Technische Universiteit
Eindhoven
University of Technology

Department of Electrical Engineering

Den Dolech 2, 5612 AZ Eindhoven
P.O. Box 513, 5600 MB Eindhoven
The Netherlands
http://w3.ele.tue.nl/nl/

Series title:
Master graduation paper,
Electrical Engineering

Commissioned by Professor:

Group / Chair:

Date of final presentation:

Report number:

by

Author:

Computer-Aided Algorithm Design for Audio Processing

J. Kraak, M.Sc. Thesis
Signal Processing Systems group, Eindhoven University of Technology
e-mail: j.kraak@student.tue.nl

***Abstract*—Audio processing algorithms are typically complex systems consisting of multiple, parametrized subsystems. Although the behavior of the individual subsystems is often well known, the interaction between them is usually not. To combat this, design processes have been described that try to solve this problem by optimizing over the parameter space of the complete system. However these processes have to optimize over large design spaces and quickly become intractable, especially for expensive loss functions. In this work we investigate the applicability of a robust, Gaussian process based optimization process to this kind of optimization problems. Through a design example, where we optimize the performance of a noise reduction algorithm, we show that it is possible to achieve speed ups of up to an order of magnitude with respect to more traditional optimization methods.**

***Index Terms*—Bayesian optimization, Design optimization, Gaussian processes, Audio processing**

I. Introduction

AUDIO processing algorithms often aim to solve multiple problems at the same time. For instance, in a hearing instrument algorithm we can identify, amongst others, subsystems for amplification, compression, feedback cancellation, noise reduction and beamforming. Although most of these subalgorithms are well understood, they each have their own parameters and performance measures, adding to the complexity of the complete system. On top of that, the interaction between the different algorithms is usually not understood as well as the subsystems themselves.

The design of such complex algorithms is usually performed using an iterative approach based on hypothesis testing to come to some optimal solution with respect to a given performance measure. Hypothesis testing in this case means defining some hypothesis on the performance of an alternative algorithm with respect to a reference algorithm and verifying whether this hypothesis is satisfied. This hypothesis testing is then done iteratively, replacing the reference algorithm by the alternative algorithm whenever the latter performs better. Alternative algorithms can be created by, for instance, adding or removing components or by tuning parameters of components. Although this approach will typically result in satisfactory results, it has a number of inherent problems, as described by Hoos in [1], that can cause these results to be suboptimal:

1) The iterative approach is inherently labor-intensive, which results in researchers only evaluating a relatively small subset of possible algorithm designs within a large design space. To make matters worse, this subset tends to be chosen in an *ad-hoc* manner instead of through a carefully planned procedure.
2) To keep the design manageable, an algorithm will typically be split into smaller subsystems. Which are then optimized individually for a particular benchmark problem. This approach assumes that the performance of the individual subsystems is independent, which is not true in general. In a worst-case scenario the optimal subsystems can negatively affect each other when combined, resulting in suboptimal overall performance.
3) Human designers have a tendency to overgeneralize conclusions drawn within the limited context of a subsystem. These can be exaggerated by *a priori* expectations of a certain pattern or regularity, which results in a risk of overestimating the value of supporting evidence versus underestimating evidence to the contrary.
4) The iterative design approach has a tendency to focus on creating increasingly more complex systems over time. Due to personal investment, people typically do not like to remove functionality they have previously added, even if the benefits of that functionality are negligible. As before, the possibility exists that different solutions counteract each others performance.

In short we can conclude that, although the iterative design approach based on hypothesis testing usually produces satisfactory results, its labor intensive nature leads to suboptimal designs for larger systems due to assumed independence of subsystems and the tendency of these subsystems to become increasingly more complex as a result of personal investment of their human designers.

In an effort to overcome these problems, Hoos describes an automated approach to the algorithm design process called *computer-aided algorithm design* (CAAD), [1], [2]. This approach assigns the tasks that resulted in the problems mentioned above to an automatic optimization process, leaving the design of the search space for the process to the researcher.

In this thesis, we have begun to investigate the automated design process described by Hoos in the context of designing an audio processing algorithm. We start by mathematically describing this design process in the form of an optimization problem and show that this problem is difficult to solve for expensive loss functions. Next we identify a number of possible solutions in the literature that could be employed to solve the optimization (and thus the design) problem more efficiently using metamodels. We conclude with an analysis of

the implementations of these solutions and give an example of formulating a specific audio processing problem in such a way that it becomes solvable using the described techniques. The described optimization process using Gaussian processes can not be used as a black box optimization tool due to constraints it imposes on the description of the optimization problem. Using the design example, we show that, for problems formulated within its constraints, the optimization process gives a significant performance boost in finding optimal parameters of an algorithm by requiring an order of magnitude less evaluations of the loss function to reach the optimum. Unfortunately the optimization process can become hampered by numerical issues and its results should be handled with care.

II. Problem Definition

We describe an audio processing algorithm (APA) as a dynamical system by

$$s_{t+1} = H\left(\boldsymbol{\theta}, x_t, s_t\right), \tag{1}$$

$$y_t = G(s_t) \tag{2}$$

where H is the state update function, G is a mapping from the system state to the output of the system, $\boldsymbol{\theta} \in \Theta$ is a vector of parameters of the state update function and x_t, y_t and s_t are respectively an acoustic input signal, the corresponding acoustic output signal and the system state at time t.

The performance of the APA for a given input signal $x = [x_1, \ldots, x_T]$ is measured using a performance function. This performance function has access to both the final state of the system $s_f(\boldsymbol{\theta}, x)$ as well as a corresponding reference (or desired) state $r_f(x)$. For the purposes of our discussion this performance function is taken to be a scalar loss function defined as

$$L(s_f(\boldsymbol{\theta}, x), r_f(x)). \tag{3}$$

We will assume that the structure of H and the construction of r_f are not optimizable by a computer and are therefore chosen to be fixed, effectively making the loss function dependent only on $\boldsymbol{\theta}$ and x. Furthermore, instead of trying to obtain optimal parameters $\boldsymbol{\theta}^*$ for every possible x, we would like to find $\boldsymbol{\theta}^*$ which is on average optimal for a set of problem instances x_k. This set is denoted by $\mathcal{X}$, with a probability distribution $p(\mathcal{X})$ defined over it which assigns relevance weights to each of the problem instances. Combining this idea with (3) we can create an *expected loss function*, which is defined as

$$\mathrm{EL}\left(\boldsymbol{\theta}\right) = \sum_{k=1}^{K} p(x_k) L(\boldsymbol{\theta}, x_k). \tag{4}$$

Minimization of (4) leads to the optimal parameters $\boldsymbol{\theta}^*$, defined by

$$\boldsymbol{\theta}^* = \arg\min_{\boldsymbol{\theta}} \mathrm{EL}\left(\boldsymbol{\theta}\right). \tag{5}$$

This type of problem formulation has been shown to have applications in, amongst others, personalization of medical devices [3]. Furthermore this formulation can also be used to combine multiple loss or utility functions as proposed by Tashev *et al.* [4].

By describing the process of designing an APA as solving (5), we arrive at what we call computer-aided algorithm design for audio processing algorithms. In this framework an optimization algorithm is tasked with solving (5), whereas the tasks of a human researcher are reduced to designing:

1) the structure of the algorithm H;
2) a (possibly very large) design space Θ describing parameters of H;
3) a set of problem instances $\mathcal{X}$ (with probability density $p(\mathcal{X})$ and corresponding reference states $r_f(x)$);
4) a loss function L to be used as a performance measure.

An interesting feature of these tasks is that they can be altered quickly to accommodate changing requirements to the algorithm.

The main problem with (5) is that L can be computationally expensive. Taking into consideration that we would like to evaluate L for a large space Θ and a large set of problem instances $\mathcal{X}$, we can show by an example that solving (5) quickly becomes intractable using exhaustive optimization techniques. Take into consideration an APA that can be simulated in real-time. To accurately represent the input signals the algorithm should be able to handle, we might need 100 problem instances of 3 seconds each (which would be an appropriate number for doing, for instance, speech in noise analysis). In this case a single evaluation of (4) would take approximately 5 minutes. Assuming we would like to evaluate 5 interesting values for a single parameter (ranging from a low to a high value), calculating (4) would take 25 minutes. For each added parameter this simulation time would grow exponentially, from which we can see that solving (5) using exhaustive optimization techniques is simply not feasible for practical numbers of parameters.

