many parts with different properties (soft parts, stiff parts, etc.), which further increase the search space. Therefore, this property is considered necessary for our goal.

Method	Multiple objectives	Different node types	Modularity	Large topologies
SUPG-HyperNEAT (2013) [26]		✓		✓
MM-NEAT (2016) [34]	✓		✓	
Our method	✓	✓	✓	✓

Table 1: The selected categorization-properties from Papavasileiou et al. [28], which are important for our proposed method and which property is possessed by which method.

Most of the methods covered by Papavasileiou et al. [28] support large topologies, which is an important property. However, most of the same methods do not possess each of the above properties, which would be desirable. We want to propose a possible combination of Modular Multiobjective (MM) NEAT [34] with Single-Unit Pattern Generator (SUPG) HyperNEAT [26] to get all of these properties in one method, as can be seen in Table 1.

3.4.1 SUPG-HyperNEAT (2013) [26]

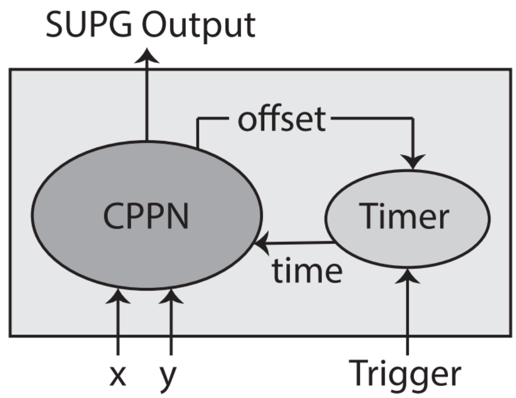


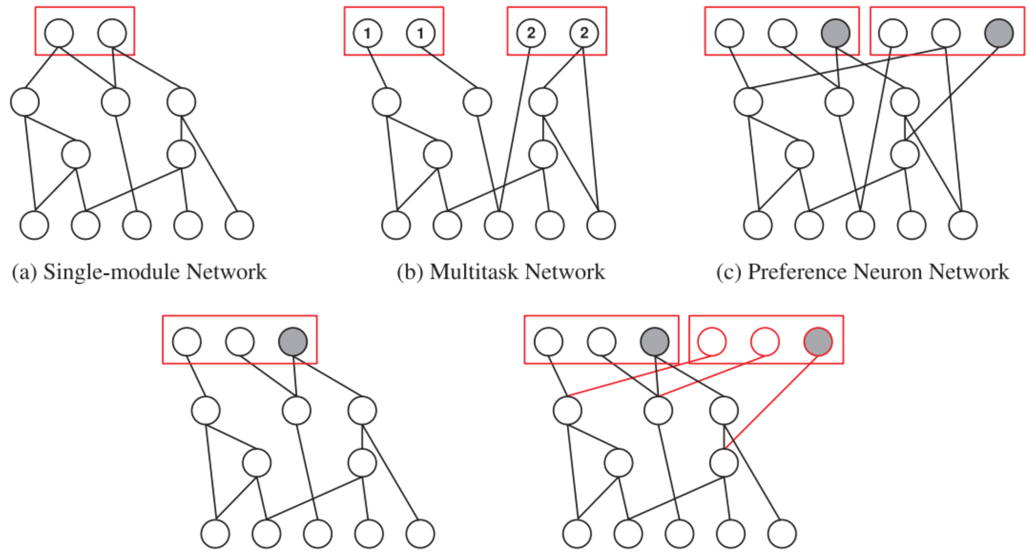
Figure 8: A single SUPG neuron. Each trigger resets the timer. The output of the SUPG depends on x, y and the internal time value. The offset is applied to the initial trigger to allow different SUPGs in the same network to start at different times [26].

