Kernel Representations for Evolving Continuous Functions

Tobias Glasmachers, Jan Koutník, and Jürgen Schmidhuber

IDSIA
University of Lugano & SUPSI
Galleria 2, Manno-Lugano, Switzerland
{tobias,hkou,juergen}@idsia.ch

Abstract. To parameterize continuous functions for evolutionary learning, we use kernel expansions in nested sequences of function spaces of growing complexity. This approach is particularly powerful when dealing with non-convex constraints and discontinuous objective functions.
Kernel methods offer a number of beneficial properties for parameterizing continuous functions, such as smoothness and locality, which make them attractive as a basis for mutation operators. Beyond such practical considerations, kernel methods make heavy use of inner products in function space and offer a well established regularization framework. We show how evolutionary computation can profit from these properties.
Searching function spaces of iteratively increasing complexity allows the solution to evolve from a simple first guess to a complex and highly refined function. At transition points where the evolution strategy is confronted with the next level of functional complexity, the kernel framework can be used to project the search distribution into the extended search space. The feasibility of the method is demonstrated on challenging trajectory planning problems where redundant robots have to avoid obstacles.

Acknowledgment This work was funded through the 7th Framework Programme of the EU under grant number 231576 (STIFF project) and SNF grant 200020-125038/1.

1 Introduction

The problem of learning continuous functions when dealing with discontinuous objective functions or non-trivial task constraints is tackled in this paper. This is a very general learning problem. For example, it arises naturally when dealing with robotic plants, typically controlled in joint angle space (configuration space), under constraints formulated in Cartesian task space. In this setup, the problem of planning a trajectory towards a target position while avoiding obstacles is considered.

Collision-free movement of robots from an initial position to a goal state in an environment with obstacles is one of the challenging problems in robotics. It

can be decomposed into two parts; kinematics of the movement, and control. In situations where a strict separation of these two movement aspects is feasible (when movements are relatively slow), most planning problems involving obstacle avoidance are due to kinematics only, while the problem of trajectory following can be decoupled. When planning a trajectory in a typical robotic application, joint and link positions need to be expressed in joint angle (or translations) coordinates. From a dynamics point of view, these positions are functions of joint torques (or forces) applied to the joints by a controller. In contrast—when decoupling planning and control—the plan can be computed first in a much simpler kinematic model, and the job of the controller is reduced to following the planned trajectory as closely as possible. This allows to solve the inverse kinematics and inverse dynamics problems separately.

In many approaches the two problems are merged into one, where no explicit motion plan is generated first and the dynamic control is executed based on the inputs from the environment and the goal [23, 14, 12, 9, 19, 32]. Several authors focus just on planning [1] and leave the control problem to well established algorithms for optimal control [21, 22]. The present study is in line with this approach.

Solving inverse kinematics optimally becomes intractable when the environment contains obstacles—the complexity of the problem is exponential in the configuration space degrees of freedom. Both problems get even more complex when the environment changes over time (e.g., there are moving targets or obstacles). In such cases, rather than solving the inverse problems analytically, various heuristic approaches are being used [5, 7], such as roadmaps [2, 17], cell decomposition [20], potential fields [18], or other dynamical systems approaches [16]. Hayashi [15] presents a method in which the joint angles are represented by a continuous function of time and the joint index, generating smooth movement plans. In such approaches overfitting is a potential issue and low solution complexity becomes an important goal.

Here, kernel expansions are used to represent the trajectory explicitly as a continuous function of time. Kernel-based learning algorithms [4, 26, 30] have found widespread use in machine learning, especially inspired by the support vector machine algorithm [6, 29]. The general methodology has been extended from binary classification to a wide variety of learning problems, ranging from supervised and unsupervised learning to elaborate statistical tests. Kernel methods offer easy-to-handle representations of continuous functions. At the same time, regularization is omnipresent in kernel approaches with the principal role of overfitting avoidance. These methods typically come with convex loss functions. Thus, efficient training schemes are available, and the uniqueness of the solution allows for a thorough mathematical analysis.

In the present study we leave this solid ground and explore learning of kernel representations under less standard conditions, particularly in non-convex learning problems with discontinuous objective functions, where most established training algorithms break down. The problem is resolved by employing evolutionary algorithms, a class of methods known to handle such challenges well.

When dealing with kernel methods, evolution strategies are of special interest, because they are designed for search and optimization in real-valued parameter spaces. State-of-the-art evolutionary strategies for black box optimization [24, 3, 8, 13, 31] provide robust and scalable techniques for pursuing global optima.

Modern, flexible, and easy to implement *natural evolution strategies* [31, 28, 27, 11, 10, 25] are used as search algorithms for kernel representations of continuous functions of a single variable (time). The efficiency is demonstrated on the problem of finding kinematic plan representations for redundant robot arms movement. Such arms are controlled in joint-angle space, which has a highly non-trivial, non-linear relationship to the three-space configuration of the different limbs and the end-effector.

Our method can be summarized as follows (refer to figure 1 for an illustration of the overall process): The algorithm iterates over a sequence of finite-dimensional, nested, kernel-based function spaces that—in the limit—exhaust the space of continuous functions. An evolution strategy is used to search the current sub-space for a trajectory solving a planning task at hand. For this purpose an individual consists of coefficients defining multiple continuous functions, one per degree of freedom, representing the joint angles progression over the (pre-specified) time course. This trajectory is transformed into Euclidean three-space by applying simple forward kinematics. The evolutionary search is subject to non-linear, non-convex constraints and a (potentially) discontinuous fitness function, as follows: A simulation of the trajectory in a forward kinematics model is run for each individual. The position of the robot as well as collisions of the robot with itself and with obstacles are recorded. Finally, (violations of) the constraints and the fitness value are computed from this data. For example, the fitness function for a reaching task can be the distance of the gripper position at the end of the movement from a target location plus a regularization term (the squared norm of the trajectory in a function Hilbert space). A collision indicates a constraint violation, and thus an infeasible individual.

This paper is organized as follows. The next section provides a short introduction to evolution strategies, with an emphasis on a modern class of algorithms called natural evolution strategies. The use of kernel methods in machine learning is discussed in section 3. A new method for kernel-based evolutionary learning of trajectories is introduced in section 4. The method comprises an incremental search scheme based on increasing the kernel representation complexity. It uses a slick algorithm that gradually extends the covariance matrix of the search distribution. The method generalizes to evolutionary learning of any continuous function. Experiments with a redundant arm in Euclidean two-space and a human-like 7 DOF arm in Euclidean three-space are described in section 5. The paper concludes with section 7.

2 Evolution Strategies

Evolution Strategies (ES) are evolutionary algorithms specialized for search and optimization in real vector spaces. There exists a wide variety of ES in the liter-

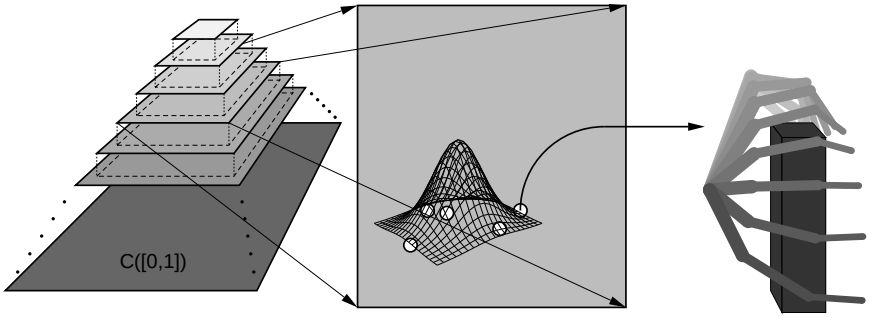


Fig. 1. Illustration of the methodology introduced in this paper. A nested sequence of kernel-based function spaces of growing complexity (left) is searched successively with an evolution strategy (middle). Each individual encodes a continuous trajectory for the robot arm (right). The fitness function reflects how well the candidate trajectory suits the task, while the constraints are evaluated by means of collision detection.

ature, refer to [24, 3, 8] for a comprehensive introduction. In canonical form, ES generate their offspring by Gaussian mutations, but other parametric families of search distributions have also been investigated. The different strategies vary widely w.r.t. their selection and recombination schemes. In population-based variants (μ, λ) selection is common, and recombination is often performed globally among all surviving individuals, such as in CMA-ES [13]. However, elitism in the form of hill-climbers and steady-state selection are common as well.

Typical ES are well suited to follow a 'global trend' of the fitness function $f : \mathbb{R}^d \to \mathbb{R}$. They identify (local) optima efficiently and with arbitrarily high precision. This requires a continuous adaptation of the search strategy parameters to the local characteristics of the problem at hand. The most important parameters to adapt are center and scale of the search distribution, while adaptation of the full covariance matrix has become state-of-the-art [13, 31]. Strategy adaptation, making mutations that have proven advantageous more probable in the future, is performed either actively or passively by means of mutation and selection.

Important aspects of evolution strategies are self-adaptation of the search strategy and invariance under certain transformations of the search space and the fitness function. Most evolution strategies are (up to initialization) invariant under translation and scaling, but often also rotations, or even all affine linear transformations. In addition, rank-based selection makes modern ES invariant under monotone fitness transformations.

Natural Evolution Strategies (NES) [31, 28, 27, 11, 10, 25] have been derived as population-based variants and as hill-climbers from the simple and powerful principle to adapt the search distribution in order to optimize (here, minimize) expected fitness by means of natural gradient descent. This general paradigm can be applied to all kinds of search distributions. Gaussians with different subsets of

adaptive parameters have been treated in the literature, such as adaptation of the full covariance matrix [31, 28, 27, 11, 10] and diagonal covariance matrices [25].