Taking the illustrated exponential growth in evaluation time of (4) into account, the desire to be able to do computer-aided algorithm design for audio-processing algorithms leads us to define the problem to be solved in this work as *finding an optimization algorithm that can solve* (5) *adequately in as few evaluations of* (3) *as possible.*

III. Methods for Solving the Optimization Problem

A. Introduction to the proposed approach

Before discussing possible approaches for solving (5), let us first try to visualize our desired solution strategy using a toy example for (3). This toy example is a slightly modified version of the *six hump camelback* function introduced by Dixon and Szegö [5]. This is a two-dimensional function, which is widely used to test global optimization algorithms. It has a total of 6 minima (2 equal global minima located at $(-0.7126, 0.0898)$ and $(0.7126, -0.0898)$ and 4 local minima). To be able to more easily demonstrate the effect of defining a probability density along one of the dimensions when calculating (4), we modify the function in such a way that its response is rotated over $90°$. A contour plot of this modified function, along with the location of the global optima, is shown in Fig. 1.

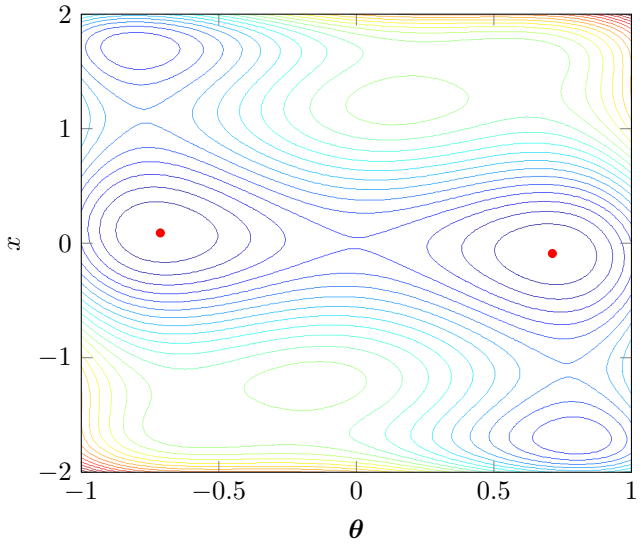

Fig. 1. A contour plot of the modified six hump camelback function representing (3). The two equal, global optima of the function are shown as ●.

Introducing a Gaussian probability distribution, $\mathcal{N}(\mu, \sigma^2)$, over x such that $p(x) \sim \mathcal{N}(0.5, 0.01)$, we can visualize what (4) looks like for this toy example. Since the chosen probability distribution is rather narrow, the values around $x = 0.5$ are assigned a much higher weight than the ones further away. As a consequence we expect the optimum at $(-0.7126, 0.0898)$ to be more explicitly visible in a plot of (4) than the one at $(0.7126, -0.0898)$. From Fig. 2 can be seen that this is indeed the case.

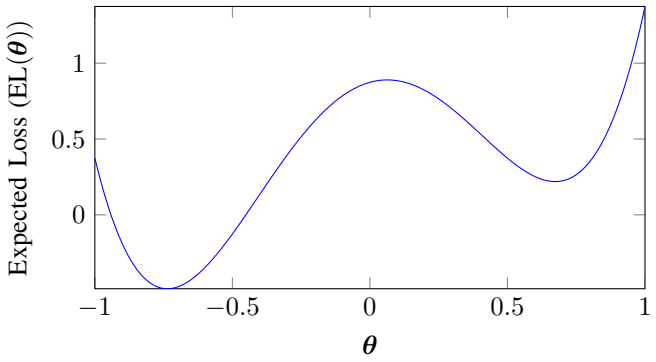

Fig. 2. The expected loss function EL when using the modified six hump camelback function for the loss function L and letting $p(X) \sim \mathcal{N}(0.5, 0.01)$.

From the illustration using our toy example we can devise a general strategy for our optimization process. As mentioned at the end of Section II, our goal is to find the optimum of EL($\boldsymbol{\theta}$) in as few evaluations of $L(\boldsymbol{\theta}, x)$ as possible. Using Figs. 1 and 2, this can be reformulated as that the optimization process should be able to infer an accurate approximation of Fig. 2 using as few points sampled from Fig. 1 as possible. Assuming the optimization process can obtain an accurate model $\widehat{EL}(\boldsymbol{\theta})$ of (4), it can use this model to estimate the optimum of (4) instead of the true function. Additionally the model can be used in an iterative optimization process to determine interesting points to evaluate $L(\boldsymbol{\theta}, x)$ at, thereby improving the accuracy of the model in each iteration until some stopping criterion has been met. The steps of the proposed optimization strategy are visualized in Fig. 3.

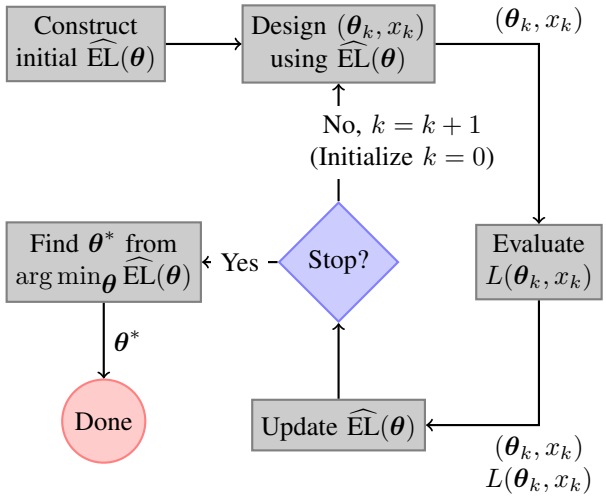

Fig. 3. Visualization of the proposed iterative optimization strategy.

B. Gaussian Processes

An optimization method that fits our desire of using models to approximate the response of the loss function is Efficient Global Optimization (EGO) by Jones *et al.* [6]. Since EGO uses Gaussian Processes to construct this model, we will first take a short look at how GPs can be used for modelling and optimization. More extensive treatises on this subject can be found in [7]–[10].

A GP is defined in [7] by a probability distribution over functions $y(\mathbf{x})$, such that the values of $y(\mathbf{x})$ evaluated at any arbitrary set of points $\{\mathbf{x}_1, \ldots, \mathbf{x}_N\}$ jointly have a Gaussian distribution. We can get an intuitive feel for this by looking at an example.

We start from a linear regression model defined as

$$y(\mathbf{x}) = \mathbf{w}^{\mathrm{T}} \phi(\mathbf{x}), \tag{6}$$

where $\mathbf{x}$ is the input vector, $\mathbf{w}$ is an M-dimensional weight vector and $\phi(\mathbf{x})$ is a vector of M fixed basis functions. Furthermore we assume a prior distribution over $\mathbf{w}$, governed by a hyperparameter ϑ, given by

$$p(\mathbf{w}) = \mathcal{N}\left(\mathbf{w} | \mathbf{0}, \vartheta^{-1}\mathbf{I}\right), \tag{7}$$

where $\mathcal{N}$ denotes the multivariate Gaussian distribution. For any given value of $\mathbf{w}$, (6) defines a specific function of $\mathbf{x}$. The probability distribution given by (7) therefore induces a probability distribution over functions $y(\mathbf{x})$.

In practice we want to evaluate $y(\mathbf{x})$ only for specific values of $\mathbf{x}$, for instance at training points $\mathbf{x}_1, \ldots, \mathbf{x}_N$. From our initial definition of a GP, we are interested in the joint distribution of the function values $y(\mathbf{x}_1), \ldots, y(\mathbf{x}_N)$, which we can collect in a vector $\mathbf{y}$. Using (6), $\mathbf{y}$ is given by

$$\mathbf{y} = \mathbf{\Phi}\mathbf{w}, \tag{8}$$

where $\mathbf{\Phi}$ is a matrix with elements $\Phi_{ij} = \phi_j(\mathbf{x}_i)$.