Even though Single-Unit Pattern Generator (SUPG)-HyperNEAT [26] is intended for legged, quadruped robots, we hypothesize that the concept can be applied to voxel-based soft robots as well. The SUPG extends HyperNEAT with a new type of neuron which contains a timer to allow for time-dependent activation patterns. Even though the SUPG itself represents just a node in the final network, it is encoded by a CPPN. Figure 8 shows, how a single SUPG node consumes input coordinates and a temporal trigger. The internal timer's output (the time since the last trigger) is the time input of the internal CPPN. This allows to generate patterns which are not just across space, but also across time and therefore oscillatory systems can be evolved and encoded with this approach.

The CPPN inside a SUPG is not directly the SUPG itself, but it is *describing* the actual network's weights and connections depending on the input coordinates. Hence, HyperNEAT, because Morse et al. make use of this indirect encoding to efficiently describe a large SUPG with a small CPPN. They also add, that the depicted input coordinates x and y are for simplicity reasons compacting the usually used connection-coordinates for HyperNEAT, namely x_1, y_1, x_2 and y_2.

3.4.2 MM-NEAT (2016) [34]

Modular Multiobjective (MM)-NEAT [34] proposes *preference neurons* and the use of modules within a network. The preference neurons are used to decide which module is used for the actual behaviour of the agent at a time. The preference neuron with the highest output specifies the network's output. The other output neurons in each module are called *policy neurons.* They define the behaviour of the corresponding module.



(a) Single-module Network (b) Multitask Network (c) Preference Neuron Network

(d) Before Module Mutation (e) Module Mutation Duplicate

Figure 9: Modular networks. Policy neurons define the behaviour of an agent (empty circles inside the red rectangles). Inputs are at the bottom and outputs are in the red boxes. The grey circles are the preference neurons. (a) Neural network with one module. (b) Multitask network which needs human interaction to choose the active module. (c) Fixed network with preference neurons to select the active module. (d) Module mutation is enabled. The network has not yet mutated in order to evolve more modules, therefore only one module with an irrelevant preference neuron exists. (e) A second module is added through module mutation. Its policy neurons are linked to the same neurons as the module-duplication source. The preference neuron is linked to a random neuron with random weight. All preference neurons are now relevant. Each behavioural mode is represented by a different module, therefore multiple objectives are more easily learned by these networks [34].

Module mutation is used to add more modules during the evolution process. Each new module is initially created by duplicating an existing module. This concept is shown in Figure 9.

4 Proposal of novel Method: MM-SUPG-HyperNEAT

We propose a novel combination of previous NEAT-based methods [26, 34], and call it Modular Multiobjective Single-Unit Pattern Generator HyperNEAT (MM-SUPG-HyperNEAT), to obtain a method with all of the four benefits listed in Table 1. The purposes of the four properties are explained in Section 3.4. Our method is intended

