Proceedings of the 17th World Congress
The International Federation of Automatic Control
Seoul, Korea, July 6-11, 2008

Approximate Viability using Quasi-Random Samples and a Neural Network Classifier

B. Djeridane * **J. Lygeros** **

* *ETH-Zurich, Automatic Control Laboratory,8092 Zurich, Switzerland, djeridane@control.ee.ethz.ch*
** *ETH-Zurich, Automatic Control Laboratory,8092 Zurich, Switzerland,lygeros@control.ee.ethz.ch*

Abstract: We propose a novel approach to the computational investigation of reachability properties for nonlinear control systems. Our goal is to combat the curse of dimensionality, by proposing a mesh-free algorithm to numerically approximate the viability kernel of a given compact set. Our algorithm is based on a non-smooth analysis characterization of the viability kernel. At its heart is a neural network classifier based on Bayesian regularization, which operates on a pseudorandom sample extracted from the state-space (instead of a regular grid). The algorithm was implemented in Matlab and applied successfully to examples with linear and nonlinear dynamics.

Keywords: Neural Network, Viability, Quasi-random technique.

1. INTRODUCTION

The *viability set* of dynamic system is the set of states from which trajectories start that and remain within a set under the system's prescribed dynamic. Calculating a viable sets for continuous dynamic systems is a challenge because the set involved contain an uncountable number of states. Furthermore, existing algorithms for viability are complex enough that their implementations must be carefully validated against known examples.

The question of viability set for nonlinear can be formulated as optimal control problem, whose solution can be characterized using variants of the Hamilton-Jacobi-Bellman equation [8]. The formal link between the solution to the partial differential equation and reachability problem is usually established in the framework of viscosity solutions. The main advantage of this approach is an numerical Toolbox [10] is already developed to solve the this particular form of PDE [11].

Alternative approach to deal with viability set computation, is to address the problem by viability theory [1]. This approach is based on geometrical characterization of the viability set, provide an approximation of the viability set by discretization both in time and in space, and pointwise convergence of a numerical scheme. The development of computational tools to support this approach is not available [3].

These both approach suffer from exponential complexity. It is for these reasons that we accept approximate solution to provide us with certain guarantees in such viability problem. This is when sampling methods come into picture to try and remedy the "complexity of solution" problem by drawing samples a sample space, and providing approximate solution [13]. Then instead of griding use samples from state space and compute viability kernel by using viability theory [Djeridane et al.]. And we have used 1-nearest neighborhood method to find out the viable samples, this strategy require the entire training data set to be stored, leading to expensive computation if the data set is large, the complexity of this problem can be reduced by constructing tree-based search structures to allow near neighborhood to be found efficiently without doing an exhaustive search of the data set, but still expensive in resources required.

It well known that Neural Networks are good classifier [2]. Since we are interested to classify the samples in two set: "viable" samples and "not viable" samples, then neural networks seem like ideal candidate to approximate the viable set. To the best of our knowledge, this approach is novel and no prior works on viability set using neural networks as classifier. This approach provide us an analytical approximation with fewer parameters and number of samples point necessary to compute viability grow polynomially with dimension of the state space.

The paper is organized as follows. Section 2 briefly describes the problem of viability computation, while 3 introduce Neural Networks classifier, method used to generate samples and the proposed algorithm to compute the viability set and their results are presented and discussed in 4. Section 5 concludes.

2. PROBLEM FORMULATION

Consider the controlled system

$$\dot{x}(t) = f(x(t), u(t)) \tag{1}$$

with $u \in U$ this classical formulation of the system (1) can be represented by the following differential inclusion

$$\dot{x}(t) \in F(x(t)) \tag{2}$$

where $F : X \to X$ is the set-valued map defined by

$$\forall x \in X, F(x) := \{f(x,u), u \in U\}$$

The systems (1) and (2) have continuously differentiable functions. Let us consider a closed nonempty set $K \subset X$. We shall say that a solution to (1) (or equivalently to (2)) is viable in K if and only if $x(t) \in K$ for any $tgeq0$. In other words, the viability kernel of K for F is the set [3]

$$Viab_F(K) := \{x_0 \in K | \exists x(.) \in S_F(x_0), x(t) \in K, \forall t \geq 0\}$$

with $S_F(x_0)$ is denote continuous solution of (2) from x_0 starting at $t = 0$. For computing $Viab_F(K)$ without computing trajectory, we need to characterize $Viab_F(K)$ in geometrical way. We first replace the initial differential inclusion system by a finite difference inclusion system (semi-discrete scheme). Secondly, we replace the state space X by an integer lattice X_h of X (fully discrete scheme). Finally we apply refinement principle.