We can find the probability distribution over $\mathbf{y}$ by noting that it is a linear combination of Gaussian distributed variables, given by the elements of $\mathbf{w}$, and that it therefore is itself Gaussian distributed. Given that $\mathbf{y}$ is Gaussian distributed we only need to find the mean and covariance of the distribution, which we can derive from (7) and (8) as follows

$$\mathbb{E}[\mathbf{y}] = \mathbf{\Phi}\mathbb{E}[\mathbf{w}] = \mathbf{0}, \tag{9}$$

$$\text{cov}[\mathbf{y}] = \mathbb{E}[\mathbf{y}\mathbf{y}^{\mathrm{T}}] = \mathbf{\Phi}\mathbb{E}[\mathbf{w}\mathbf{w}^{\mathrm{T}}]\mathbf{\Phi}^{\mathrm{T}} = \frac{1}{\vartheta}\mathbf{\Phi}\mathbf{\Phi}^{\mathrm{T}} = \mathbf{K}, \tag{10}$$

where $\mathbf{K}$ is a Gram matrix with elements

$$K_{ij} = k(\mathbf{x}_i, \mathbf{x}_j) = \frac{1}{\vartheta}\phi(\mathbf{x}_i)^{\mathrm{T}}\phi(\mathbf{x}_j), \tag{11}$$

and $k(\mathbf{x}, \mathbf{x}')$ a kernel function.

When using GPs, it is typically the kernel function that is selected from which the basis functions follow. Popular choices for kernel functions are the polynomial, squared exponential and Matern kernels. More complex kernels can be constructed from existing kernels as described in [7].

From this example we have seen that it is possible to define a probability distribution over functions using GPs. The type of functions that can be sampled from these probability distributions is dependent on the kernel function $k(\mathbf{x}, \mathbf{x}')$ and its parameters, which are effectively the hyperparameters of the distribution. Fig. 4 shows several example functions drawn from a GP.

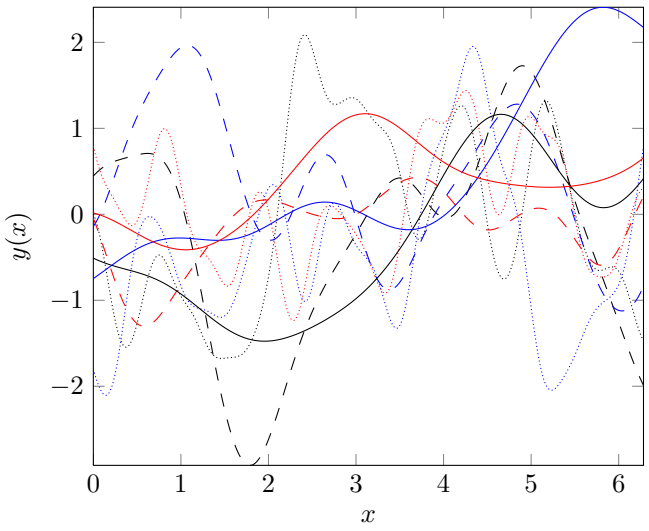

Fig. 4. Example functions drawn from a Gaussian process $Y(x) \sim \mathcal{N}(\mathbf{0}, \mathbf{K})$, with $K_{ij} = k(\mathbf{x}_i, \mathbf{x}_j) = \exp\left(-\frac{\|\mathbf{x}_i - \mathbf{x}_j\|^2}{2\vartheta_0^2}\right)$ for $\vartheta_0 = 2$ (——), $\vartheta_0 = 0.5$ (− −) and $\vartheta_0 = 0.25$ (·····).

C. Predictive Distributions for Unobserved Data Points

Now that we have a notion of probability distributions over functions in the form of Gaussian processes, we can examine how optimization methods like EGO use GPs as metamodels for the response of a simulator which is to be optimized.

These methods define a predictive distribution for response values at unobserved datapoints, given a set of training points $\mathbf{X}_N = \{\mathbf{x}_1, \ldots \mathbf{x}_N\}$, the corresponding response values $y(\mathbf{x})$ collected in a vector $\mathbf{y}_N$ and a covariance kernel $k(\mathbf{x}, \mathbf{x}')$.

From the definition of a GP we know that the marginal distribution of $\mathbf{y}_N$ is given by

$$p(\mathbf{y}_N) = \mathcal{N}(\mathbf{y}_N|\mathbf{0}, \mathbf{C}_N), \tag{12}$$

where $\mathbf{C}_N$ is an $N \times N$ covariance matrix with elements

$$C_{ij} = k(\mathbf{x}_i, \mathbf{x}_j) \tag{13}$$

for $i, j = 1, \ldots, N$.

To be able to make predictions about the response y_{N+1} for a new input $\mathbf{x}_{N+1}$ we need to evaluate the predictive distribution $p(y_{N+1}|\mathbf{y}_N)$, where we have removed the conditioning on $\mathbf{X}_N$ and $\mathbf{x}_{N+1}$ for notational convenience. As shown in [7], this predictive conditional distribution $p(y_{N+1}|\mathbf{y}_N) \sim \mathcal{N}(\hat{y}, \sigma_y^2)$ is a Gaussian distribution with

$$\hat{y}(\mathbf{x}_{N+1}) = \mathbf{k}^{\mathrm{T}}\mathbf{C}_N^{-1}\mathbf{y}_N, \tag{14}$$

$$\sigma_y^2(\mathbf{x}_{N+1}) = v - \mathbf{k}^{\mathrm{T}}\mathbf{C}_N^{-1}\mathbf{k}, \tag{15}$$

where $\mathbf{C}_N$ is the covariance matrix from (12), the vector $\mathbf{k} = k(\mathbf{x}_i, \mathbf{x}_{N+1})$ for $i = 1, \ldots, N$ is the covariance between the training points and the test point and the scalar $v = k(\mathbf{x}_{N+1}, \mathbf{x}_{N+1})$.

In other words, given $\mathbf{X}_N, \mathbf{y}_N$ and $\mathbf{x}_{N+1}$, we can estimate the corresponding response y_{N+1} by (14) and the uncertainty about that response by (15).

D. Bayesian Optimization Using Gaussian Processes

In any given iteration of the optimization procedure, methods like EGO evaluate the predictive distribution derived in Section III-C for a large number of test points to determine at which point to sample the simulator in the next iteration. After evaluating the simulator at the desired point, the selected point and the obtained response are then used to update the covariance matrices of the predictive distribution for the following iteration.

The decision on what point to evaluate the simulator at is typically dictated by an experimental design function, which is also known as the *acquisition function*. Through proper construction of these acquisition functions, it is possible to switch these optimization methods from a mode with the focus on exploitation, obtaining a better estimate of the current optimum, to a mode focused on exploration of the solution space, depending on the need of the optimization procedure.

An example of an acquisition function which is targeted mostly on exploitation is $\alpha(\mathbf{x}) = \hat{y}(\mathbf{x}) - 2\sigma_y(\mathbf{x})$. In the case of a minimization problem the optimizer would select the point of interest as the one for which $\alpha(\mathbf{x})$ is minimal. After repeatedly evaluating the simulator and obtaining new points, this strategy should lead to the actual minimum of the simulator. Figure 5 shows an example of Bayesian optimization using GPs for finding the minimum of a function using this particular strategy.

The benefit of using Gaussian processes for Bayesian optimization is that they provide the acquisition function of an optimization procedure with a way to measure how good a newly added point might be, since an estimate of the mean

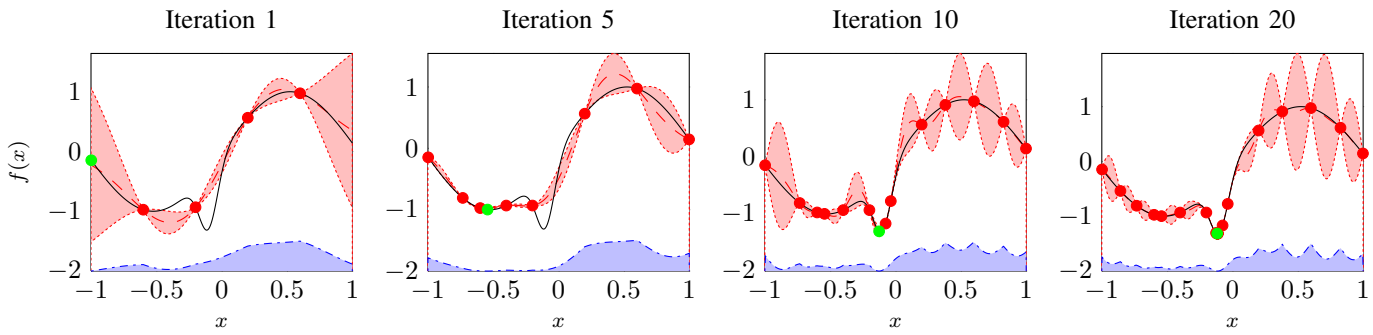

Fig. 5. Iterations 1, 5, 10 and 20 from a minimization problem solved through Bayesian optimization using GPs. In each iteration the point of interest is selected according to $x_{N+1} = \arg\min_x [\hat{y}(x) - 2\sigma_y(x)|\mathbf{x}_N]$. The response of the simulator is shown as ——, $\hat{y}(x)$ as – –, $\hat{y}(x) \pm 2\sigma_y(x)$ as ·····, the training points as ●, the acquisition function, which has been scaled between 0 and 0.5 and offset by -2 for clarity, as ·-·-· and the point at which the simulator will be evaluated in the next iteration is shown as ●.

and variance at each unobserved point is known. This means that for a properly selected acquisition function, it is possible to do optimal experimental design in each iteration of the optimization process.