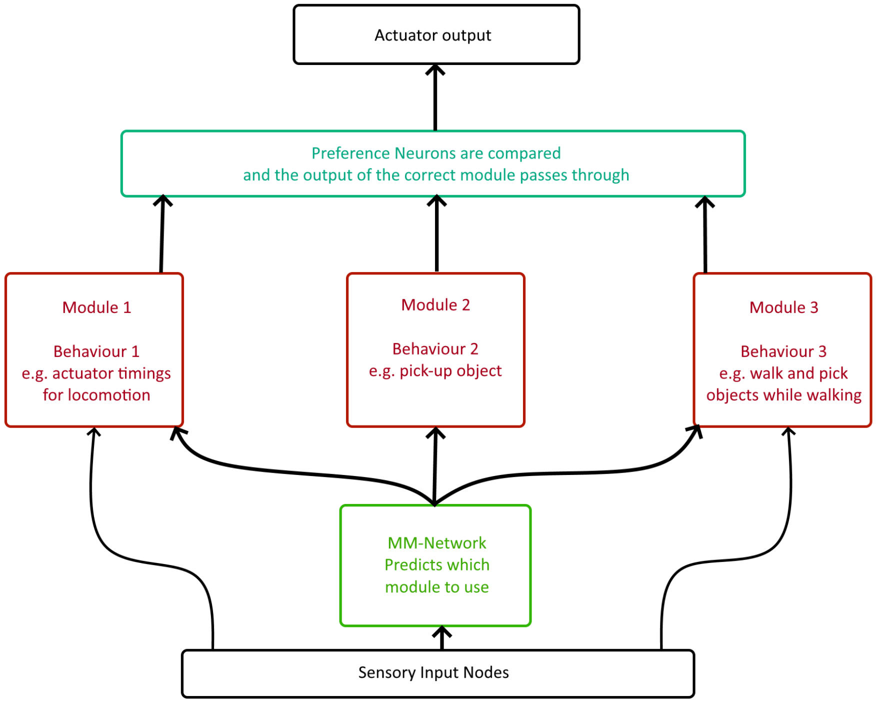


Figure 10: A rough overview of the composition of MM-NEAT and SUPGs in our method. The SUPGs are not directly visible in this graphic because they are contained in each module. The MM uses the sensory input to output a module preference proximity, which is used in the second-top layer to let the output of the correct module through to the robot's actuators. Each sensory input (or any other node) can also be used as input for the individual triggers of the SUPGs, hence the connection between the sensory inputs and the modules.

for the evolution of soft voxel-based robots, even though SUPG-HyperNEAT is intended for quadruped legged robots. Our method combines the two state-of-the-art methods as follows: Each output *module* of the MM-network consists of another evolved network which also contains SUPGs which are mapped to the actuators of the robot. The output of a module is the input to the actuators. Each module defines the behaviour of the robot, namely the individual movements of the individual voxels which can be actuated. In order to allow those actuators to produce stable movements across a relatively long duration (Morse et al. [26] tested their method with 5 minute time-frames, however, they claimed that their robots walked stably for up to three hours), it makes sense to have a SUPG for every actuator. Figures 10 and 11 show how this setup is composed.