In each generation the population-based NES algorithm samples $\lambda \in \mathbb{N}$ individuals $\mathbf{z}_k \sim \mathcal{N}(\mathbf{z} \,|\, \theta)$, $k \in \{1, \ldots, \lambda\}$, i.i.d. from its Gaussian search distribution, which is parametrized by $\theta = (\boldsymbol{\mu}, \boldsymbol{\Sigma})$, with the goal to minimize a fitness function $f : \mathbb{R}^n \to \mathbb{R}$. Let $p(\mathbf{z} \,|\, \theta)$ denote the density of the Gaussian with parameters θ. Then, the expected fitness under the search distribution is

$$J(\theta) = \mathbb{E}_\theta[f(\mathbf{z})] = \int f(\mathbf{z})\; p(\mathbf{z} \,|\, \theta)\; d\mathbf{z} \;\; .$$

The gradient w.r.t. the parameters can be rewritten as

$$\begin{aligned} \nabla_\theta J(\theta) =& \nabla_\theta \int f(\mathbf{z})\; p(\mathbf{z} \,|\, \theta)\; d\mathbf{z} \\ =& \mathbb{E}_\theta \left[f(\mathbf{z})\; \nabla_\theta \log \left(p(\mathbf{z} \,|\, \theta) \right) \right] \;\; , \end{aligned}$$

(see [31] for the full derivation) from which we obtain the Monte Carlo estimate

$$\nabla_\theta J(\theta) \approx \frac{1}{\lambda} \sum_{k=1}^{\lambda} f(\mathbf{z}_k)\; \nabla_\theta \log \left(p(\mathbf{z}_k \,|\, \theta) \right)$$

of the search gradient. The key step then consists in replacing this gradient, pointing into the direction of (locally) steepest descent w.r.t. the given parametrization, by the natural gradient

$$\widetilde{\nabla}_\theta J = \mathbf{F}^{-1} \nabla_\theta J(\theta) \;\; ,$$

where $\mathbf{F} = \mathbb{E}\left[\nabla_\theta \log \left(p\left(\mathbf{z}|\theta\right) \right) \nabla_\theta \log \left(p\left(\mathbf{z}|\theta\right) \right)^\top \right]$ is the Fisher information matrix; leading to a straightforward scheme of natural gradient descent for iteratively updating the search distribution

$$\theta \leftarrow \theta - \eta \widetilde{\nabla}_\theta J = \theta - \eta \mathbf{F}^{-1} \nabla_\theta J(\theta) \;\; ,$$

with learning rate parameter η. The sequence of 1) sampling an offspring population, 2) computing the corresponding Monte Carlo estimate of the fitness gradient, 3) transforming it into the natural gradient, and 4) updating the search distribution, constitutes one generation of NES.

Fitness shaping [31] is used to normalize the fitness values by shaping them into rank-based utility values $u_k \in \mathbb{R}$, $k \in \{1, \ldots, \lambda\}$. For this purpose the individuals are ordered by fitness, with $\mathbf{z}_{1:\lambda}$ denoting the best and $\mathbf{z}_{\lambda:\lambda}$ denoting the worst offspring.

Then the "fitness-shaped" gradient

$$\nabla_\theta J = \sum_{k=1}^{\lambda} u_k \cdot \nabla_{(\theta)} \log \left(p(\mathbf{z}_{k:\lambda} \,|\, \theta) \right)$$

is used to update the parameters of the search distribution. Typically, the utility values are either non-negative numbers that sum to one, or are shifted to zero mean. The most important role of rank-based fitness shaping is to render the algorithm invariant under monotonic (rank preserving) transformations of the fitness values. The standard fitness shaping function is

$$u_k = \frac{\max\left(0, \log(\frac{\lambda}{2}+1) - \log(i)\right)}{\sum_{j=1}^{n} \max\left(0, \log(\frac{\lambda}{2}+1) - \log(j)\right)} - \frac{1}{\lambda} \ .$$

Algorithm 1: The xNES Algorithm

Input: $d \in \mathbb{N}$, $F : \mathbb{R}^d \to \mathbb{R}$, $\mu \in \mathbb{R}^d$, $\sigma > 0$, $\mathbf{B} \in \mathbb{R}^{d \times d}$ with $\det(\mathbf{B}) = 1$
while *stopping condition not met* **do**
  for $i \in \{1, \ldots, \lambda\}$ **do**
    $\mathbf{s}_i \leftarrow \mathcal{N}(\mathbf{0}, \mathbb{I})$
    $\mathbf{z}_i \leftarrow \boldsymbol{\mu} + \sigma \mathbf{B} \cdot \mathbf{s}_i$
  end
  sort $\{(\mathbf{s}_i, \mathbf{z}_i)\}$ with respect to $F(\mathbf{z}_i)$
  $\mathbf{g}_{\boldsymbol{\delta}} \leftarrow \sum_{i=1}^{n} u_i \cdot \mathbf{s}_i$
  $\mathbf{G}_{\mathbf{M}} \leftarrow \sum_{i=1}^{n} u_i \cdot (\mathbf{s}_i \mathbf{s}_i^T - \mathbb{I})$
  $g_\sigma \leftarrow \mathrm{tr}(\mathbf{G}_{\mathbf{M}})/d$
  $\mathbf{G}_{\mathbf{B}} \leftarrow \mathbf{G}_{\mathbf{M}} - g_\sigma \cdot \mathbb{I}$
  $\boldsymbol{\mu} \leftarrow \boldsymbol{\mu} + \eta_{\boldsymbol{\mu}} \cdot \sigma \mathbf{B} \cdot \mathbf{g}_{\boldsymbol{\delta}}$
  $\sigma \leftarrow \sigma \cdot \exp(\eta_\sigma / 2 \cdot g_\sigma)$
  $\mathbf{B} \leftarrow \mathbf{B} \cdot \exp(\eta_B / 2 \cdot \mathbf{G}_{\mathbf{B}})$
end

An efficient scheme for Gaussian distributions with fully adaptive covariance matrix called xNES, see algorithm 1, has been derived in [11]. The algorithm maintains a Gaussian search distribution $\mathcal{N}(\boldsymbol{\mu}, \sigma^2 \mathbf{B}\mathbf{B}^T)$, with center $\boldsymbol{\mu} \in \mathbb{R}^n$ global step size $\sigma > 0$, and shape matrix $\mathbf{B}$. The determinant of the transformation $\mathbf{B}$ is kept constantly at one, such that the complete scale information is bundled in the step size σ. The parameter vector is then composed of $\boldsymbol{\mu}$, σ, and $\mathbf{B}$. The decisive technique to turn the natural gradient update into a computationally tractable algorithm is to perform all updates in local coordinates. At the same time the matrix exponential is used to encode the positive definite symmetric covariance matrix in an unconstrained way: a vector $\boldsymbol{\delta}$ and a symmetric matrix $\mathbf{M}$ define local (exponential) coordinates around the current search distribution $(\boldsymbol{\mu}, \sigma, \mathbf{B})$, given by

$$(\boldsymbol{\delta}, \mathbf{M}) \mapsto (\boldsymbol{\mu} + \sigma \mathbf{B} \boldsymbol{\delta}, \sigma^2 \mathbf{B} \exp(\mathbf{M}) \mathbf{B}^T) \ .$$

It turns out that in these coordinates the Fisher matrix is the identity matrix, which saves its computation (or estimation), as well as its inversion, and the plain (Euclidean) gradient in these coordinates coincides with the natural gradient. It

remains to compute $\nabla_\theta \log\left(p(\mathbf{z}\,|\,\theta)\right)$ in these coordinates for the current search distribution, which is encoded by $\boldsymbol{\delta} = 0$ and $\mathbf{M} = 0$. We obtain the surprisingly simple formulas $\nabla_{\boldsymbol{\delta}} \log(p(\mathbf{z}|\boldsymbol{\delta}, \mathbf{M})) = \mathbf{s}$ and $\nabla_{\mathbf{M}} \log(p(\mathbf{z}|\boldsymbol{\delta}, \mathbf{M})) = \frac{1}{2}(\mathbf{s}\mathbf{s}^T - \mathbb{I})$, where $\mathbf{s}$ is the standard normally distributed vector corresponding to $\mathbf{z}$ (see algorithm 1). The covariance term is split into the trace of the second expression, corresponding to the scale variable σ, and its orthogonal complement, corresponding to the shape variable $\mathbf{B}$ (refer to [11] for details). Putting everything together results in algorithm 1.

The corresponding hill-climber variant (1+1)-xNES primarily differs in its step size adaptation rule [10]. Success-based utility values (which are rank-based in the sense of (1+1) selection) are used to implement a success-based self-adaptation rule, resulting in a behavior close to Rechenberg's 1/5-rule [24].

Both the population-based xNES and the hill-climber (1+1)-xNES come with good default settings for population size and learning rates, depending only on the search space dimension. In this sense, they are completely parameter free and applicable to black box problems.

3 Kernel-based Machine Learning

A kernel-based learning algorithm operates on an architecture that is linear in its parameters: Let $\{f_1, \ldots, f_n\}$ be a fixed set of basis functions, then $f = \sum_{i=1}^{n} \alpha_i \cdot f_n$ is linear in the coefficients $\boldsymbol{\alpha} \in \mathbb{R}^n$, while suitably chosen basis functions allow to add the necessary level of complexity to the model.

Typically, classic linear algorithms like linear regression or principal component analysis can be extended to this type of linear function architecture. Besides standard vector space operations, most such algorithms require only an inner product. The resulting learning problem is often convex in $\boldsymbol{\alpha}$, allowing for its efficient solution even in high-dimensional cases. All Hilbert space operations on f: addition, scalar multiplication, and inner products, can be expressed in terms of linear combinations of inner products of the basis functions f_i. Thus, only a finite number of "basis" inner products needs to be known, which are collected in the Gram matrix $\mathbf{K} \in \mathbb{R}^{n \times n}$ with entries $K_{ij} = \langle f_i, f_j \rangle$. The resulting learning algorithm is then formulated for the coefficients $\boldsymbol{\alpha}$, and it depends on the basis functions only in terms of the positive semi-definite "kernel" Gram matrix $\mathbf{K}$.

The basis functions f_i usually originate from a kernel function $k : X \times X \to \mathbb{R}$ given 'training' data $\{x_1, \ldots, x_n\}$, by defining $f_i(\cdot) = k(\cdot, x_i)$. For an input space X and a positive definite kernel function $k : X \times X \to \mathbb{R}$, Mercer's theorem ensures the existence of an only implicitly defined feature (Hilbert) space $\mathcal{H}$ and a feature map $\phi : X \to \mathcal{H}$, such that the kernel $k(x, x') = \langle \phi(x), \phi(x') \rangle$ computes the inner product of the features. Alternatively one can start from an embedding into a Hilbert space with given inner product and define the kernel function as $k(x, x') := \langle \phi(x), \phi(x') \rangle$. In this study we are interested in learning functions, and thus an only implicitly defined (function) feature space is not helpful. Thus,

we will rely on the latter approach starting from a given set of basis functions with explicitly defined inner product thereon.

Kernel methods are typically non-parametric, since the set of basis functions in use scales naturally with the data at hand. In learning, this high flexibility comes at the danger of easily overfitting the training data. The standard counter measure is regularization, which is most commonly achieved by augmenting the objective function of the learning problem with a so-called complexity penalty. The penalty is introduced by minimizing the squared norm $\|f\|^2$ of the function f in the Hilbert space. For a fixed set of basis functions with kernel matrix $\mathbf{K}$ and a function f with coefficient vector $\boldsymbol{\alpha}$ the squared norm can be written as $\|f\|^2 = \boldsymbol{\alpha}^T \mathbf{K} \boldsymbol{\alpha}$. This relation demonstrates also how computations in the infinite dimensional function Hilbert space can be expressed by finite computations based on pairwise inner products of basis functions.