E. Robust Efficient Global Optimization

In the previous section we have described the EGO optimization procedure that can be used to find an optimal solution in the $(\Theta, \mathbf{x})$-space. As mentioned however, we are interested in the optimal solution $\boldsymbol{\theta}^*$ of (5), which can not be calculated directly using this optimization procedure. It might be possible to find $\boldsymbol{\theta}^*$ from the model EGO builds of the simulator, but this would still require a rather detailed model $\hat{L}(\boldsymbol{\theta}, x)$ only available after evaluation of $L(\boldsymbol{\theta}, x)$ for a large number of x. Solving this problem through Bayesian Monte Carlo has been previously investigated by Groot *et al.* [11]. In this section we will discuss an extension to EGO, which fits the profile of our ideal optimizer shown in Fig. 3 and is able to directly solve (5).

This extension is described by Janusevskis *et al.*, who use it to build GPs for noisy inputs [12]. The problem they solve is very similar to (4), hence the method appears to be a good fit for our ideal optimizer. The main strength of the algorithm is that it allows sampling a single point straight in the (Θ, X)-space to obtain an estimate of (4), instead of having to evaluate (3) for a number of points x_k to obtain that same estimate. In this section we will summarize how this algorithm works and will refer to it as Robust Efficient Global Optimization (REGO) from here on.

The optimization problem that Janusevskis *et al.* treat is a minimization problem. They recognize that it is possible to split the inputs to the simulator into two sets of parameters: deterministic and random. This causes them to write the optimization problem as

$$\boldsymbol{\theta}^* = \arg\min_{\boldsymbol{\theta}} \int L(\boldsymbol{\theta}, \mathbf{x}) p(\mathbf{x}) \, \mathrm{d}\mathbf{x}, \tag{16}$$

where $\boldsymbol{\theta} \in \mathbb{R}^n$ are deterministic parameters and $\mathbf{x} \in \mathbb{R}^m$ are instances of a multivariate random variable X with a given joint probability distribution $p(X)$. We recognize this problem to be similar to (5), albeit that the random variable is continuous, whereas in our problem we take the expectation over a set of discrete variables. This problem can be overcome by proper definition of our design space $\mathcal{X}$ as we will show later.

Using the theory discussed in Section III-B and assuming that (3) can properly be described by a GP, $L(\boldsymbol{\theta}, \mathbf{x})$ can be approximated by a Gaussian process $Y(\boldsymbol{\theta}, \mathbf{x})$. Assuming we have a number of training points available to train Y, it can be used to predict the mean and variance at any unobserved point in the joint (Θ, X)-space. As mentioned in Section III-D we could use this GP to do optimization in this joint space. However to find $\boldsymbol{\theta}^*$ we would still have to marginalize out the random variables. Instead of doing this, Janusevskis *et al.* propose the construction of a second GP $Z(\boldsymbol{\theta})$, which they call the *projected process*, by taking the expectation of Y with respect to X, thereby marginalizing out the random part of the input space to the simulator. In other words $Z(\boldsymbol{\theta})$ is a model of EL$(\boldsymbol{\theta})$. This new process Z is also a GP since any linear transformation of a multivariate Gaussian distribution is again a multivariate Gaussian distribution.

The predictive distribution of the new GP Z can be described by scaled elements of the predictive distribution of Y as given by (14) and (15). This new predictive distribution can be analytically derived under the condition that samples from X are independently Gaussian distributed and that the kernel used for the covariance matrix of the GP is a multivariate squared exponential kernel defined as

$$k(\boldsymbol{\xi}_i, \boldsymbol{\xi}_j) = \exp\left(-\frac{1}{2}(\boldsymbol{\xi}_i - \boldsymbol{\xi}_j)^{\mathrm{T}} \mathrm{diag}(\boldsymbol{\vartheta}^{-1})(\boldsymbol{\xi}_i - \boldsymbol{\xi}_j)\right), \tag{17}$$

where $\boldsymbol{\xi} \in \mathbb{R}^p$ and $\boldsymbol{\vartheta}$ is a vector with elements ϑ_i for $i = 1, \ldots, p$ governing the prior distribution over each dimension of $\boldsymbol{\xi}$.

When these conditions are satisfied, the predictive distribution of the projected process Z is very similar to (14) and (15) and given by $p_Z(y_{N+1}|\mathbf{y}_N) \sim \mathcal{N}(\hat{z}, \sigma_z^2)$ with

$$\hat{z}(\boldsymbol{\theta}_{N+1}) = (\boldsymbol{\gamma} \circ \mathbf{k})^{\mathrm{T}} \mathbf{C}_N^{-1} \mathbf{y}_N, \tag{18}$$

$$\sigma_z^2(\boldsymbol{\theta}_{N+1}) = \psi \cdot v - (\boldsymbol{\gamma} \circ \mathbf{k})^{\mathrm{T}} \mathbf{C}_N^{-1} (\boldsymbol{\gamma} \circ \mathbf{k}), \tag{19}$$

where $\mathbf{y}_N$ are the known response values, $v = k(\boldsymbol{\theta}_{N+1}, \boldsymbol{\theta}_{N+1})$, $\mathbf{k} = k(\boldsymbol{\theta}_{N+1}, \boldsymbol{\theta}_i)$ and $\mathbf{C}_N = k(\boldsymbol{\theta}_i, \boldsymbol{\theta}_j)$

with $\boldsymbol{\theta} \in \mathbb{R}^n$, k the squared exponential kernel given by (17) for $i, j = 1, \ldots, N$ and $\circ$ denotes the entrywise product.

For the hyperparameters $\boldsymbol{\vartheta}$ of the squared exponential kernel k, used for calculating $v, \mathbf{k}$ and $\mathbf{C}_N$, only those parameters governing the prior distribution over the deterministic input space are used. These hyperparameters are collected in a vector $\boldsymbol{\vartheta}_d \in \mathbb{R}^n$. The hyperparameters governing the prior distribution over the random input space are collected in a vector $\boldsymbol{\vartheta}_r \in \mathbb{R}^m$ and will be used later in the derivation. The vector $\boldsymbol{\gamma} \in \mathbb{R}^N$ represents the uncertainty for each of the training points in the random input space with elements given by

$$\gamma_i = \frac{|\mathbf{D}|^{1/2}}{|\boldsymbol{\Sigma} + \mathbf{D}|^{1/2}} \cdot \exp\left(\frac{1}{2}(\mathbf{x}_i - \mathbf{x}_\mu)^{\mathrm{T}}(\boldsymbol{\Sigma} + \mathbf{D})^{-1}(\mathbf{x}_i - \mathbf{x}_\mu)\right), \quad (20)$$

where $|\cdot|$ denotes the determinant, $\mathbf{x}_i \in \mathbb{R}^m$ is the ith training point in the random input space, $\mathbf{x}_\mu \in \mathbb{R}^m$ is the mean of the prior distribution over the random input space, $\boldsymbol{\Sigma}$ is a diagonal matrix containing the corresponding prior variance σ_i^2 for $i = 1, \ldots, m$ and $\mathbf{D}$ is a diagonal matrix containing $\boldsymbol{\vartheta}_r$. Finally ψ is the corresponding uncertainty on the new point $\boldsymbol{\theta}_{N+1}$ and given by

$$\psi = \frac{|\mathbf{D}|^{1/2}}{|2\boldsymbol{\Sigma} + \mathbf{D}|^{1/2}}. \quad (21)$$

For a full derivation of these equations we refer the interested reader to [12].

Apart from the extra steps required for calculating the predictive distribution of the projected process, the optimization process is very similar to the one for deterministic inputs as described in Section III-D. A final difference is in the acquisition function, which consists of two separate steps to determine the next sample point in the joint (Θ, X)-space.