2.1 Discrete-time Viability Kernel

Let us consider F_ϵ an approximation of F and define

$$G_\epsilon = x + \epsilon F_\epsilon(x) \tag{3}$$

The choice of F_ϵ depends on the regularity of the dynamic F. The discretized dynamic correspond to Euler scheme is

$$x_{k+1} \in G_\epsilon = x + \epsilon F_\epsilon(x) \tag{4}$$

And the viability discrete set D could be defined as: for $x_0 \in D$ there exists at least a finite sequence $(x_k)_k$ solution to the recursive inclusion (4) stay belonging to D for any $k \geq 0$. We compute it as follow:

Algorithm 1 Semi-discrete Viability Kernel Algorithm

$i = 0$
$K^0 = K$
repeat
  $K^{i+1} = \{x \in K^i | G(x) \bigcap K \neq \emptyset\}$
  $i = i + 1$
until $K^{i+1} = K^i$ OR $K^{i+1} = \emptyset$
$Viab_G(K) = \bigcap_{k=0}^{\infty} K^i$

Under the following fair assumptions [3]

- F is bounded
- F_ϵ is upper semi-continuous with convex compact nonempty values
- $Graph(F_\epsilon(.)) \subset Graph(F(.)) + \phi(\epsilon)B$ where $\lim_{\epsilon \to 0^+} \phi(\epsilon) = 0^+$
- $\forall x \in X, \bigcup_{\|y-x\| \leq M\epsilon} F(y) \subset F_\epsilon(x)$

for any $\epsilon > 0$, we can approach $Viab_F(K)$ by discrete kernels $Viab_{G_\epsilon}(K)$

$$Viab_F(K) \subset Viab_{G_\epsilon}(K)$$

and

$$\lim_{\epsilon \to 0} Viab_{G_\epsilon}(K) = Viab_F(K) \subset Viab_{G_\epsilon}(K)$$

2.2 Fully discrete viability kernel algorithm

To implement this algorithm we have to associate with G_ϵ suitable finite set-valued maps defined on finite sets. Now we are dealing with systems which are not only discrete on time but finite on state too.

With any closed set K we associate its "projection onto the grid" defined by

$$K_h := (K + hB) \cap X_h$$

Then the algorithm (1) will be as follow for this case

Algorithm 2 Fully discrete Viability Kernel Algorithm

$i = 0$
$K^0_{\epsilon,h} = K_h$
repeat
  $K^{i+1}_{\epsilon,h} = \{z_h \in K^i_{\epsilon,h} | \Gamma_{\epsilon,h}(z_h) \bigcap K^i_{\epsilon,h} \neq \emptyset\}$
  $i = i + 1$
until $K^{i+1}_{\epsilon,h} = K^i_{\epsilon,h}$ OR $K^{i+1}_{\epsilon,h} = \emptyset$
$Viab_{\Gamma_{\epsilon,h}}(K_h) = K^i_{\epsilon,h}$

Under the same assumptions as previously stated for the case of discrete-time viability kernel, in addition we assume also that $\Gamma_{\epsilon,h}$ is a good approximation of G_ϵ

$$Graph(\Gamma_{\epsilon,h}()) \subset Graph(G_\epsilon(.)) + \psi(\epsilon,h)B$$

where $\lim_{\epsilon \to 0, \frac{h}{\epsilon} \to 0} \frac{\psi(\epsilon,h)}{\epsilon} = 0$ holds. Moreover this condition express the compatibility between time and space steps discretization.

Then,

$$Viab_F(K) \subset Viab_{\Gamma_{\epsilon,h}}(K_h) + hB$$

and

$$Viab_F(k) = \lim_{\epsilon 0, \frac{h}{\epsilon} \to 0} Viab_{\Gamma_{\epsilon,h}}(K_h)$$

2.3 Our settings

When F is a Lipschitz set-valued map, we can construct the set-valued map $\Gamma_{\epsilon,h}$ as follows. Let us recall that

$$G_\epsilon = x + \epsilon F(x) + Ml\epsilon^2 B$$

and setting

$$\Gamma_{\epsilon,h}(x_h) = \left[x_h + \epsilon F(x_h) + (2h + l\epsilon h + Ml\epsilon^2)B\right] \cap X_h$$

where M is upper bound of F

$$\exists M \geq 0, \forall x \in X, \forall y \in F(x), \|y\| \leq M$$

yields approximation which satisfies the previous assumptions. In practice we can choose l an upper bound of the derivative of $f(x,u)$ for all $u \in U$ and $\epsilon^2 = h$.