4 Learning Continuous Functions

Consider learning continuous functions $f : [0,1] \to \mathbb{R}$, $f \in C([0,1])$ based on minimizing a cost or fitness function $F : C([0,1]) \to \mathbb{R}$. The space $C([0,1])$ of continuous functions on the unit interval is an infinite dimensional vector space. This space is often augmented with the inner product

$$\langle f, g \rangle = \int_0^1 f(t)\, g(t)\, \mathrm{d}\, t \ . \tag{1}$$

In most circumstances the parameter t is interpreted as time. Our method immediately generalizes from the unit interval to $\mathbb{R}$ or even to $\mathbb{R}^n$ or subspaces thereof.[1] In fact, it is one of the strengths of kernel methods that the very same proceeding can be generalized to any domain X. Our problem also requires to evolve functions $f : [0,1] \to \mathbb{R}^d$ with multiple components, which amounts to evolving d real-valued functions $f^{(1)}, \ldots, f^{(d)}$ in parallel with the fitness function being defined on $C([0,1])^d$.

4.1 Indirect Kernel Encoding

Why should the function be represented as a kernel expansion? In contrast to many other evolutionary search and learning tasks there is no direct encoding of the problem available, since (1) the search space is infinite dimensional and would thus require an infinitely long chromosome, and (2) even if this was possible, continuity of the solution would be hard to maintain in such a representation. This forces the use of an indirect encoding. Many such encodings are well established in the machine learning literature, such as feed-forward neural networks and kernel expansions, which are both known as universal function approximators. Other simple choices are polynomials or Fourier expansions.

[1] Care has to be taken that the function space is restricted to L^2 functions, that is, functions of finite norm.

However, using a chromosome of fixed finite length to encode elements of an infinite dimensional function space is conceptually unsatisfactory, since the choice of the subspace remains necessarily arbitrary. The issue is resolved by defining a nested sequence of growing subspaces, which encompass the full search space. For this purpose, let $\{f_1, f_2, \dots\}$ be a basis of a dense subspace of the function space $C([0,1])$, and let $(n_\ell)_{\ell\in\mathbb{N}}$ be a sequence of natural numbers tending to infinity. Then

$$\mathcal{F}_\ell = \operatorname{span}\left(f_1, \dots, f_{n_\ell}\right) \subset C([0,1]) \tag{2}$$

is a sequence of finite-dimensional subspaces with the property that the union

$$\bigcup_{\ell=1}^{\infty} \mathcal{F}_\ell \tag{3}$$

of all of the nested subspaces is dense in $C([0,1])$.

The resulting evolutionary search works in epochs, corresponding to the different subspaces. During the search, the sequence of spaces is traversed, starting with relatively simple low-dimensional $\mathcal{F}_\ell$ for small ℓ. The low dimensionality n_ℓ of the initial search spaces greatly simplifies the job of the evolution strategy to come up with a coarse initial guess and to put the ES on track towards the optimum. Then, during the process, the algorithm transitions iteratively to more and more complex search spaces, increasing the epoch index ℓ in order to reach a sufficient level of detail. This added flexibility allows for a controlled iterative refinement process, guiding the evolutionary search along a path of solutions of increasing complexity towards the optimum.

Sometimes the right task granularity is known in advance. Then it may be feasible to use this fixed representation instead. However, the beauty of our approach is that it eventually reaches the necessary complexity in any case.

Starting the search in relatively low-dimensional search spaces has a strong regularization effect. However, as the dimension of the search space grows it becomes more important to further regularize the solution. We use the standard two-norm penalty $\sum_{i=1}^{d} \|f^{(i)}\|^2$ over all components of the function for this purpose.

The choice of basis functions f_i is arbitrary to some extent. Natural choices are polynomials (either monomials or a Legendre basis), or Fourier basis functions. However, these choices are *global* in the sense that changing the corresponding coefficient affects the resulting function in the whole unit interval. For the purpose of evolving first a coarse approximation of the function and adding details in later stages it is more intuitive to use a basis that allows for local modifications. Therefore Gaussian kernels are employed, ranging from relatively broad to arbitrarily peaked (and therefore localized) function primitives. This property of Gaussians is important in this context, because the effect of coefficient mutation is local, and often advantageous.

At each level of complexity ℓ a growing number of more and more peaked basis functions is centered on positions on an equidistant grid, see figure 2. The

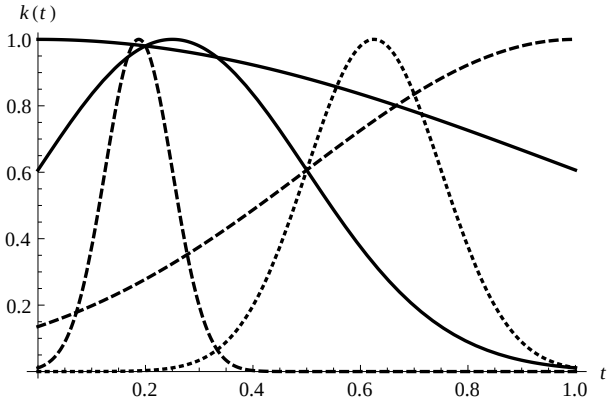


Fig. 2. Illustrative prototypical basis functions at different levels of resolution centered on basis points $b \in \{0, 1, \frac{1}{4}, \frac{5}{8}, \frac{3}{16}\}$ with $\sigma \in \{1, \frac{1}{2}, \frac{1}{4}, \frac{1}{8}, \frac{1}{16}\}$.

algorithm starts with the minimal number of two basis functions at the positions 0 and 1. At each transition each interval between two basis functions is split in half, and the new kernel width is chosen equal to the refined grid distance. This leaves us with $2^{\ell-1}+1$ additional basis functions per level of complexity, resulting in a total of $n_\ell = 2^\ell - 1 + \ell \in \mathcal{O}(2^\ell)$ basis functions. The basis functions are centered on the basis points $b_{i,\ell} = i \cdot \sigma_\ell$ for $i \in \{0, 1, \ldots, 2^{\ell-1}\}$ with $\sigma_\ell = 2^{1-\ell}$, and have the form

$$f_{(n_{\ell-1}+i)}(t) = \exp\left(-\frac{(t-b_{i,\ell})^2}{2\sigma_\ell^2}\right) = \exp\left(-2^{(2\ell-3)} \cdot (t-b_{i,\ell})^2\right) \ . \qquad (4)$$

With this choice the number of basis functions grows exponentially over the epochs. At first glance this may seem to cause computational problems. However, note that other families with polynomial growth of the number of basis functions can be used. Moreover, fast growth of the number of bases allows the algorithm to arrive quickly at the required accuracy. An equivalent point of view is that it takes only a logarithmic number of epochs to arrive at the number of basis functions required for solving a given task.

When integrating over the whole real line the inner product of each pair of these basis functions can be computed analytically. However, this is not the case for the domain $[0, 1]$, but it can easily be approximated numerically. A simple approximation is given by the expression

$$\langle f, g \rangle \approx \frac{1}{T+1} \sum_{t=0}^{T} f(t/T) \cdot g(t/T) \ , \qquad (5)$$

which has the advantage to be positive semi-definite by itself. This approximation is a natural choice if time needs to be discretized anyway, for other purposes.

The above formula (5) together with the basis functions (4) is employed in all experiments presented in the remainder of this paper.

4.2 An Illustrative Example

Let us have a look at a simple, yet non-trivial example. Consider planning a trajectory of a mobile robot in a room with obstacles. A (non-convex) room layout is depicted in figure 3. Our simplified robot is modeled as a point in this room, and it can move freely into any direction. Its trajectory is a curve $f : [0, 1] \to \mathbb{R}^2$, mapping time to position (any time interval can be re-mapped to the unit interval). Thus, in this example the $d = 2$ functions to be found correspond directly to the position of the robot in the plane.

The solution space is restricted by the condition $f(0) = \mathbf{x}_0$, since the initial position $\mathbf{x}_0$ of the robot is given. Our goal is to reach a target location at the end of the movement with an "as simple as possible" trajectory. This goal is expressed by the fitness function

$$F(f) = \|f(1) - \mathbf{x}_{\text{target}}\| + \lambda \cdot \langle f, f \rangle \ ,$$

where $\lambda > 0$ is a regularization constant. Also, the point $f(t)$ is supposed to avoid hitting a wall for all times $t \in [0, 1]$. Note that in this example the walls form highly non-convex constraints. Checking this condition amounts to collision detection, so in practice the condition is checked at discrete points in time, say, at $t \in \{0, 0.01, 0.02, \ldots, 1\}$.

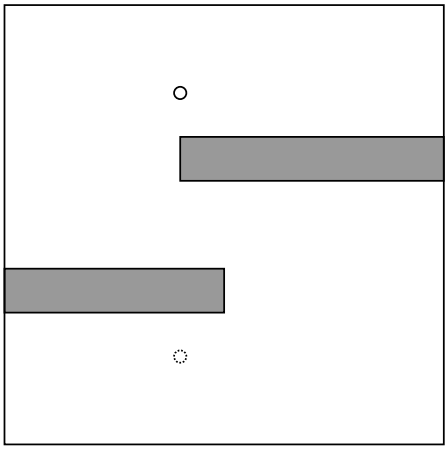


Fig. 3. Illustration of the example task of trajectory planning for a mobile "point" robot. The dotted circle indicates the starting position, and the goal is to find a continuous curve to the target location, indicated by the solid circle. The gray areas are obstacles (walls).

The constraint $f(0) = \mathbf{x}_0$ could be handled by a further term in the fitness function. However, since the initial configuration is known it is easier to encode this term into the basis functions. W.l.o.g. let us assume that $\mathbf{x}_0 = (0, 0)$. Then we modify all basis functions $f_i(t)$ by subtracting a constant, resulting in the new basis functions $\tilde{f}_i(t) = f_i(t) - f_i(0)$. This way we obtain the property $\tilde{f}_i(0) = 0$, and thus $f(0) = 0$. There are two reasons for not handling the goal position in a similar way: First, an encoding that enforces reaching the target all the time will result in a very large fraction of all solutions to be infeasible, and second, in the more involved robotic setups in the next section this would require solving the inverse kinematics problem.

The constraints (avoiding the walls) can be handled in many different ways. A simple solution is the "death penalty" strategy, which amounts to discarding (and thus never

selecting) infeasible individuals. For elitist selection this is equivalent to adding a huge (or infinite) penalty term to the fitness function, which turns the fitness into a discontinuous function.

In our example the number of real-valued functions is $d = 2$. The search starts in the first epoch with two coefficients for the widest Gaussian kernels resulting in an only four-dimensional search space of curves. The result of the first epoch is depicted in figure 4 (left). Two observations are in place: The space $\mathcal{F}_1$ is too inflexible to solve the problem, and the relatively few, simple basis functions force the trajectory to be very regular at this stage. The current solution is not good enough for our needs, so the algorithm proceeds with the next epoch, adding more peaked basis functions.