The first step of the acquisition function determines the most promising point $\boldsymbol{\theta}_{N+1}$ in the deterministic Θ-space. It does so by selecting the point $\boldsymbol{\theta}$ that maximizes what is known as the Expected Improvement (EI). The EI is an example of an acquisition function that balances between exploitation and exploration. It is defined as

$$\mathrm{EI}_Z(\boldsymbol{\theta}) = \sigma_z^2(\boldsymbol{\theta})\left[u(\boldsymbol{\theta})\Phi(u(\boldsymbol{\theta})) + \phi(u(\boldsymbol{\theta}))\right], \quad (22)$$

where

$$u(\boldsymbol{\theta}) = \frac{\hat{z}_{\min} - \hat{z}(\boldsymbol{\theta})}{\sigma_z^2(\boldsymbol{\theta})}, \quad (23)$$

with $\hat{z}_{\min} = \min_{\boldsymbol{\theta}} \hat{z}(\boldsymbol{\theta})$ and Φ and ϕ respectively the Gaussian cumulative distribution and probability density functions. The candidate point $\boldsymbol{\theta}_{N+1}$ in deterministic space is then obtained from

$$\boldsymbol{\theta}_{N+1} = \arg\max_{\boldsymbol{\theta}} \mathrm{EI}_Z(\boldsymbol{\theta}) \quad (24)$$

The second step in the acquisition function obtains the point to sample at in the probabilistic X space. For this step [12] proposes two different approaches.

The first approach selects the point $(\boldsymbol{\theta}_{N+1}, \mathbf{x}_{N+1})$ in the joint (Θ, X)-space that minimizes the expected variance of the projected process in the following optimization iteration. Although this is the more accurate approach, it is also more computationally involved, amongst others due to Cholesky decompositions in the optimization process.

The second approach selects the point $\boldsymbol{\theta}_{N+1}$ as the next point in the Θ space and picks a point

$$\mathbf{x}_{N+1} \sim p(X) \quad (25)$$

to sample at in the random input space X.

Since we are mostly interested in the feasibility of efficiently solving (5) using the techniques described, we will use the second approach. The more accurate approach can always be implemented at a later time when the additional accuracy is required.

After obtaining the point $(\boldsymbol{\theta}_{N+1}, \mathbf{x}_{N+1})$, the simulator is evaluated and the result is added to the set of training points for Y. The process then starts over again until the stopping criterion of the optimization algorithm is met. The solution to (5) can be generated at any time during the optimization process with the certainty as described by the variance of the projected process at the returned point. The stopping criterion of the optimization process is taken to be either the exhaustion of the iteration budget set for the optimization process or whenever the EI falls below a selected threshold, whichever comes first.

After the stopping criterion has been met, the optimal parameters for the algorithm under study H are given by

$$\boldsymbol{\theta}^* = \arg\min_{\boldsymbol{\theta}} \hat{z}(\boldsymbol{\theta}). \quad (26)$$

To complete the algorithm for solving (5) we need one final element. So far we have assumed that the hyperparameters of the kernel function $k(\mathbf{x}, \mathbf{x}')$ are known, this does however not need to be the case. When given a set of training data it is possible to obtain optimal hyperparameters $\boldsymbol{\vartheta}^*$ for given $k(\mathbf{x}, \mathbf{x}')$ through a training process. Optimal in this sense meaning that the *evidence* for the training data is maximal for $\boldsymbol{\vartheta}^*$. Mathematically, this evidence is given by the marginal log likelihood of the training data under the Gaussian process model with hyperparameters $\boldsymbol{\vartheta}^*$, defined as

$$\log p(\mathbf{y}_N|\mathbf{x}_1, \ldots, \mathbf{x}_N, \boldsymbol{\vartheta}) = -\frac{1}{2}\log\det(\mathbf{C}_N) - \frac{1}{2}\mathbf{y}_N^{\mathrm{T}}\mathbf{C}_N^{-1}\mathbf{y}_N - \frac{N}{2}\log 2\pi, \quad (27)$$

with $\mathbf{C}_N$ calculated in the same way as for (18) and (19). Using this definition, the training process can be described as

$$\boldsymbol{\vartheta}^* = \arg\max_{\boldsymbol{\vartheta}} \log p(\mathbf{y}_N|\mathbf{x}_1, \ldots, \mathbf{x}_N, \boldsymbol{\vartheta}). \quad (28)$$

Now that all elements are in place, the final algorithm for solving (5) is outlined in Table I.

We should note that this algorithm for solving the optimization of our expensive simulator actually contains three new optimization problems for finding:

- the minimum of $\hat{z}$ as given by (26), which is used for solving (24) as well,
- the maximum of the expected improvement as given by (24), and

TABLE I
THE REGO ALGORITHM

1: Collect initial training points $\mathbf{X} = \{(\boldsymbol{\theta}_1, \mathbf{x}_1), \ldots, (\boldsymbol{\theta}_N, \mathbf{x}_N)\}$
2: Collect response values $\mathbf{L} = \{L(\boldsymbol{\theta}_1, \mathbf{x}_1), \ldots, L(\boldsymbol{\theta}_N, \mathbf{x}_N)\}$
3: Create a GP with kernel $k(\mathbf{x}, \mathbf{x}', \boldsymbol{\vartheta})$ as defined in (17)
4: **while** stopping criterion not met **do**
5: Find $\boldsymbol{\vartheta}^*$ using the selected $k(\mathbf{x}, \mathbf{x}', \boldsymbol{\vartheta})$, $\mathbf{X}$, $\mathbf{L}$ and (28)
6: Determine $\hat{z}(\boldsymbol{\theta})$ and $\sigma_z^2(\boldsymbol{\theta})$ using (18) and (19)
7: Obtain point $\boldsymbol{\theta}_{N+1}$ in deterministic input space from (24)
8: Obtain point $\mathbf{x}_{N+1}$ in probabilistic input space from (25)
9: Add $(\boldsymbol{\theta}_{N+1}, \mathbf{x}_{N+1})$ and $L(\boldsymbol{\theta}_{N+1}, \mathbf{x}_{N+1})$ to $\mathbf{X}$ and $\mathbf{L}$ respectively
10: **end while**
11: **return** $\boldsymbol{\theta}^*$ as given by (26)

- the optimal hyperparameters for a given training set as given by (28).

Since each of these optimization problems works on the model of the simulator instead of on the simulator itself, they are considerably faster to solve. This in turn means that the overall optimization problem should also become faster to solve, as long as the computational complexity of creating the model is lower than that of evaluating the simulator. Although a number of different optimization strategies can be used to solve these optimization problems, we will use random search and grid optimization to keep the implementation relatively simple.

IV. RESULTS

We have implemented both the EGO and REGO algorithms as described in Section III. In this section we will describe the results of three types of experiments that we have performed using these implementations:

1) *Verification* experiments, to verify the behavior of the implemented algorithms by using them on optimization toy problems with a known solution.
2) *Validation* experiments, to validate that REGO can indeed be used to solve optimization problems with an expected loss function in the form as we have discussed in Section II for audio processing algorithms.
3) A *performance comparison* experiment comparing the performance of REGO with a more traditional optimization algorithm (DIRECT [13], [14]) on the expected loss function given by (4).

The three suboptimization tasks described at the end of Section III-E each require their own set of parameters. Since we are using random search optimization for each of these tasks, the only parameters we have to select are the bounds on the search space and a stopping criterion. As stopping criteria we use iteration budgets for each of the tasks. The budgets we use are the same as the ones used by Janusevskis *et al.*. For optimization of the acquisition function (24) and minimization of the predicted mean (26), this means that we set the budget to $300n \cdot (4 + \lfloor 3\log(n)n \rfloor)$ and for optimization of the hyperparameters (28), to $200 + 100(m + n)$, where m and n denote the dimensions of the random and deterministic input space respectively. The bounds on the search space of the first two problems are dependent on the experiment that is being run and will be presented in Sections IV-A and IV-B. The bounds on the hyperparameter optimization are set to two times the bounds on the other optimization problems. Finally we set the number of initial design points to $5(m + n)$ for each of the experiments.

A. Verification

Since REGO extends the EGO algorithm we have decided to verify the behavior of our implementations of both algorithms. As the test function we have chosen the modified six hump camelback function described in Section III. Since we know the locations of the global optima for this function, which we will denote by $\mathbf{x}_{\text{opt}}$, we can measure the convergence behavior of both algorithms by measuring the (Euclidian) distance to these optima, given by $\|\hat{\mathbf{x}}_{\text{opt}} - \mathbf{x}_{\text{opt}}\|$, for each optimization iteration. From the generated data we can verify whether the implementations of the algorithms work as intended.