3. VIABILITY COMPUTATION

3.1 Generating sampling points

Assume that we are trying to minimize a function over the unit cube ($[0,1]^d$), and the minimum point is exactly at the center of the hypercube. Also consider a smaller hypercube inside of the original one with sides equal to $1-\frac{\epsilon}{2}$. Now, extract a random point inside the hypercube according to an uniform distribution probability, then the results in the probability of sampling inside the smaller cube is

$$(1-\epsilon)^d$$

where d is the dimension of the hypercube [12]. Then, the probability of the sampling inside the smaller cube tends to zero as d goes to infinity, hence the clustering effect on the surface is observed as we go into higher dimensions. We would like to replace the random samples required for computing a viability set with deterministic samples that posses certain regularity condition, i.e. they are regularly spread within the sampling space. This method is also independent of the sampling space. It has shown its superiority over classic Monte Carlo methods in the calculation of certain integrals [5] and robust control problem [7].

There are many sequences in the literature used to generate a sampling points, which include the property of "evenly distributed", such as Halton , Hammersley , Sobol sequences. In this paper, we are interested in Hatlon sequence which has low discrepany sequence (measure of how the samples set is equi-distributed within integration domain) and efficient computation technique. Based on prime numbers, these pseudo-random distribution is uniform and irregular, but lack point in close proximity, i.e., they have a minimum resolution that increases as the number of points in the sample increases. The gives the algorithm for computing Halton points in up to ten dimension. The function, $p_2(n,d)$, takes the number of points and parameter as input and returns a (rational) number belonging to the interval $[0,1)$. [14]

Algorithm 3 Algorithm for Generating Halton

$prime = \{2,3,5,7,11,13,17,19,23,29\}$
choose p_1 from $prime$
$p_2 = p_1$
$\phi = 0$
repeat
  $a = n \mod p_1$
  $\phi = \phi + \frac{a}{p2}$
  $n = int(\frac{n}{p1})$
  $p_2 = p_2 p_1$
until $n \leq 0$

where $int(x)$ returns the integer part of x.

Figure (1) compares the uniform random Halton distribution in two dimension. The Halton distribution is more even, but the random distribution displays a wider range of difference vector magnitudes.

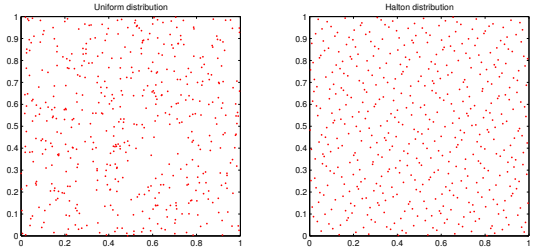

Fig. 1. 500 points distributed with random uniform distribution (left) and according to a two-dimensional Halton point set (right)

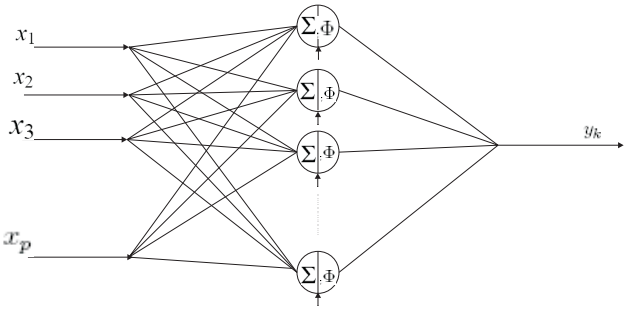

Fig. 2. Neural Networks Architecture

3.2 Neural Network Classification

General presentation Neural Network are closely related to the Bayesian probabilities. They may be used for unsupervised learning (density estimation problem), and mainly, supervised learning problems (regression, classification) [6]. Density estimation is not the scope of our paper, so we will not detail it. The aim of regression is to find a statistical model producing an output y from input variables (let us denote them by y), so that the output y is as close as possible to a target variable, which we shall denote by t. In the case of *classification* problems, the target variables represent class labels, and the aim is to assign each input vector x to a class which is the scope of the paper.

We will use a specific class of neural networks, referred to as feed-forward networks. In such networks, the units (neurons) are arranged in fully-connected layers: an *input layer*, one or several *hidden layers*, and an *output layer*. Figure xx shows an example of such a network.

For a network with one hidden layer, the output vector $y = (y_1, \cdots, y_k, \cdots, y_q)^T$ is expressed as a function of the input vector $x = (x_1, \cdots, x_i, \cdots, x_p)^T$ as follows:

$$y_k = \Psi\left(\sum_{j=1}^{q} w_{jk}\Phi\left(\sum_{i=1}^{p} w_{ij}x_i + w_{0j}\right) + w_{0k}\right)$$

where the w_{ij} and w_{jk} are weights assigned to the connections between the input layer and the hidden layer, and the output layer, respectively, and where w_{0j} and w_{0k} are biases. Φ is an activation function, applied to the weighted output of the preceding layer, and Ψ is a function applied, by each output unit, to the weighted sum of activations of the hidden layer. This expression can be generalized to network with several hidden layers.