It takes the algorithm five epochs to solve the problem with high accuracy. This amounts to $2^5 + 5 - 1 = 36$ coefficients per function, and $2 \cdot 36 = 72$ coefficients in total. The result is shown in figure 4 (right). The final trajectory plan is obviously more complex than the coarse initial plan, but it is not over-complicated. Note that the objective term $\|f(1) - \mathbf{x}_{\text{target}}\|$ does not keep the trajectory from evolving all kinds of bumps in the time interval $[0, 1)$, just by chance, as a by-product of the randomized search. However, the regularization term is an elegant way of avoiding such correct but overly complex solutions.

Now let us pretend that we knew beforehand that five epochs, or 36 basis function, are sufficient for this task. What happens if we run only epoch number five of the algorithm, starting from scratch? We observe two properties of the resulting trajectories: First, the trajectories found are indeed much more complex, since the process does not profit from the implicit regularization of starting with the best solution from the previous epoch. This issue could be fixed by increasing the complexity penalty λ in the fitness function. But more importantly, the algorithm converges into a local optimum in about 50% of the runs, that is, it does not reach the target location. Thus, iteratively increasing the solution complexity stabilizes the search process.

4.3 Covariance Matrix Extension

Search in a fixed subspace $\mathcal{F}_\ell$ is relatively straightforward, once an indirect kernel-based function encoding is fixed. At the end of each epoch the evolution strategy has identified a (local) optimum in this space. If the corresponding function is not a sufficiently good solution to the problem at hand then the search space needs to be extended to the next subspace $\mathcal{F}_{\ell+1}$. Since the subspaces are nested we can reuse previously accumulated knowledge by performing a "warm-start" of the search. Projecting individuals and even the mean of the search distribution from $\mathcal{F}_\ell$ into the extended subspace $\mathcal{F}_{\ell+1}$ is rather trivial; the corresponding coefficient vectors can simply be padded with zeros.

From an evolutionary algorithms point of view the transition from a set $\{f_1, \ldots, f_n\}$ to an extended set $\{f_1, \ldots, f_n, f_{n+1}, \ldots, f_N\}$ of basis functions is a non-trivial step. In this situation the coefficient vector $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_n)$ is extended to $\boldsymbol{\alpha}' = (\alpha_1, \ldots, \alpha_N)$, and thus the xNES search distribution needs to be extended to the new coefficients. Let $\boldsymbol{\mu} \in \mathbb{R}^n$, $\sigma > 0$ and $\mathbf{B} \in \mathbb{R}^{n \times n}$

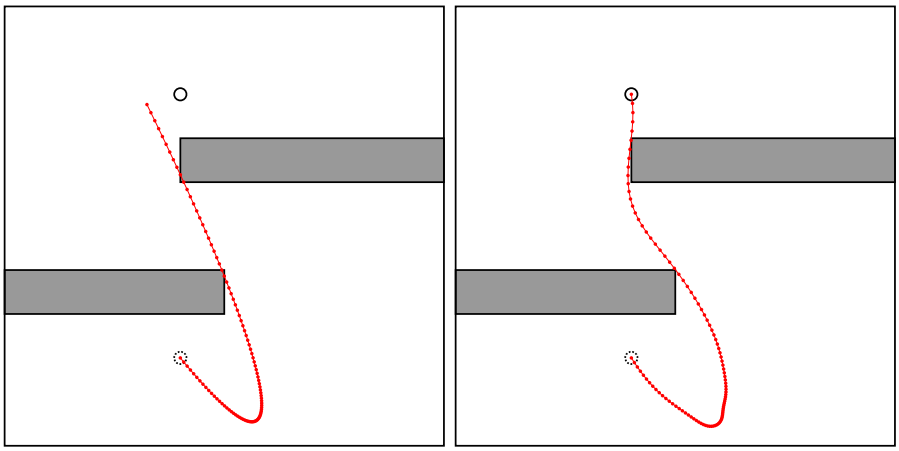


Fig. 4. Left: best trajectory found in the first epoch; right: solution of the problem, found in epoch five. Both trajectories seem to come dangerously close to the wall, which could be avoided by adding a further penalty term for a safety distance to the fitness function. This term is omitted here for simplicity. Also, the trajectory could be shorter, but again, we ignored several reasonable goals in order to keep the fitness function clean and simple.

denote distribution mean, standard deviation, and normalized covariance factor of the search distribution over $\boldsymbol{\alpha}$, and let $\boldsymbol{\mu}' \in \mathbb{R}^N$, σ', and $\mathbf{B}' \in \mathbb{R}^{N \times N}$ denote the extended parameters, describing a search distribution over the extended coefficient vector $\boldsymbol{\alpha}'$.

Three different approaches for extending the search distribution are considered. The seemingly most naïve approach is to reset the search distribution to a radially symmetrical shape, which amounts to setting the normalized covariance factor $\mathbf{B}'$ to the identity matrix of dimension N. This approach can be reasonable under the assumption that the information contained in the search distribution by the end of the previous epoch is of no or very limited use, for example, because the search distribution encoded by $\mathbf{B}$ was too much tailored towards fine tuning. However, the scale σ can be preserved, or truncated to plausible limits.

Alternatively, the evolved covariance matrix for the old coefficients can be kept, which leaves us with the problem of filling in the new entries of the matrix $\mathbf{B}'$. One seemingly canonical way of extending the search distribution is to pick the parameters that minimize the Kullback-Leibler divergence (or any other distance measure) between the old and the extended search distribution. However, because the old distribution can be represented exactly with the new set of coefficients this would amount to not using the new components at all, which does not result in additional flexibility. Instead, we need a compromise between preserving learned knowledge encoded in the current search distribution, and

extending the search to new directions. The simplest extension takes the form

$$\mathbf{B}' = \begin{pmatrix} \mathbf{B} & 0 \\ 0 & \mathbb{I} \end{pmatrix} \quad , \tag{6}$$

where $\mathbb{I}$ denotes the identity matrix of appropriate dimension. However, the previously available and the extended basis functions are not independent. Although we can not know the coupling of old and extended coefficients that will result in optimal progress of the ES on the current fitness function, we can take the structure of the basis functions themselves into account. Ideally, the extension of the search distribution should have the following properties:

1. The distribution center (encoding the currently best solution in (1+1)-xNES) should not change.
2. The covariance of the extended distribution, projected to the subspace spanned by the previously available directions, should also be preserved.
3. In the newly available subspace orthogonal to the previously available directions the covariance matrix should be a multiple of the unit matrix, such as $\sigma^2\mathbb{I}$.

These properties involve a decomposition into orthogonal subspaces and thus involve the inner product. The non-trivial assumption here is that the Hilbert space structure induced by the kernel framework allows for a useful problem decomposition. Note that the simpler strategy of extending the covariance factor with the unit matrix makes a different assumption, namely that the covariance matrix restricted to the space spanned by the new basis functions should be $\sigma^2\mathbb{I}$. However, this space is usually not orthogonal to the search space of the previous epoch. Property 3 requires a uniform extension only in the subspace orthogonal to the space in which self-adaptation has already taken place.

The following text thoroughly describes the technical details of fulfilling property 3. As usual in kernel methods, all computations needed to extend the covariance matrix can be expressed in terms of inner products of the basis functions. Let $\mathbf{K} \in \mathbb{R}^{N\times N}$ denote the kernel Gram matrix with entries $K_{ij} = \langle f_i, f_j\rangle$, collecting all pairwise inner products. The matrix is split into the sub-matrices

$$\mathbf{K} = \begin{pmatrix} \mathbf{K}_{oo} & \mathbf{K}_{oe} \\ \mathbf{K}_{eo} & \mathbf{K}_{ee} \end{pmatrix} \quad ,$$

where o and e indicate *old* indices $\{1,\ldots,n\}$ and *extended* indices $\{n+1,\ldots,N\}$, respectively. Let further $\Phi = (f_1,\ldots,f_N)$ denote the collection of basis functions, thought of as "column" feature vectors, and let Φ_o and Φ_e denote the corresponding sub-matrices, split into old and extended basis functions. Then we have $\mathbf{K} = \Phi^T\Phi$, $\mathbf{K}_{oe} = \Phi_o^T\Phi_e$, and so on.

The split of the function space $\mathcal{F}' = \mathrm{span}(f_1,\ldots,f_N)$ into $\mathcal{F} = \mathrm{span}(f_1,\ldots,f_n)$ and its orthogonal complement $\mathcal{F}^\perp$ can be expressed by means of the orthogonal projection $\Pi : \mathcal{F}' \to \mathcal{F}'$ onto $\mathcal{F}$, $\Pi(f) = \Phi_o(\Phi_o^T\Phi_o)^{-1}\Phi_o^T f$. This linear mapping

written in matrix notation becomes $\mathbf{\Pi} = \Phi_o(\Phi_o^T\Phi_o)^{-1}\Phi_o^T$.[2] The corresponding projection for the coefficients $\boldsymbol{\alpha}$ is $\pi : \mathbb{R}^N \to \mathbb{R}^N$, $\boldsymbol{\pi}(\boldsymbol{\alpha}) = \Phi^{-1}\mathbf{\Pi}\Phi\boldsymbol{\alpha}$. We rewrite

$$\begin{aligned}\boldsymbol{\pi} &= \Phi^{-1} \cdot \mathbf{\Pi} \cdot \Phi \\ &= \Phi^{-1} \cdot \Phi_o(\Phi_o^T\Phi_o)^{-1}\Phi_o^T \cdot \Phi \\ &= \left[(\Phi^T\Phi)^{-1} \cdot (\Phi^T\Phi)\right] \cdot \Phi^{-1} \cdot \Phi_o(\Phi_o^T\Phi_o)^{-1}\Phi_o^T \cdot \Phi \\ &= (\Phi^T\Phi)^{-1} \cdot \Phi^T\Phi_o \cdot (\Phi_o^T\Phi_o)^{-1} \cdot \Phi_o^T\Phi \\ &= \mathbf{K}^{-1} \cdot \begin{pmatrix}\mathbf{K}_{oo} \\ \mathbf{K}_{eo}\end{pmatrix} \cdot \mathbf{K}_{oo}^{-1} \cdot \begin{pmatrix}\mathbf{K}_{oo} & \mathbf{K}_{oe}\end{pmatrix} \ .\end{aligned}$$

The matrices $\mathbf{\Pi}$ and $\boldsymbol{\pi}$ allow to represent the two different sub-spaces involved in properties 2 and 3. The old search space is the image of the projection, and the component of a vector orthogonal to the old subspace is the vector itself minus its projection.