As inputs to the test function we consider the parameters $\boldsymbol{\theta}$ and x. The bounds imposed on these parameters can be used to control the difficulty of the optimization problem. The parameter $\boldsymbol{\theta}$ will be bounded as $-1 \leq \boldsymbol{\theta} \leq 1$ in all problems. The parameter x can either be bounded as $-1 \leq x_E \leq 1$, which contains only the global minima of the six hump camelback function, or as $-1.9 \leq x_H \leq 1.9$ in which case the search space also contains the 4 local optima.

Since we are only interested in the location of the best performing parameters, we measure the convergence of the minimum distance to either optimum in case multiple equally good optima exist. This is only necessary when verifying the behavior for EGO on the six hump camelback function. To be able to increase the difficulty of the problems the optimization problem is evaluated on, we have also evaluated EGO on a one-dimensional function. This function is a sine wave on the interval $[-1, 1]$, with an extra minimum added around 0. The function is illustrated in the example shown in Fig. 5.

For the REGO test cases we define two probability distributions for $p(X)$, resulting in 2 possible cases for each selected domain. These distributions are given by $\mathcal{N}(0.5, 0.01)$, which should lead the optimization algorithm towards the optimum around -0.7126 and $\mathcal{N}(-0.5, 0.01)$ which should lead to the optimum around 0.7126.

The complete set of verification problems is shown in Table II, where D_d indicates the deterministic input domain, D_r indicates the random input domain and the problems are labeled for future reference.

TABLE II
VERIFICATION EXPERIMENTS

Label	Algorithm	D_d	D_r	$p(X)$
1	EGO (1D)	$\boldsymbol{\theta}$	-	-
2	EGO	$\boldsymbol{\theta}, x_E$	-	-
3	EGO	$\boldsymbol{\theta}, x_H$	-	-
4	REGO	$\boldsymbol{\theta}$	x_E	$\mathcal{N}(0.5, 0.01)$
5	REGO	$\boldsymbol{\theta}$	x_E	$\mathcal{N}(-0.5, 0.01)$
6	REGO	$\boldsymbol{\theta}$	x_H	$\mathcal{N}(0.5, 0.01)$
7	REGO	$\boldsymbol{\theta}$	x_H	$\mathcal{N}(-0.5, 0.01)$

We have measured the convergence behavior over 50 iterations for each of the verification problems in Table II and repeated this experiment 20 times for different initial models. The resulting mean convergence behavior for each

problem is shown in Fig. 6. From this figure we conclude that the implementations of the optimization algorithms behave as intended. The optimization procedures only approximately acquire the optimum, but we attribute this to the random search optimizer used for finding the minimum of the predicted mean, together with its limited iteration budget. We expect these results to become better, when a more accurate optimizer is used.

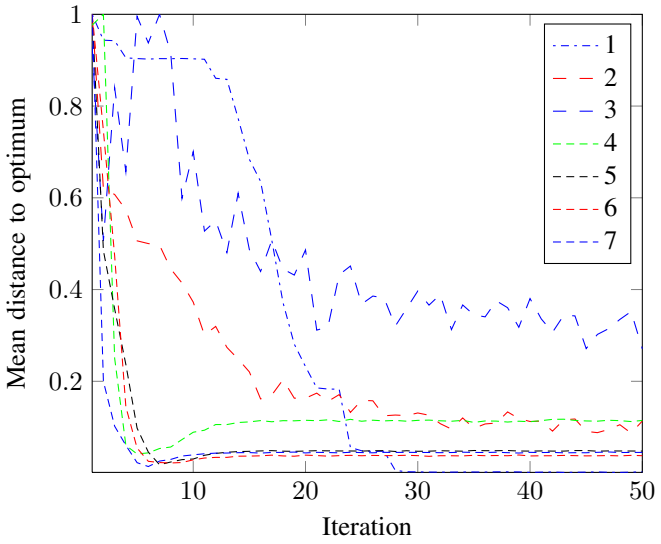

Fig. 6. Convergence behavior for the experiments described in Table II over 20 repetitions for 50 iterations of the optimization algorithm. The mean distance in each iteration is given by $\frac{1}{20}\sum_{k=1}^{20}\|\hat{\mathbf{x}}_{\text{opt},k} - \mathbf{x}_{\text{opt}}\|$, where $\mathbf{x}_{\text{opt}}$ denotes the known optimum and $\hat{\mathbf{x}}_{\text{opt},k}$ is the estimated optimum in the kth repetition.

B. Validation

To be able to validate our implementation of REGO and to determine the feasibility of computer-aided algorithm design for audio algorithms, we will first describe the design spaces $H, \Theta, \mathcal{X}, r_f$ and L required by the CAAD process for an example problem.

1) The Noise Reduction Algorithm: As the element H of our design space we have chosen a variant of a spectral noise subtraction algorithm. This algorithm tries to improve the overall intelligibility of clean speech signal c embedded in a noisy signal x by removing power from the frequency bands that negatively affect the overall signal-to-noise ratio (SNR) of the signal. How much of the signal power is removed per band is controlled by a gain factor $G_{\min} \leq G_f \leq 1$, which depends on the estimated $\widehat{\text{SNR}}_f$ in band f and is given by

$$G_f = \frac{\widehat{\text{SNR}}_f + G_{\min}}{\widehat{\text{SNR}}_f + 1}. \tag{29}$$

The parameter that we would like to tune during our investigation is $G_{\min}$, which is bounded as $0 \leq G_{\min} \leq 1$. The main reason for choosing this parameter is that it has a clear interpretation as the allowed aggressiveness of the algorithm. Choosing $G_{\min} = 0$ puts the algorithm in its most aggressive mode, enabling it to completely remove a band from the power spectrum, whereas setting $G_{\min} = 1$ effectively disables the algorithm in a band. We should note that the scaling using $G_{\min}$ is in practice performed in the dB domain, so that the bounds used for the experiments with the optimization process are given by $-\infty \leq G_{\min} \leq 0$, this explains why we will see that the distance to the optimal value of $G_{\min}$ can be larger than 1 in our experiments.

2) Problem Instances: Since the algorithm we are optimizing aims to improve intelligibility of a speech signal c embedded in a noisy signal x, we would like our problem instances $\mathcal{X}$ to reflect this class of signals. The problem here is that our set of signals represents a discrete input space, whereas REGO requires a continuous one. Furthermore the process requires these signals to be Gaussian distributed over the set. We can solve this problem by choosing a suitable continuous input space and generating signals x_k from this space for the points where REGO wants to sample.

We arrive at this continuous input space through a measure for intelligibility. Intelligibility can be quantified using the speech reception threshold (SRT). When measured in noise, the SRT is defined as the SNR level in dB at which listeners identify words with 50% accuracy, [15]. The SRT is an empirical measure and can be determined from a performance SNR plot, examples of which are shown in Fig. 7.

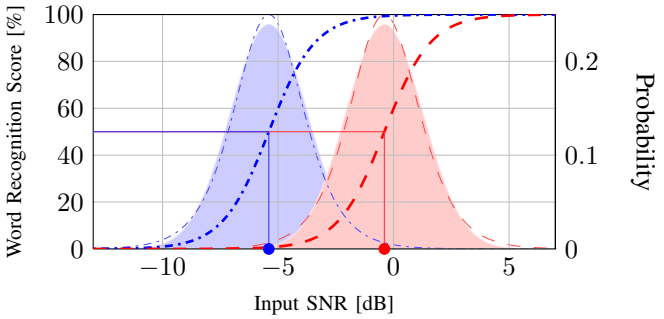

Fig. 7. Performance SNR plots for an average normal hearing person (·-·-·) and an average hearing-impaired person with 5 dB SNR loss (– –), [16], [17]. These plots can be used to determine the respective average speech reception thresholds (SRTs) (● and ●). The lines ·-·-· and – – show the respective derivatives of the SNR plots and the shaded areas show Gaussian distributions fitted to these derivatives.