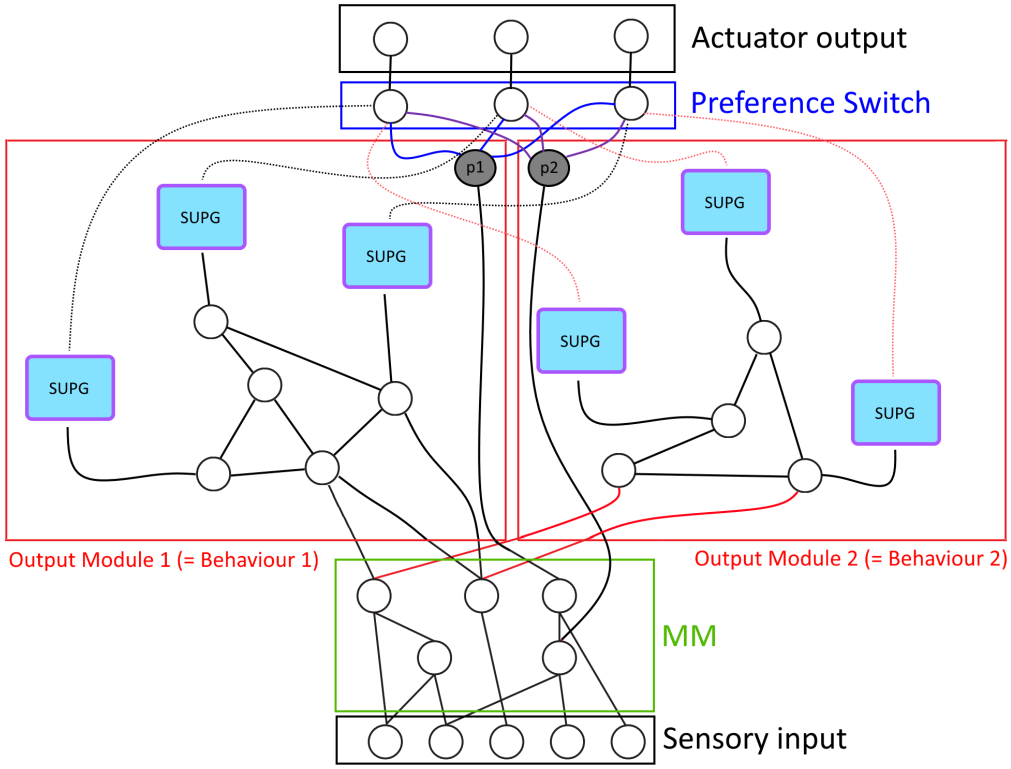


Figure 11: A more detailed sketch of the MM-SUPG-HyperNEAT method: Each of the colored boxes refer to the same boxes as in Figure 10. Here, we can also see the internal nodes of these networks and modules. As described in Section 3.4.2, the MM network will evolve new modules and each module will consist of policy neurons and one preference neuron. The preference neurons can be seen at the top of the modules (p1 and p2) in grey and are connected to our preference switch, which is just a possible implementation to automatically select the active module. That switch compares each preference neuron's value and passes only the output of the preferred module forward to the actuators. The actuators are in our case voxels which can be contracted depending on a signal. The SUPGs can be seen inside the modules: For simplicity reasons, the trigger inputs are not connected visually to other nodes, but they are in a real implementation. Each actuator is mapped (through the switch) to exactly one SUPG, however, as SUPGs are designed by intention with an internal offset, one and the same SUPG can be used for every actuator in a module, but with a different offset value.

The stability of movements means, that the robot should be able to perform a task over a long duration before it tips over or fails to continue. This stability is also largely affected by the environment. The SUPGs have the advantage that they can be reset by a trigger-signal and therefore possibly resolve any state of error which can occur during the tasks. The input for those trigger signals can come from sensors directly or from other nodes inside the network to possibly interpret the sensor-values before resetting the SUPGs.

The morphology of the robot has to be co-evolved separately, which is not included in the Figures 10 and 11, except for the evolved actuators which can be seen in those figures. This can be done with a simple CPPN or even with HyperNEAT. The two extensions MM-NEAT and SUPGs are not necessary for the evolution of the morphology alone. They are only proposed by us for the evolution of the controller of the robot.

5 Conclusion

We proposed a novel method as a combination of SUPG-HyperNEAT [26] and MM-NEAT [34] to allow a more efficient evolution process of soft voxel-based robots than previous methods, especially with the goal that such robots should support multiple objectives. This novel approach is called Modular Multiobjective Single-Unit Pattern Generator HyperNEAT (MM-SUPG-HyperNEAT). As the evaluation results of Schrum and Miikkulainen [34] proofed, the MM aspect solves the issue of supporting multiple objectives. The evaluation results of Morse et al. [26] proofed that the SUPG allows to evolve robots which can better adapt their locomotion to unknown and unpredictable environments and utilize the benefits of CPPNs and HyperNEAT to support different node types and large topologies. Whether such a combination of MM and SUPG performs as we hope is subject to future work with evaluation of our hypothesis.

References

[1] Peter J Angeline. Morphogenic evolutionary computations: Introduction, issues and example. In *Evolutionary Programming*, pages 387–401, 1995.

[2] Joshua E Auerbach and Josh C Bongard. Evolving complete robots with cppn-neat: the utility of recurrent connections. In *Proceedings of the 13th annual conference on Genetic and evolutionary computation*, pages 1475–1482, 2011.

[3] Peter J Bentley, Sanjeev Kumar, et al. Three ways to grow designs: A comparison of embryogenies for an evolutionary design problem. In *GECCO*, volume 99, pages 35–43, 1999.

[4] Josh Bongard. Evolving modular genetic regulatory networks. In *Proceedings of the 2002 Congress on Evolutionary Computation. CEC'02 (Cat. No. 02TH8600)*, volume 2, pages 1872–1877. IEEE, 2002.

[5] Josh C Bongard and Rolf Pfeifer. Evolving complete agents using artificial ontogeny. In *Morpho-functional Machines: The new species*, pages 237–258. Springer, 2003.

[6] Josh C Bongard, Rolf Pfeifer, et al. Repeated structure and dissociation of genotypic and phenotypic complexity in artificial ontogeny. In *Proceedings of the genetic and evolutionary computation conference*, pages 829–836, 2001.