Let us introduce the basis functions $\bar{f}_i = f_i - \Pi(f_i)$ for the extended indices, which are projected to $\mathcal{F}^\perp$. Thus, the vectors $\{\bar{f}_{n+1}, \ldots, \bar{f}_N\}$ span the orthogonal complement of the old search space. Then properties 2 and 3 in the above list indicate that the covariance matrix w.r.t. coefficients $\boldsymbol{\beta}$ of the basis functions $\{f_1, \ldots, f_n, \bar{f}_{n+1}, \ldots, \bar{f}_N\}$ should be chosen as

$$\overline{\mathbf{\Sigma}'} = \begin{pmatrix}\mathbf{\Sigma} & 0 \\ 0 & \sigma^2\mathbb{I}\end{pmatrix}.$$

The coefficients $\boldsymbol{\beta}$ are linearly related to $\boldsymbol{\alpha}'$ by $\boldsymbol{\beta} = \mathbf{M}\boldsymbol{\alpha}'$, with the matrix

$$\mathbf{M} = \begin{pmatrix}2\mathbb{I} & 0 \\ 0 & \mathbb{I}\end{pmatrix} - \boldsymbol{\pi} \ .$$

Hence, the extended covariance matrix for the coefficients $\boldsymbol{\alpha}'$ is given by

$$\mathbf{\Sigma}' = (\mathbf{M}^{-1})^T \cdot \overline{\mathbf{\Sigma}'} \cdot (\mathbf{M}^{-1}) \ ,$$

which amounts to the update

$$\mathbf{B}' = \begin{pmatrix}\mathbf{B} & 0 \\ 0 & \mathbb{I}\end{pmatrix} \cdot (\mathbf{M}^{-1}) \tag{7}$$

of the normalized covariance matrix factor.

This covariance matrix extension is the canonical choice in the sense of properties 2 and 3. It can be computed relatively straightforward with simple matrix operations involving the kernel Gram matrix $\mathbf{K}$, the old covariance factor $\mathbf{B}$, and the scaled unit matrix $\sigma^2\mathbb{I}$ on the new components.

When adapting multiple functions simultaneously, there is no direct structural connection between coefficients of kernels corresponding to different components. Thus, when evolving a function $f : [0, 1] \to \mathbb{R}^d$ for $d > 1$, the kernel

[2] For any (not necessarily square) matrix M we write M^{-1} for the Moore-Penrose pseudo-inverse.

matrix $\mathbf{K}$ may be replaced by a d-fold block diagonal matrix with the kernel Gram matrix in each block. However, depending on the task, the connection between different components may be clear, in which case this general scheme should be modified accordingly.

4.4 Switching between Epochs and Representations

A principled ES for search in the nested spaces $\mathcal{F}_\ell$ should provide an automated way of deciding when to switch from an underlying representation to the next more complex one. Intuitively, this should happen whenever better progress can be made on an extended set of basis functions, and at the latest when the evolution strategy converges.

The only reasonable switching criteria that can be derived from the sequence of fitness values, success rates, or the internal state of the ES are related to convergence. Such criteria are typically used as stopping conditions. For example, the algorithm is stopped when the step size σ of xNES falls below some value $\sigma_{\min}$, or the number of fitness evaluations exceeds a threshold N_0. When dealing with nested search spaces the algorithm moves on to the next representation instead of stopping.

The resulting switching behavior is not necessarily the best possible. It may be advantageous to switch the representation earlier in order to avoid the possibly long convergence phase. Thus, we want the algorithm to switch to the next more complex representation *as soon as this pays off*. It is virtually impossible to find a good switching point by just looking at the history of fitness values or success rates. Instead one can use a trial-and-error procedure, which comes at the cost of a few additional fitness evaluations: Every T generations of the ES we sample a set of λ_{ext} points from the current search distribution, but extended to the next set of basis functions. We obtain fitness values $\{f_1^{\text{ext}}, \ldots, f_{\lambda_{\text{ext}}}^{\text{ext}}\}$ for the extended representation, and $\{f_1^{\text{cur}}, \ldots, f_{\lambda_{\text{ext}}}^{\text{cur}}\}$ for the current search distribution, by projecting the search points back to the current set of basis functions. The question whether the extended fitness values systematically outperform the current ones can be answered based on the confidence score of a one-sided Mann-Whitney U-test (also known as Wilcoxon rank sum test), with null hypothesis $f^{\text{ext}} \geq f^{\text{cur}}$. The algorithm decides to switch to the extended set of basis functions if the null hypothesis is rejected at a confidence level of p_0, indicating that the extended representation makes consistently better progress, or if the global step size σ falls below a critical threshold σ_0, indicating convergence of the ES.

The parameters T, λ_{ext}, and p_0 have to be chosen carefully. We propose to set the constants to conservative values, because switching between levels of complexity works only in one direction. Missing an opportunity to switch is not dramatic, because most probably the switch will then happen after only T further iterations. However, increasing the complexity too early during the optimization may seriously impair performance. In the experiments, it turns out that such an automated switching technique is indeed hard to adjust, and that in practice it is easier to rely solely on convergence detection, which also saves the additional

Algorithm 2: Epoch-based search with iterative refinement of the representation.

Input: sequence of basis functions $(f_i)_{i\in\mathbb{N}}$,
sequence of cardinalities $(n_\ell)_{\ell\in\mathbb{N}}$,
solution dimension d,
initial search point $\boldsymbol{\mu} \in \mathbb{R}^{d\cdot n_\ell}$,
initial step size σ,
fitness function F,
convergence criterion, e.g. a threshold,
covariance matrix extension mechanism EXTEND(), e.g., equation (7).

epoch counter: $\ell \leftarrow 1$
$\mathbf{B} \leftarrow \mathbb{I}$
while *stopping condition not met* **do**
  iterate xNES: $(\boldsymbol{\mu}, \sigma, \mathbf{B}) \leftarrow \text{xNES}(\boldsymbol{\mu}, \sigma, \mathbf{B}, F)$
  if *convergence (e.g., $\sigma <$ threshold)* **then**
    // start next epoch:
    $\ell \leftarrow \ell + 1$
    $\boldsymbol{\mu} \leftarrow \begin{pmatrix} \boldsymbol{\mu} \\ 0 \end{pmatrix} \in \mathbb{R}^{d\cdot n_\ell}$
    optionally truncate the range of sigma: $\sigma \leftarrow \max\{\sigma, \sigma_0\}$
    extend the covariance matrix factor: $\mathbf{B} \leftarrow \text{EXTEND}(\mathbf{B})$
  end
end
return $\boldsymbol{\mu}$

fitness evaluations. The epoch-based algorithm for search in infinite-dimensional function spaces is summarized in algorithm 2.

4.5 Comparison to Alternative Methods

Depending on the context there are many different ways to represent and learn continuous functions. The method of choice depends in particular on properties of the objective function and of eventual constraints. Our approach is particularly useful if alternative, more efficient techniques such as analytic solutions of regression problems, gradient-based methods, or convex optimization are not applicable. Thus, our method is targeted at non-smooth or even discontinuous objective functions and at non-convex constraints. It is even applicable if no analytic expression for the constraints is available at all. Simple constraint handling techniques for evolutionary algorithms, such as resampling and the death-penalty strategy, require only that the feasibility of a search point can be queried (e.g., by a call to a function returning a boolean, similar to the fitness function returning a real number). In such situations most standard methods are not applicable at all, leaving direct search as the only option.

The problem of trajectory planning while avoiding collisions with arbitrary obstacles is of exactly this type. The feasible region is in general non-convex, which can be expressed either with non-convex constraints or with a discontin-

uous objective function. A robot arm configuration is a function of the joint angles, and therefore trajectory planning naturally takes place in joint angle coordinates. On the other hand, obstacles are represented in task coordinates (typically three-space), and no analytic expression is available in joint angle (configuration) space. The two spaces are connected by the kinematics of the robot, which is a unique and simple-to-compute mapping in the forward direction (joint angles to task space configuration), but inverse kinematics is in general non-unique and much harder to compute. Also, collision detection in task-space is typically conceptually simple and computationally cheap. Our approach requires only forward kinematics and collision detection.

Dynamical systems approaches, such as the potential field method, are a popular alternatives for trajectory planning. One advantage is that such methods can naturally work online. The potential field dynamics can be defined either in configuration space or in task space. When planning in configuration space, the obstacles need to be transformed into joint angle representations, such that appropriate repellers can be placed. In a non-trivial robotics scenario this is generally infeasible. Dynamics in task space are easy to define. This has the disadvantage that a large share of the complexity of path planning is left to a sub-module that translates movements of the end-effector in task space into joint angles. This amounts to continuously solving (at least a linearized version of) the inverse kinematics problem.

Another conceptual distinction to online methods is as follows: Online methods find exactly one trajectory, by computing the best control output in each step, typically without ever making a "global" movement plan. Our offline method does make such a global movement plan, at the expense of searching a possibly large number of trajectories. These offline simulated trajectories can be thought of as "mental trials", and execution starts only once a sufficiently good plan is identified. Of course, this offline planning paradigm requires a certain level of continuity of the environment. An online method is more suitable in scenarios that require continuous adaptation of the movement plan during the movement due to quickly or abruptly changing circumstances.

In the light of this discussion our method compares favorably for its minimal prerequisites. However, depending on the application, it may be limited by its computational needs for simulating a possibly large number of candidate trajectories in a forward kinematics model, including collision detection, and by its offline character.

5 Experiments

In this section, our method is evaluated on two different robot planning tasks. Redundant robot arms consisting of a number of linked segments are placed in environments with obstacles. The tasks challenge the algorithm with the difficulty of handling the constraints, a problem that can equivalently be expressed by a non-continuous fitness function, making evolutionary strategies interesting candidates. The method generates solutions with arbitrary precision, from coarse

functions represented by few kernels to fine solutions constructed from a large number of kernels.

The complexity of the task can be tuned by adjusting the length of the kinematic chain of the arm as well as by placing an arbitrary number of obstacles in the environment. The obstacles can make the problem really difficult by forming a narrow, curved corridor in a high-dimensional search space that leads to the optimal solution. This is the case in particular in joint space coordinates. The mapping from joint angles to Cartesian task space is highly non-linear, especially for long kinematic chains. Thus, even obstacles with a relatively simple structure in Cartesian space, such as boxes, result in highly curved and typically non-convex boundaries of the feasible region in joint space.

The following sections introduce the robot models, the parameterization of trajectories, collision detection, and the form of the fitness function. Then the actual experiments and results are presented.

5.1 Arm Models

The first benchmark uses a kinematics-only variant of an octopus-like highly redundant arm [33]. This two-dimensional plant allows for easy design of interesting benchmarks, and both kinematics and collision detection are easy to implement and very fast to compute. As already mentioned in the introduction we focus on evolution of a kinematic plan, under the assumption that the actual control, involving the dynamics of the arm, can be decoupled from this problem. Using the kinematic simulation only reduces complexity of the simulation and speeds up the objective function evaluation significantly.