From Fig. 7 we see that it is possible to fit a Gaussian distribution around the SRT. This Gaussian distribution captures the notion that the main region of interest, in terms of input SNR, for improving intelligibility lies around the SRT. For SNRs further away from the SRT there is either no need to improve intelligibility or it becomes almost impossible to improve intelligibility. Using the definition of SNR in dB

$$\text{SNR} \triangleq 10\log_{10}\frac{P_c}{P_n}, \text{with } P_x = \frac{1}{T}\sum_{t=1}^{T} x^2(t) \tag{30}$$

we can derive an exact relation between $\text{SNR}_k \sim \mathcal{N}(\mu_{\text{SNR}}, \sigma^2_{\text{SNR}})$ and a problem instance x_k. When the signals c and n are of equal average power, this relation is given by

$$x_k = c + \sqrt{10^{-\frac{\text{SNR}_k}{10}}} \cdot n. \tag{31}$$

For our validation experiment, we derive the values of μ_{SNR} and σ^2_{SNR} from Fig. 7 and set them to the SRT and $\frac{25}{9}$ respectively. For c we pick a single sentence from the TIMIT database ("Kindergarten children decorate their classrooms for all holidays"), [18] and for n we pick white noise equal in average power to c.

3) Loss Function: We would like the noise reduction algorithm to try and make a hearing impaired person perceptually indistinguishable from a normal hearing person with respect to a psychoacoustic Turing test [19]. From this we would like the loss function to be perceptually relevant. To this end we will use a modified version of the segmental logarithmic spectral estimation error defined by Erkelens and Heusdens [20] as

$$L(y_k, r_k) = \frac{20}{F \cdot T} \sum_{f=1}^{F} \sum_{t=1}^{T} \left| \log_{10} \frac{|R_k(t,f)|}{|Y_k(t,f)|} \right|, \tag{32}$$

where Y_k and R_k are the short-time Fourier transforms of the output y_k of the algorithm and a reference signal r_k respectively.

From Fig. 7 we know that for a hearing impaired person to be indistinguishable from a normal hearing person, the SNR of y has to be SNR_{loss} better than that of the input signal x. Using (31) this leads us to define the reference signals r_k for each x_k as

$$r_k = c + \sqrt{10^{-\frac{\text{SNR}_k + \text{SNR}_{\text{loss}}}{10}}} \cdot n, \tag{33}$$

where we pick c and n equal to our choices in Section IV-B2.

4) Validation Results: Using the design space described in Sections IV-B1 to IV-B3 we have performed three separate optimizations to validate our implementation of the REGO algorithm and to check whether the algorithm can indeed be used to solve (5). The different optimization problems were created by picking three different SNR losses (5, 7 and 10 dB) for generating the reference signals as described in Section IV-B2. Each of these optimization problems was given an iteration budget of 25 and was repeated 10 times.

To validate the optimization algorithm we investigate the convergence behavior of the distance of the estimated optimum in each iteration to the final optimum for a given repetition. If this convergence behavior averaged over all repetitions converges to an optimal value, we assume that our implementation of the algorithm works as intended. The convergence behavior for each of the three problems is shown in Fig. 8.

From these plots we conclude that the REGO algorithm can indeed be used to solve (5) given a properly chosen design space. We do have to note however that, given that the convergence behavior is not monotonically decreasing (as can be seen in Fig. 8), care should be taken when interpreting the optima returned by the algorithm. This non-monotonicity is introduced by the optimization of the hyperparameters with respect to the observed data in each of the optimization iterations. We do believe however that this optimization step can not be removed from the process, since it is not feasible to a priori determine appropriate hyperparameter values. On the other hand the algorithm will indicate that it is uncertain about these optima. So given a proper stopping criterion, such as the expected improvement measure, these faulty optima will typically not cause the optimization process to stop. Furthermore it is relatively easy for a researcher to inspect the convergence behavior of the algorithm allowing for appropriate conclusions to be drawn.

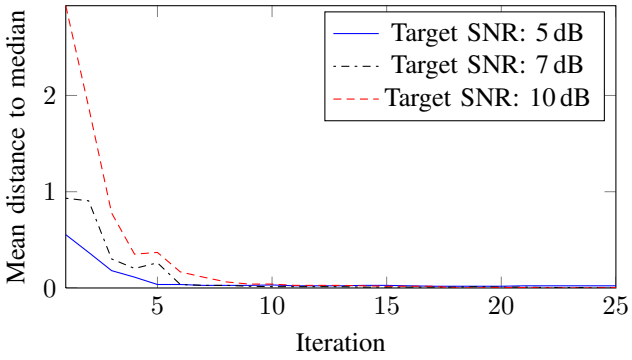

Fig. 8. Convergence behavior for the validation experiments over 10 repetitions for 25 iterations of the optimization algorithm. The mean distance to the final optimum in each iteration is given by $\frac{1}{10}\sum_{k=1}^{10} \|\boldsymbol{\theta}_k^* - \boldsymbol{\theta}_{k,25}^*\|$, where $\boldsymbol{\theta}_k^*$ is the optimum for an iteration in repetition k and $\boldsymbol{\theta}_{k,25}^*$ denotes the optimum in the 25th iteration of the kth repetition.

Furthermore we have to note that the algorithm suffers from an inherent problem when working with covariance matrices. Since the algorithm will try to keep sampling near the predicted optimum, the covariance matrix can become ill-conditioned. Although this can be partially solved by using the Cholesky factorization of the covariance matrix instead of the covariance matrix itself, as proposed in [10], we have noticed that the algorithm will sometimes lose track of the optimum due to ill-conditioned covariance matrices. For our validation experiments we have solved this problem by adding a small regularization term (10^{-6}) to the covariance matrix to make it numerically more stable. Although this works for our experiment, we do not believe that this will be easily applicable in general. Possible solutions could be to have the optimization algorithm determine the regularization factor as part of the hyperparameter optimization process as done by Janusevskis *et al.* [12] or applying techniques using derivative observations used to solve similar problems with the EGO algorithm as described by Osborne [21].

C. Performance Comparison

Using the results of Section IV-B we can compare the performance of REGO against a more traditional optimization algorithm in the form of DIRECT [13], [14]. Our main interest here is the difference in the number of evaluations of (3) required to obtain the optimum.

Since DIRECT is not able to create a model of the expected loss function, we have to define a range of SNRs to evaluate (4) over for each $\boldsymbol{\theta}$ the algorithm picks. By inspecting Fig. 7, we define this range to capture the full SNR range of interest for a given $\text{SNR}_{\text{target}}$, from $(-14 + \text{SNR}_{\text{target}})$ to $(2 + \text{SNR}_{\text{target}})$ dB and pick 50 SNRs from this range to describe the behavior of the algorithm under study for any given $\boldsymbol{\theta}$. Hence every evaluation of (4) requires 50 evaluations of (3). Running the

same experiment as we ran for REGO in Section IV-B, with the described setup for the expected loss function, results in the performance shown in Fig. 9.

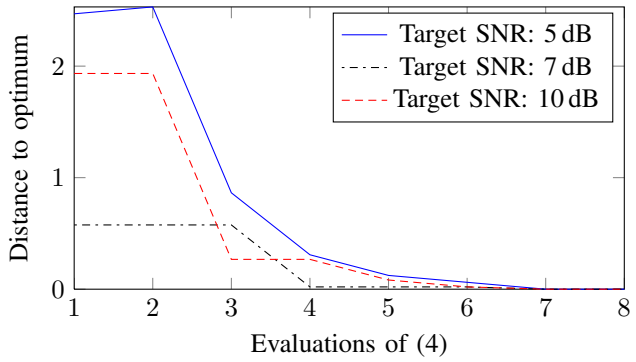

Fig. 9. Convergence behavior of $\|\boldsymbol{\theta}^* - \boldsymbol{\theta}^*_{\text{end}}\|$ using DIRECT to optimize the expected loss function (4) for $K = 50$, where $\boldsymbol{\theta}^*_{\text{end}}$ denotes the optimum found just before the optimization process is stopped.

From Fig. 9 we see that DIRECT obtains the optimal value of $\boldsymbol{\theta}$ after approximately 7 evaluations of (4), which translates into 350 evaluations of (3). Taking into account that REGO requires a number of initial evaluations of (3), which we have set to $5(n + m) = 10$, we can see from Fig. 8 that it requires approximately 30 evaluations of (3) to obtain the same optimum. From this we conclude that using REGO allows for significant performance gains to be reached over traditional methods and that it makes it possible to do CAAD for audio algorithms. One can argue that the number of 50 SNRs used for describing the SNR space in DIRECT's optimization case is rather arbitrary, but to reach a performance equal to that of REGO this number would have to be reduced to 5. This in turn does not seem like a feasible amount to get a proper representation of the true behavior over SNRs for a given $\boldsymbol{\theta}$.