[7] Nick Cheney, Josh Bongard, and Hod Lipson. Evolving soft robots in tight spaces. In *Proceedings of the 2015 annual conference on Genetic and Evolutionary Computation*, pages 935–942, 2015.

[8] Nick Cheney, Robert MacCurdy, Jeff Clune, and Hod Lipson. Unshackling evolution: evolving soft robots with multiple materials and a powerful generative encoding. *ACM SIGEVOlution*, 7(1):11–23, 2014.

[9] Francesco Corucci, Nick Cheney, Francesco Giorgio-Serchi, Josh Bongard, and Cecilia Laschi. Evolving soft locomotion in aquatic and terrestrial environments: effects of material properties and environmental transitions. *Soft robotics*, 5(4):475–495, 2018.

[10] Francesco Corucci, Nick Cheney, Sam Kriegman, Josh Bongard, and Cecilia Laschi. Evolutionary developmental soft robotics as a framework to study intelligence and adaptive behavior in animals and plants. *Frontiers in Robotics and AI*, 4:34, 2017.

[11] Frank Dellaert and Randall D Beer. A developmental model for the evolution of complete autonomous agents. In *Proceedings of the fourth international conference on simulation of adaptive behavior*, pages 393–401. MIT Press Cambridge, MA, 1996.

[12] David B D'Ambrosio, Jason Gauci, and Kenneth O Stanley. Hyperneat: The first five years. *Growing adaptive machines*, pages 159–185, 2014.

[13] Peter Eggenberger. Evolving morphologies of simulated 3d organisms based on differential gene expression. In *Proceedings of the fourth european conference on Artificial Life*, pages 205–213. Citeseer, 1997.

[14] Frederic Gruau, Darrell Whitley, and Larry Pyeatt. A comparison between cellular encoding and direct encoding for genetic neural networks. In *Proceedings of the 1st annual conference on genetic programming*, pages 81–89, 1996.

[15] David Ha, Andrew Dai, and Quoc V Le. Hypernetworks. *arXiv preprint arXiv:1609.09106*, 2016.

[16] Shigeo Hirose and Yoji Umetani. The development of soft gripper for the versatile robot hand. *Mechanism and machine theory*, 13(3):351–359, 1978.

[17] Nick Hockings and David Howard. New biological morphogenetic methods for evolutionary design of robot bodies. *Frontiers in Bioengineering and Biotechnology*, 8, 2020.

[18] Gregory S Hornby and Jordan B Pollack. Body-brain co-evolution using l-systems as a generative encoding. In *Proceedings of the 3rd Annual Conference on Genetic and Evolutionary Computation*, pages 868–875, 2001.

[19] Tomoya Kimura, Ziqiao Jin, Ryuma Niiyama, and Yasuo Kuniyoshi. Evolving soft robots to execute multiple tasks with combined-cppn-neat. In *2016 IEEE/SICE International Symposium on System Integration (SII)*, pages 409–414. IEEE, 2016.

[20] Maciej Komosiński and Adam Rotaru-Varga. Comparison of different genotype encodings for simulated three-dimensional agents. *Artificial Life*, 7(4):395–418, 2001.

[21] Aristid Lindenmayer. Mathematical models for cellular interactions in development i. filaments with one-sided inputs. *Journal of theoretical biology*, 18(3):280–299, 1968.

[22] Hod Lipson. Challenges and opportunities for design, simulation, and fabrication of soft robots. *Soft Robotics*, 1(1):21–27, 2014.

[23] Eric Medvet, Alberto Bartoli, Andrea De Lorenzo, and Giulio Fidel. Evolution of distributed neural controllers for voxel-based soft robots. In *Proceedings of the 2020 Genetic and Evolutionary Computation Conference*, pages 112–120, 2020.

[24] Eric Medvet, Alberto Bartoli, Andrea De Lorenzo, and Stefano Seriani. 2D-VSR-sim: A simulation tool for the optimization of 2D voxel-based soft robots. *SoftwareX*, 12:100573, 2020.