The octopus arm consists of $S = 8$ segments, with (arbitrarily chosen) lengths

$$\mathbf{\Lambda} = \big(0.9, 0.9, 0.8, 0.8, 0.7, 0.7, 0.6, 0.5\big) \ .$$

The kinematics of each joint are described by an initial position and upper and lower joint limits. The initial positions are set to 0, and joints are limited to the interval $[-2.5, 2.5]$ (in radians). Given an anchor point $\mathbf{x}_0$ as well as joint angles $\xi_1, \ldots, \xi_S$ the positions of all joints can be computed recursively as

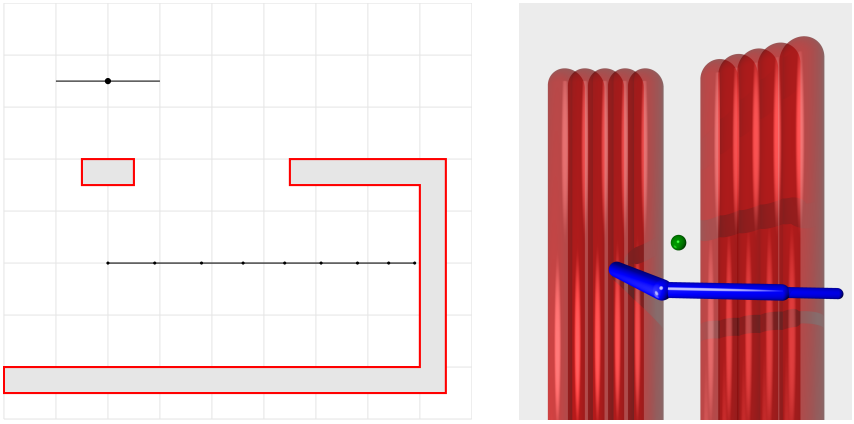
$$\mathbf{x}_n = \mathbf{x}_{n-1} + \Lambda_n \cdot \begin{pmatrix} \cos(\sum_{i=1}^{n} \xi_i) \\ \sin(\sum_{i=1}^{n} \xi_i) \end{pmatrix} \ . \tag{8}$$

In a second experiment this setup is extended to a redundant humanoid robot arm with seven degrees of freedom (7-DOF). A nice side effect of planning in joint space is that our method works unchanged in three-space, without even affecting its complexity.

The three-dimensional 7-DOF arm follows the typical humanoid setup of three DOF in the shoulder, and two DOF in the elbow and the wrist, respectively, with plausible human-like joint limits. In the (initial) zero position the arm is spread out sideways, with the elbow slightly angled. Its kinematics can be computed along the chain with the same recursive procedure as for the two-space

arm, for example, by concatenating parameterized 4×4 affine transformation matrices.

The initial configurations of the robot arms together with the task constraints (obstacles) and target locations are illustrated in figure 5.



(a) (b)

Fig. 5. Environments and initial configurations in the different trajectory planning tasks. The octopus arm (a) and the humanoid 7-DOF (b) arm are shown in their initial configurations, with kinematic chains from left to right. In the left figure the target position is indicated by the dot in the upper left. The target is moving forth and back twice on a line during the second half of the time frame (see section 5.5). In the three-dimensional figure on the right the target location is indicated by a little ball, which is placed behind two big obstacles, such that the arm needs to reach through a narrow slit in between.

5.2 Trajectory Parameterization

Trajectories are encoded as tuples of functions, with each joint angle being a function of time $t \in [0, 1]$. This amounts to eight functions (eight joints) for the octopus arm and seven functions for the 7-DOF humanoid arm.

The joint limits are a special type of constraint. They form a simple box independent of the task in the joint angle space. Therefore, it is reasonable to encode these constraints implicitly into the parameterization. We achieve this by translating function values to joint angles with a sigmoid. For a function value f, lower and upper limits L_l and L_u, and zero position angle ξ_0, the joint angle ξ is computed as

$$\xi = \sigma(f) = \begin{cases} \xi_0 + (\xi_0 - L_l) \cdot \tanh\left(\frac{f}{\xi_0 - L_l}\right) & \text{for } f \leq 0 \ , \\ \xi_0 + (L_u - \xi_0) \cdot \tanh\left(\frac{f}{L_u - \xi_0}\right) & \text{for } f \geq 0 \ . \end{cases}$$

This piecewise defined sigmoid function makes sure that the joint angle is in the zero position ξ_0 for a function value of zero, and that the joint limits are only reached in the limits of $f = \pm\infty$.

This property also allows for a simple encoding of the constraint that the trajectory needs to start in the given initial arm configuration. It is reflected by the following change in the basis functions: Instead of the Gaussians f_i the shifted Gaussians $f_i'(t) = f_i(t) - f_i(0)$ are used, automatically fulfilling the property $f(0) = \sum_{i=1}^{n} \alpha_i f_i'(0) = 0$ for all coefficient vectors $\boldsymbol{\alpha}$.

In summary, the indirect trajectory encoding is defined by the following chain of computations:

$$\begin{pmatrix} \alpha_{1,1} & \dots & \alpha_{S,1} \\ \vdots & & \vdots \\ \alpha_{1,n} & \dots & \alpha_{S,n} \end{pmatrix} \mapsto \begin{pmatrix} f^{(1)}(t) = \sum_{i=1}^{n} \alpha_{1,i} \big[f_i(t) - f_i(0) \big] \\ \vdots \\ f^{(S)}(t) = \sum_{i=1}^{n} \alpha_{S,i} \big[f_i(t) - f_i(0) \big] \end{pmatrix} \mapsto \begin{pmatrix} \sigma_1(f^{(1)}(t)) \\ \vdots \\ \sigma_S(f^{(S)}(t)) \end{pmatrix}$$

In the first step the coefficients $\boldsymbol{\alpha}$ are translated into functions $f^{(i)}(t)$ encoded as kernel expansions in the basis functions $f_i(t) - f_i(0)$ (which ensures $f^{(i)}(0) = 0$). In the second step the unconstrained function values are squeezed into the joint limits with a sigmoid. The result is a trajectory $\boldsymbol{\xi}(t)$ in joint space. This joint-space trajectory can be translated into an operational space trajectory $\mathbf{x}(t)$ by means of the arm model (8).

5.3 Collision Detection

The search aims for movement plans without arm self-collisions and collisions with obstacles. The arm as well as all obstacles are modeled as unions of simple geometries. In the two-dimensional case of the octopus these are line segments, and all geometries in the three-dimensional case are represented as capsules (cylinders with rounded ends). Let S be the number of segments of the robot arm, and let O be the number of simple geometries describing the obstacles. Then a brute-force collision test takes $\mathcal{O}(S^2 + O \cdot S)$ operations. The number S is bounded a priori, and the test scales linear in the number O of obstacles. In our experiments this number is low enough such that the brute-force test of all possibly colliding pairs is feasible (see also figure 5).

Time is discretized into 100 steps, and thus into 101 configurations at times $\mathcal{T} = \{0, 0.01, 0.02, \dots, 1\}$. Instead of much more complex continuous time collision detection, this discretization allows us to compute collisions only for fixed configurations, which greatly increases computational efficiency and simplifies the implementation.

5.4 Fitness Function

In the most general formulation the robots are supposed to reach with the tip $\mathbf{x}_S$ of the arm for a target location $\mathbf{x}_{\text{target}}(t)$ within the time interval $[t_{\text{target}}, 1]$. In the simplest case $t_{\text{target}} = 1$ and $\mathbf{x}_{\text{target}}$ is a single point. As mentioned before,

the arm starts from a given initial configuration and it is supposed to avoid self-collisions and collisions with obstacles during the whole time course. On top of that the algorithm should avoid over-complex solutions. These goals are encoded in the following form of the fitness function:

$$\begin{aligned} F = \sum_{t \in \mathcal{T}} \Bigg[& \gamma_{\text{target}} \cdot \mathbf{1}_{[t_{\text{target}},1]}(t) \cdot \left\| \mathbf{x}_S(t) - \mathbf{x}_{\text{target}}(t) \right\| \\ & + \sum_{s_1=1}^{S} \left(\sum_{s_2=1}^{s_1-2} C_a\big(d(A_{s_1}(t), A_{s_2}(t))\big) + \sum_{s_2=s_1+2}^{S} C_a\big(d(A_{s_1}(t), A_{s_2}(t))\big) \right) \\ & + \sum_{s=1}^{S} \sum_{o=1}^{O} C_o\big(d(M_o, A_s(t))\big) \Bigg] \\ & + \gamma_c \cdot \sum_{s=1}^{S} \boldsymbol{\alpha}_s^T \mathbf{K} \boldsymbol{\alpha}_s \end{aligned}$$

The coefficients γ_{target} and γ_c encode the relative importance of the different sub-goals: reaching the target, avoiding collisions, and finding a low complexity solution, respectively. The vectors $\mathbf{x}_S(t)$ and $\mathbf{x}_{\text{target}}(t)$ denote the positions of the tip of the arm and the target. The arm segments are encoded as the sets $A_s(t)$ and the (in this case static) obstacles as M_o, with S and O denoting the number of arm segments and obstacles. These sets are compared with the usual distance function $d(A, B) = \min\left\{ \|a - b\| \,\middle|\, a \in A, b \in B \right\}$ for compact sets A and B. The vectors $\boldsymbol{\alpha}_s$ hold the coefficients encoding function number $s \in \{1, \ldots, S\}$ that controls joint angle ξ_s. The discontinuous collision penalty function C_a takes a value of 10^{10} at zero, indicating collision, and zero otherwise. The desire to stay away from obstacles is encoded in the ramp shaped penalty function

$$C_o(d) = \begin{cases} 10^{10} & \text{for } d = 0, \\ \gamma \cdot (d_0 - d) & \text{for } 0 < d < d_0, \\ 0 & \text{otherwise.} \end{cases}$$

The complexity control constant γ_c is setup such that the coefficients are typically kept within a range of about ± 10, resulting in smooth trajectories. This is an easy to check criterion. The evaluation of the fitness function requires the execution of the plan on the kinematic model, including collision detection in each time frame.

The (1+1)-xNES hill-climber is applied in all experiments. The draconic collision penalty of 10^{10} effectively results in a restriction to feasible collision free movements, because the fitness of any trajectory involving colliding poses is far worse than the initial fitness of the motionless arm in its zero position (corresponding to the starting point $\boldsymbol{\alpha} = \mathbf{0}$). Hence, in this setting constraint handling via a fitness penalty and other established techniques—such as death penalty constraint handling—are equivalent, and the discontinuity in fitness can be translated one-to-one into the complicated task constraints in joint space

Algorithm 3: Robot simulation and fitness evaluation for the octopus arm.