V. Conclusion

We have demonstrated that it is possible to apply the ideas behind computer-aided algorithm design to audio processing algorithms. The process even becomes feasible for expensive loss functions when using the robust efficient global optimization algorithm. We must conclude though that it is not a design process that is applicable out of the box. The restrictions placed upon the response surface, which must be able to be modelled using a Gaussian process with a squared exponential kernel, and on the random input space, which needs to be Gaussian distributed, limit the applicability of the optimization algorithm. However when a problem is known to fit these descriptions, or can be altered to fit the descriptions as we have done in our example case, the design process yields answers far more efficiently than traditional optimization processes. We have observed efficiency increases of an order of magnitude.

Interpreting the results of the optimization process does require some extra attention from the researcher, since the process does not monotonically decrease towards an optimum due to the optimization of the hyperparameters. It does however appear to converge to an optimum overall and whenever the process tends away from the optimum, this is reflected in its uncertainty about the selected optimum in that iteration. Unfortunately this effect can be exaggerated by the covariance matrix becoming ill-conditioned. We believe however that given further research this latter problem can be mitigated.

Other topics that warrant future research, since we have not been able to study them yet, are higher dimensional problems in both the deterministic and random input spaces. Extensions of the deterministic input space can be implemented by evaluating a larger number of parameters of the algorithm, whereas as an extension to the random input space in our example problem could for instance be a distribution over common SNR losses.

Finally an interesting topic in extending the CAAD process would be to also assign the task of designing the structure of H to the automated process. This should be possible as long as this structure can be parametrized in Θ.

VI. Contributions

In closing the author would like to summarize his contributions to the research project which led to the conclusions drawn in this thesis:

- Performed literature research to reach a common language to talk about design processes and to identify common problems inherent in the design of algorithms;
- Performed literature research for optimization techniques allowing us to solve our problem statement;
- Created a framework in MATLAB [22] for building "Ask & Tell" optimizers as defined by Collette *et al.* [23] and implemented several optimization strategies such as random search and optimization over a grid;
- Implemented Robust Efficient Global Optimization in MATLAB after a Scilab [24] implementation of EGO by Janusevskis and Le Riche [25];
- Identified a number of numerical instability issues which were hard to diagnose due to the extended feature set of the reference implementation;
- Linked the mathematics of REGO described by Janusevskis *et al.* [12] in vector form to the fundamental equations for Gaussian processes described by Rasmussen and Williams [10];
- Implemented Efficient Global Optimization and Robust Efficient Global Optimization using the Ask & Tell framework;
- Verified implementation of EGO and REGO on a set of toy problems as mentioned throughout this work;
- Constructed an experiment to validate the behavior of REGO and verify its applicability to solving the optimization process in our problem statement;
- Compared the performance of REGO on the designed experiment to a more traditional optimization process in the form of DIRECT [13], [14].

References

[1] H. H. Hoos, "Computer-aided design of high-performance algorithms," University of Britisch Columbia, Tech. Rep., 2008.

[2] ——, "Computer-aided algorithm design: Automated tuning, configuration, selection, and beyond," in *Proceedings of the 29th International Conference on Automated Planning and Scheduling, ICAPS 2010, Toronto, Ontario, Canada, May 12-16, 2010*, R. I. Brafman, H. Geffner, J. Hoffmann, and H. A. Kautz, Eds. AAAI, 2010, pp. 268–269.

[3] T. Heskes and B. de Vries, "Incremental utility elicitation for adaptive personalization," in *BNAIC 2005, Proceedings of the Seventeenth Belgium-Netherlands Conference on Artificial Intelligence*, K. Verbeeck, K. Tuyls, A. Nowé, B. Manderick, and B. Kuijpers, Eds. Brussels: Koninklijke Vlaamse Academie van België voor Wetenschappen en Kunsten, 2005, pp. 127–134.

[4] I. Tashev, A. Lovitt, and A. Acero, "Unified framework for single channel speech enhancement," in *Communications, Computers and Signal Processing, 2009. PacRim 2009. IEEE Pacific Rim Conference on*, aug. 2009, pp. 883 –888.

[5] L. Dixon and G. Szegö, "The global optimization problem: an introduction," in *Towards Global Optimisation 2*, L. Dixon and G. Szegö, Eds. North-Holland Publishing Company, Amsterdam, 1978, pp. 1–15.

[6] D. R. Jones, M. Schonlau, and W. J. Welch, "Efficient global optimization of expensive black-box functions," *Journal of Global Optimization*, vol. 13, pp. 455–492, 1998, 10.1023/A:1008306431147. [Online]. Available: http://dx.doi.org/10.1023/A:1008306431147

[7] C. M. Bishop, *Pattern recognition and machine learning*, 1st ed. Springer, Oct 2006.

[8] E. Brochu, V. M. Cora, and N. de Freitas, "A tutorial on Bayesian optimization of expensive cost functions, with application to active user modeling and hierarchical reinforcement learning," *CoRR*, vol. abs/1012.2599, 2010.

[9] A. Girard, "Approximate methods for propagation of uncertainty with gaussian process models," 2004.

[10] C. E. Rasmussen and C. K. Williams, *Gaussian processes for machine learning*. MIT Press, 2006.

[11] P. Groot, A. Birlutiu, and T. Heskes, "Bayesian monte carlo for the global optimization of expensive functions," in *ECAI 2010*, H. Coelho, R. Studer, and M. Wooldridge, Eds., vol. 215, 2010, pp. 249–254.

[12] J. Janusevskis and R. L. Riche, "Simultaneous kriging-based sampling for optimization and uncertainty propagation," 2010, deliverable no. 2.2.2-A of the ANR / OMD2 project. [Online]. Available: http://hal.archives-ouvertes.fr/hal-00506957/en/

[13] D. R. Jones, C. D. Perttunen, and B. E. Stuckman, "Lipschitzian optimization without the Lipschitz constant," *J. Optim. Theory Appl.*, vol. 79, pp. 157–181, October 1993. [Online]. Available: http://dl.acm.org/citation.cfm?id=173669.173678

[14] D. E. Finkel, *DIRECT Optimization Algorithm User Guide*, 2003. [Online]. Available: http://www4.ncsu.edu/~ctk/Finkel_Direct/DirectUserGuide_pdf.pdf

[15] P. Loizou, *Speech enhancement: theory and practice*, ser. Signal processing and communications. CRC Press, 2007. [Online]. Available: http://books.google.nl/books?id=3CZLAQAAIAAJ

[16] R. Plomp and A. M. Mimpen, "Speech-reception threshold for sentences as a function of age and noise level," *The Journal of the Acoustical Society of America*, vol. 66, no. 5, pp. 1333–1342, 1979. [Online]. Available: http://link.aip.org/link/?JAS/66/1333/1

[17] M. C. Killion, "SNR loss: 'I can hear what people say, but I can't understand them'," *The Hearing Review*, vol. 4, pp. 8–14, December 1997.

[18] J. S. Garofolo *et al.*, "TIMIT acoustic-phonetic continuous speech corpus," 1993.

[19] B. Edwards, "Signal processing, hearing aid design, and the psychoacoustic Turing test," in *Acoustics, Speech, and Signal Processing (ICASSP), 2002 IEEE International Conference on*, vol. 4, may 2002, pp. IV–3996 –IV–3999.

[20] J. S. Erkelens and R. Heusdens, "Tracking of nonstationary noise based on data-driven recursive noise power estimation," *IEEE Transactions on Audio, Speech & Language Processing*, vol. 16, no. 6, pp. 1112–1123, 2008.

[21] M. Osborne, R. Garnett, and S. Roberts, "Gaussian processes for global optimization," *3rd International Conference on Learning and Intelligent Optimization (LION3)*, pp. 1–15, 2009.

[22] MATLAB, *version 7.13.0.564 (R2011b)*. Natick, Massachusetts: The MathWorks Inc., 2011.

[23] Y. Collette, N. Hansen, G. Pujol, D. S. Aponte, and R. L. Riche, "On object-oriented programming of optimizers – examples in Scilab," in *Multidisciplinary Design Optimization in Computational Mechanics*, P. Breitkopf and R. F. Coelho, Eds. Wiley, 2010, ch. 14, pp. 527–565.

[24] Scilab Consortium, *Scilab: Free and Open Source software for numerical computation*, Scilab Consortium, Digiteo, Paris, France, 2011. [Online]. Available: http://www.scilab.org

[25] J. Janusevskis and R. L. Riche, "KRIging Scilab Package (KRISP) v0.9-2," January 2011, note. [Online]. Available: http://atoms.scilab.org/toolboxes/krisp/0.9