[25] David J Montana and Lawrence Davis. Training feedforward neural networks using genetic algorithms. In *IJCAI*, volume 89, pages 762–767, 1989.

[26] Gregory Morse, Sebastian Risi, Charles R Snyder, and Kenneth O Stanley. Single-unit pattern generators for quadruped locomotion. In *Proceedings of the 15th annual conference on Genetic and evolutionary computation*, pages 719–726, 2013.

[27] Stefano Nichele, Mathias Berild Ose, Sebastian Risi, and Gunnar Tufte. Ca-neat: evolved compositional pattern producing networks for cellular automata morphogenesis and replication. *IEEE Transactions on Cognitive and Developmental Systems*, 10(3):687–700, 2017.

[28] Evgenia Papavasileiou, Jan Cornelis, and Bart Jansen. A systematic literature review of the successors of "neuroevolution of augmenting topologies". *Evolutionary Computation*, 29(1):1–73, 2021.

[29] Nicholas J Radcliffe. Genetic set recombination and its application to neural network topology optimisation. *Neural Computing & Applications*, 1(1):67–90, 1993.

[30] Daniel Richards and Martyn Amos. Evolving morphologies with cppn-neat and a dynamic substrate. In *Artificial Life Conference Proceedings 14*, pages 255–262. MIT Press, 2014.

[31] John Rieffel, Davis Knox, Schuyler Smith, and Barry Trimmer. Growing and evolving soft robots. *Artificial life*, 20(1):143–162, 2014.

[32] Siddharth Sanan, J Moidel, and Christopher G Atkeson. A continuum approach to safe robots for physical human interaction. In *International Symposium on Quality of Life Technology*. Citeseer, 2011.

[33] J David Schaffer, Darrell Whitley, and Larry J Eshelman. Combinations of genetic algorithms and neural networks: A survey of the state of the art. In *[Proceedings] COGANN-92: International Workshop on Combinations of Genetic Algorithms and Neural Networks*, pages 1–37. IEEE, 1992.

[34] Jacob Schrum and Risto Miikkulainen. Solving multiple isolated, interleaved, and blended tasks through modular neuroevolution. *Evolutionary computation*, 24(3):459–490, 2016.

[35] Karl Sims. Evolving 3d morphology and behavior by competition. *Artificial life*, 1(4):353–372, 1994.

[36] Kenneth O Stanley. Compositional pattern producing networks: A novel abstraction of development. *Genetic programming and evolvable machines*, 8(2):131–162, 2007.

[37] Kenneth O Stanley, Jeff Clune, Joel Lehman, and Risto Miikkulainen. Designing neural networks through neuroevolution. *Nature Machine Intelligence*, 1(1):24–35, 2019.

[38] Kenneth O Stanley, David B D'Ambrosio, and Jason Gauci. A hypercube-based encoding for evolving large-scale neural networks. *Artificial life*, 15(2):185–212, 2009.

[39] Kenneth O Stanley and Risto Miikkulainen. Evolving neural networks through augmenting topologies. *Evolutionary computation*, 10(2):99–127, 2002.

[40] Kenneth O Stanley and Risto Miikkulainen. A taxonomy for artificial embryogeny. *Artificial Life*, 9(2):93–130, 2003.

[41] Alan Mathison Turing. The chemical basis of morphogenesis. *Bulletin of mathematical biology*, 52(1):153–197, 1990.

[42] Joshuah Wolper and George Abraham. Evolving novel cellular automaton seeds using compositional pattern producing networks (cppn). In *Proceedings of the 2016 on Genetic and Evolutionary Computation Conference Companion*, pages 27–28, 2016.

[43] Haoling Zhang, Chao-Han Huck Yang, Hector Zenil, Narsis A Kiani, Yue Shen, and Jesper N Tegner. Evolving neural networks through a reverse encoding tree. In *2020 IEEE Congress on Evolutionary Computation (CEC)*, pages 1–10. IEEE, 2020.