$F \leftarrow 0$
for $s \in \{1, \ldots, S\}$ **do**
  $f^{(s)}(t) = \sum_{i=1}^{n} \alpha_{i,s} \cdot [f_i(t) - f_i(0)]$
  $\sigma_s(f) = \begin{cases} (\xi_0)_s + ((\xi_0)_s - (L_l)_s) \cdot \tanh\left(\frac{f}{(\xi_0)_s - (L_l)_s}\right) & \text{for } f \leq 0 \ , \\ (\xi_0)_s + ((L_u)_s - (\xi_0)_s) \cdot \tanh\left(\frac{f}{(L_u)_s - (\xi_0)_s}\right) & \text{for } f \geq 0 \ . \end{cases}$
  $F \leftarrow F + \gamma_c \cdot \boldsymbol{\alpha_s}^T \mathbf{K} \boldsymbol{\alpha_s}$
end
for $t \in \{0, 0.01, 0.02, \ldots, 1\}$ **do**
  $\beta \leftarrow 0$
  for $s \in \{1, \ldots, S\}$ **do**
    $\xi^{(s)}(t) \leftarrow \sigma_s\left(f^{(s)}(t)\right)$
    $\beta \leftarrow \beta + \xi_s(t)$
    $\mathbf{x}_s \leftarrow \mathbf{x}_{s-1} + \Lambda_s \cdot \begin{pmatrix} \cos(\beta) \\ \sin(\beta) \end{pmatrix}$
    $A_s \leftarrow$ line segment $[\mathbf{x}_{s-1}(t), \mathbf{x}_s(t)]$
  end
  $F \leftarrow F + \gamma_{\text{target}} \cdot \mathbf{1}_{[t_{\text{target}}, 1]}(t) \cdot \left\| \mathbf{x}_S(t) - \mathbf{x}_{\text{target}}(t) \right\|$
  $F \leftarrow F + \sum_{s_1=1}^{S} \left(\sum_{s_2=1}^{s_1-2} C_a\big(d(A_{s_1}(t), A_{s_2}(t))\big) + \sum_{s_2=s_1+2}^{S} C_a\big(d(A_{s_1}(t), A_{s_2}(t))\big) \right)$
  $F \leftarrow F + \sum_{s=1}^{S} \sum_{o=1}^{O} C_o\big(d(M_o, A_s(t))\big)$
end
return F

posed by the obstacles. The whole process of function composition, joint limit handling, forward kinematics, collision detection, and evaluation of the regularized fitness function is summarized in algorithm 3.

5.5 Setup

The tasks for the two robot arms, consisting of initial configuration, obstacles, and target location, are illustrated in figure 5.

The octopus arm faces a casing-like obstacle that keeps it from moving straight (in joint space) to the target location. A second obstacle interferes with movements close to the target. In the first half of the time frame the octopus arm needs to reach the target location, avoiding the obstacles. In the second half of the time frame the target starts to move, making two oscillation around its initial position. The target location relative to the shoulder (anchor) point $\mathbf{x}_0$ of

the octopus arm is given by

$$\mathbf{x}_{\text{target}}(t) = \begin{cases} \begin{pmatrix} 0 \\ 3.5 \end{pmatrix} & \text{for } t \leq 0.5, \\ \begin{pmatrix} \sin(8\pi t) \\ 3.5 \end{pmatrix} & \text{for } t > 0.5. \end{cases}$$

The octopus arm first needs to contract and turn to reach the target location around the obstacles. Then it needs to track the moving target as closely as possible.

The humanoid 7-DOF arm is confronted with the task to reach to a target location that is hidden behind two large wall-like obstacles. The only way to reach the goal location is to carefully reach through a narrow slit in between the two obstacles. The slit is in no way aligned with the robot's joints. Thus, this task requires a highly coordinated movements of different joints.

5.6 Results

The octopus arm trajectory in a typical run is shown in figure 6. In the first epoch the algorithm quickly manages to move the arm towards the target, where the obstacles force it to contract and turn in a controlled way. In a few trials it reaches local optima with the arm trapped behind an obstacle in a nearly self-colliding position. However, in the majority of runs the algorithm manages to find a coarse initial movement plan towards the goal, while following the moving target is far beyond the available solution complexity within the first epoch. In later epochs, typically in epochs three or four, the fitness starts to improve again considerably. At this time the solution space has become sufficiently rich to allow for tracking the moving target. The trajectory becomes more and more refined, finally following the moving target with convincing precision.

The second experiment demonstrates that our method is suitable for realistic robots, and easily scales to this case. A typical trajectory is depicted in figure 7. The algorithm managed to solve this problem in 9 out of 10 trials, which makes multi-start strategies nearly superfluous. The solution is consistently found already at the first level of complexity, with only two coefficients per joint. A very good approximation is already available after around 1.000 fitness evaluations, which takes around one second on a standard laptop. Thus, the method is feasible for real-time planning.

In both experiments we observed differences in fitness between trials due to random effects introduced by the evolution strategy. The experiments indicate that these effects exceed the impact of the methods used for switching between epochs and extending the covariance matrix. We found nearly no impact of the different strategies for extending the covariance matrix, and any sufficiently conservative epoch switching strategy results in qualitatively equivalent trajectories. Thus, in accordance with Occam's razor, we rely on the simplest such strategy in our experiments, as follows: We switch to the next epoch only when the evolution strategy converges, by checking whether the step size parameter falls below

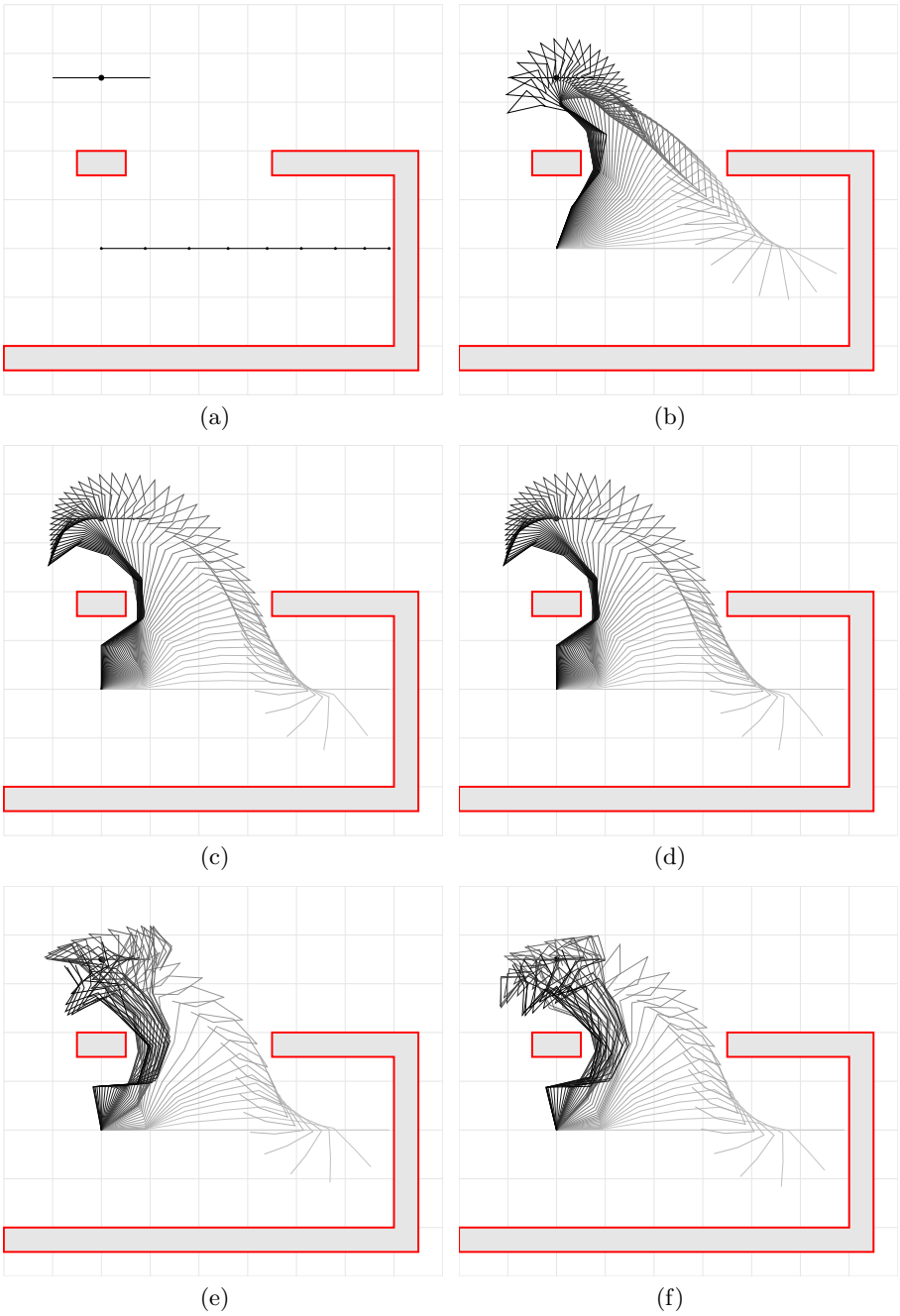


Fig. 6. Following a moving target under obstacle avoidance constraints. Sub-figure (a) shows the initial setup of the arm, with obstacles in place. The moving target, which performs two sine wave movements during the second half of the time frame, is in the upper left. To solve the task, the arm needs to contract first, then turn and extend around the small obstacle, and finally follow the target with its tip. Sub-figures (b) to (f) correspond to the solutions at the end of epochs 1 to 5, with the movement of the arm depicted in shades of gray. The behavior changes qualitatively at epoch 4 because the solution complexity becomes sufficiently rich to follow the moving target.

a small threshold, in this case 10^{-14}. For the next epoch we reset this parameter to a more reasonable value of 10^{-8} and extend the covariance matrix factor with the unit matrix.

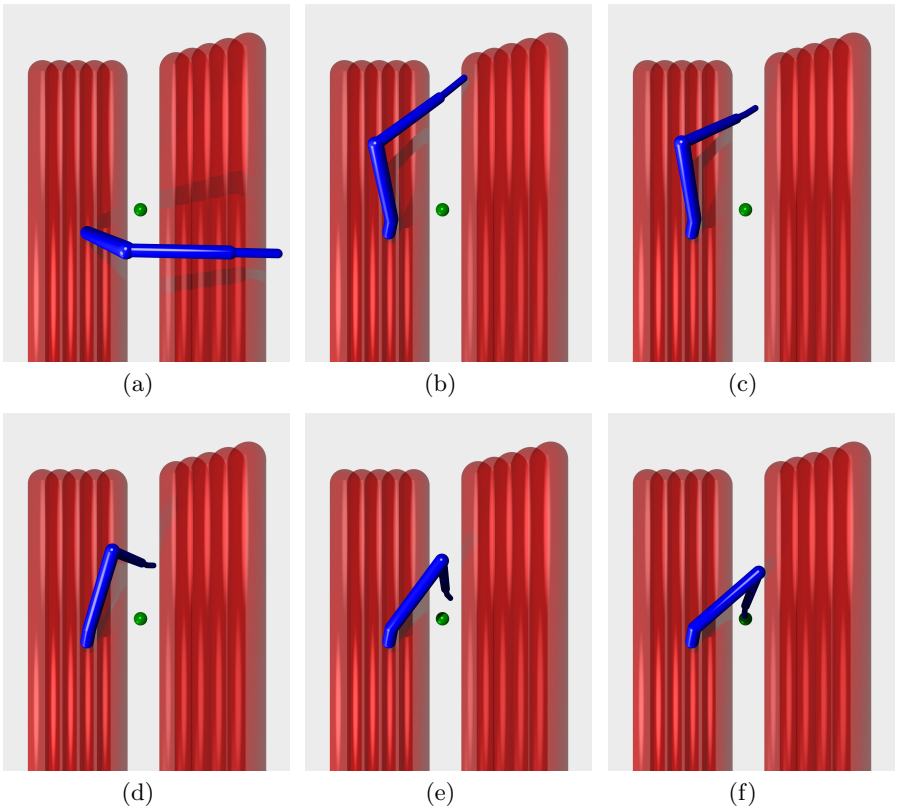


Fig. 7. Trajectory planning for a 7-DOF robot arm. The goal of the robot (blue) is to reach the target position (green), through the narrow slit between the obstacles (red). Sub-figures (a) to (f) show time steps $t \in \{0, 0.25, 0.5, 0.75, 0.9, 1\}$.

6 Discussion

The results have a number of implications. First of all, we have shown that iteratively increasing the solution complexity results in more robust solutions, compared to starting with a (fixed) high complexity. Furthermore, increasing the solution complexity in an iterative manner allows the method to figure out the 'right' level of complexity for the problem at hand by itself.

The seemingly complex trajectory planning problem for the 7-DOF humanoid arm can be solved with only two coefficients per joint. This illustrates the power

of our parameterization of squashing a Gaussian kernel expansion with a sigmoid function between the joint limits. For complex tasks the method indeed profits from an increase in solution complexity.

We have found that the epoch switch and the covariance matrix extension methods can be conservative and simple in practice. This relieves us from tuning sensitive hyper-parameters. We conjecture that the reasons might be as follows: Switching too late is better than switching too early, which is why a conservative switching strategy is more important than an elaborate one. Furthermore, the self-adaptation capability of the evolution strategy can make up for possible sub-optimal covariance matrix extensions. This indicates that for robustness and simplicity of implementation, relatively simple strategies should be used in practice.

The xNES algorithm is designed to follow the global trend of the fitness landscape and to quickly identify a (local) optimum. However, the trajectory planning problem often results in highly multi-modal landscapes, where the arm can get trapped behind an obstacle or in a self-collision. Thus, in principle, our method should be combined with a technique dealing with local optima. In the tasks investigated in this study this turns out to be superfluous, and in general a simple restart strategy should be sufficient as long as the success probability is sufficiently high.

Our method is a powerful tool for trajectory planning under constraints, such as obstacle avoidance. The experiment with a humanoid 7-DOF arm model meets real-time constraints, mostly because the solution can be well approximated at the first level of complexity. However, being based on an evolutionary algorithm, the method can easily be parallelized. Parallelism is even easier to exploit in a multi-start strategy.

7 Conclusion

We propose a principled framework for evolutionary learning in the infinite dimensional search space of continuous functions on the unit interval. This is a very general setting, with many potential applications. Evolutionary algorithms allow for searching in the presence of discontinuous objective functions and/or non-trivial task constraints.

The proposed method of exhausting (a dense subspace of) the space of continuous functions with a nested sequence of finite dimensional sub-spaces makes evolutionary search tractable without loss of conceptual flexibility provided by the full function space.

The method profits from the kernel framework in two ways: First, the squared norm in function space is a natural regularizer avoiding over-complex solutions. Second, the kernel framework provides a canonical way of extending the evolutionary search distribution when iterating over a sequence of nested sub-spaces.

The trajectory planning experiments demonstrate the advantage of iteratively evolving gradually more complex function representations. For a flexible, redundant arm in two-space and a 7-DOF robot arm in three-space the resulting

evolution strategy successfully copes with difficult trajectory learning problems involving moving targets and obstacle avoidance.

References

1. Alfaro, T., Rojas, M.C.R.: An on-the-fly evolutionary algorithm for robot motion planning. In: ICES. pp. 119–130 (2005)
2. Barraquand, J., Latombe, J.C.: Robot Motion Planning: A Distributed Representation Approach. The International Journal of Robotics Research 10(6), 628–649 (Dec 1991)
3. Beyer, H.G., Schwefel, H.P.: "Evolution strategies"–a comprehensive introduction. Natural Computing 1, 3–52 (2002)
4. Boser, B.E., Guyon, I.M., Vapnik, V.N.: A training algorithm for optimal margin classifiers. In: Proceedings of the fifth annual workshop on Computational learning theory. pp. 144–152. COLT '92, ACM, New York, NY, USA (1992)
5. Conkur, E.S., Buckingham, R.: Manoeuvring highly redundant manipulators. Robotica 15, 435–447 (July 1997)
6. Cortes, C., Vapnik, V.: Support-vector networks. Machine Learning 20(3), 273–297 (1995)
7. Denker, A., Atherton, D.: No-overshoot control of robotic manipulators in the presence of obstacles. J. Robotic Systems 11(7), 665–678 (1994)
8. Eiben, A.E., Smith, J.E.: Introduction to Evolutionary Computing. Springer, Berlin, Germany (2003)
9. Floreano, D., Mitri, S., Perez-Uribe, A., Keller, L.: Evolution of altruistic robots. In: Proceedings of the WCCI 2008. vol. 5050, pp. 232–248. Springer Berlin / Heidelberg (2008)
10. Glasmachers, T., Schaul, T., Schmidhuber, J.: A Natural Evolution Strategy for Multi-Objective Optimization. In: Parallel Problem Solving from Nature (PPSN) (2010)
11. Glasmachers, T., Schaul, T., Sun, Y., Wierstra, D., Schmidhuber, J.: Exponential Natural Evolution Strategies. In: Genetic and Evolutionary Computation Conference (GECCO). Portland, OR (2010)
12. Gomez, F., Schmidhuber, J., Miikkulainen, R.: Efficient non-linear control through neuroevolution. In: Fürnkranz, J., Scheffer, T., Spiliopoulou, M. (eds.) Proceeding of the European Conference on Machine Learning. pp. 654–662. No. 4212 in LNAI, Springer (2006)
13. Hansen, N., Ostermeier, A.: Completely derandomized self-adaptation in evolution strategies. Evolutionary Computation 9(2), 159–195 (2001)
14. Harding, S., Miller, J.F.: Evolution of Robot Controller Using Cartesian Genetic Programming. Genetic Programming pp. 62–73 (2005)
15. Hayashi, A.: Geometrical Motion Planning For Highly Redundant Manipulators Using A Continuous Model. Ph.D. thesis, University of Texas Austin (1994)
16. Iossifidis, I., Schöner, G.: Dynamical Systems Approach for the Autonomous Avoidance of Obstacles and Joint-limits for an Redundant Robot Arm. In: IEEE/RSJ International Conference on Intelligent Robots and Systems. pp. 580–585 (2006)
17. Kavraki, L., Svestka, P., claude Latombe, J., Overmars, M.: Probabilistic roadmaps for path planning in high-dimensional configuration spaces. In: IEEE International Conference on Robotics and Automation. pp. 566–580 (1996)

18. Khatib, O.: Real-time obstacle avoidance for manipulators and mobile robots. International Journal of Robotic Research 5(1), 90–98 (1986)
19. Koutník, J., Gomez, F.J., Schmidhuber, J.: Evolving neural networks in compressed weight space. In: GECCO. pp. 619–626 (2010)
20. Latombe, J.C.: Robot Motion Planning. Kluwer Academic Publishers, Norwell, MA, USA (1991)
21. Lee, J.D., Wang, B.L.: Optimal control of a flexible robot arm. Computers Ŝtructures 29(3), 459 – 467 (1988)
22. Mitrovic D, Klanke S, V.S.: Adaptive optimal feedback control with learned internal dynamics models. In: Sigaud, O., Peters, J. (eds.) From Motor Learning to Interaction Learning in Robots, pp. 65–84. Springer Berlin / Heidelberg (2010)
23. Nolfi, S., Marocco, D.: Evolving robots able to integrate sensory-motor information over time. Theory in Biosciences 120:, 287–310 (2001)
24. Rechenberg, I., Eigen, M.: Evolutionsstrategie: Optimierung technischer Systeme nach Prinzipien der biologischen Evolution. Frommann-Holzboog Stuttgart (1973)
25. Schaul, T., Glasmachers, T., Schmidhuber, J.: High Dimensions and Heavy Tails for Natural Evolution Strategies (2011), under review
26. Scholkopf, B., Smola, A., Muller, K.R.: Nonlinear Component Analysis as a Kernel Eigenvalue Problem. Neural Comp. 10(5), 1299–1319 (Jul 1998)
27. Sun, Y., Wierstra, D., Schaul, T., Schmidhuber, J.: Efficient Natural Evolution Strategies. In: Genetic and Evolutionary Computation Conference (GECCO) (2009)
28. Sun, Y., Wierstra, D., Schaul, T., Schmidhuber, J.: Stochastic Search using the Natural Gradient. In: International Conference on Machine Learning (ICML) (2009)
29. Vapnik, V.: The support vector method. In: ICANN. pp. 263–271 (1997)
30. Vapnik, V.N.: Statistical Learning Theory. Wiley-Interscience (Sep 1998)
31. Wierstra, D., Schaul, T., Peters, J., Schmidhuber, J.: Natural Evolution Strategies. In: Proceedings of the Congress on Evolutionary Computation (CEC08), Hongkong. IEEE Press (2008)
32. Woolley, B.G., Stanley, K.O.: Evolving a single scalable controller for an octopus arm with a variable number of segments. In: PPSN (2). pp. 270–279 (2010)
33. Yekutieli, Y., Sagiv-Zohar, R., Aharonov, R., Engel, Y., Hochner, B., Flash, T.: A dynamic model of the octopus arm. I. biomechanics of the octopus reaching movement. Journal of Neurophysiology 94(2), 1443–1458 (2005)