The output error - i.e. the difference between the target values t and the output y computed by the network - will depend on the parameters w (weights and biases). The *training* aims at choosing these parameters, so as to minimize a chosen function of the output error.

In the case of *classification* problem, it is recommended to consider a log-likelihood function. The network is then design with one output unit per class. When the output of such unit is 1, the input x is assigned to the class corresponding to the unit, and when the output is 0, it is not. Let us consider a problem with 2 classes. The sum of squares error function minimized during training is the following:

$$E_D(w) = \frac{1}{2}\sum_{i=1}^{n}\sum_{k=1}^{2}(y_k^i - t_k^i)^2$$

where t^i and y^i are the i^{th} target and output vectors, respectively. The set of weights w for which the error function $E_D(w)$ is minimum, is obtained through *back-propagation* of the error through the network. The training with backpropagation consists in computing small modifications of the weights, for successive layers, starting from output layer, and considering the influence of each weighted connection in the output error variations. Several *backpropagation* methods which are basically optimizations techniques, are proposed in the literature and in existing software: gradient methods, or global optimizations methods.

Neural networks applied to our problem For our problem, we have chosen three-layers feed-forward networks, denoted I_pH_qO in the rest of the paper, with p units in the input layer, q units in the hidden layer and 1 unit in the output layer (because we are dealing with two classes). The input variables are normalized, by subtracting the mean value and dividing by standard deviation. There are many possible choices for the function Ψ and Φ, depending on the problem being addressed. A common choice for Φ is logistic function $\Phi(z) = \frac{1}{1+e^{-z}}$. This is the activation function that was used in our experiments. As we address a classification problem - assign each input vector to a class representing the point (viable or not) - we have chosen to minimize the cross entropy function. Therefore, the transfer function Ψ applied to the output layer, must be the *logistic* function $\Psi(z) = \frac{1}{1+e^{-z}}$.

A well-known problem, when using neural network (or other regression methods), is *overfitting*: with enough parameters and enough training cycles, it is always possible to find a good fit for a given data set. So one may find perfect fit for chosen data simple, and then feel disappointed when trained network lakes wrong predictions on fresh data. So, we will systematically proceed as follows: train the network on a randomly chosen data sample (called *train*), then check the results, first on the same data, and second on a fresh data sample (called *test*), that was not used for the training. We may use the fit criterion but it does not reflect the influence of the number of weights (and biases) in the neural network. It is known that a network with too few weights may not able to capture all the variations of the response to the input x, whereas a network with too many weights will more likely be subject to "over fitting". However, with objective functions $E(w) = \beta E_D(w) + \alpha E_W(w)$ where E_w is the sum square of the network weights and α is objective function parameters. If $\alpha << \beta$, then the training algorithm will drive the errors smaller. If $\alpha >> \beta$, training will emphasize reduction at the expense of the network errors, thus reducing a smoother network response. The main challenge to optimize E is setting the correct values for the objectives function parameters. In [4] have developed technique to compute α and β based on the application of Bayes' rule to neural network training and optimization regularization. In this approach, the weights and the biases of the network are assumed to be random variables with specified distribution. To deduce the optimal values of these parameters, the Bayesian framework of [9] is applied. In it the weights of the network are considered random variables. Assuming that both the noise in the data set and the prior distribution for the weights are Gaussian, it can found that, as detailed in [4], the optimal values of $\alpha = \frac{\gamma}{2E_w}, \beta = \frac{n-\gamma}{2E_D}$ where $\gamma = N - 2\alpha tr(H^{-1})$ is called the effective number of parameters, N is the total number of parameters in the network, n is the number of examples in the data set, and H is the Hessian of the objective function.

Once the error index to minimize is fixed, a suitable minimization method must be selected to perform the network training. A usual choice is the Levenberg-Marquardt algorithm.

3.3 Computation techniques

In this subsection, we introduce a randomized approach for computing the viability kernel. As pointed out in the introduction, many pessimistic results on the complexity theoretic barriers of classical computation of the viability set have simulated research in the direction of finding alternative solutions. One of these solutions, which is the main objective of this paper, is first to shift the meaning of representing a set from its unusual deterministic sense to probabilistic one. In this respect, we accept the risk that viability set computed being violated by a set of points of K having small probability measure. Such viability set can be viewed as being *almost accurate* approximation.

We now study the computation of viability set problems previously discussed by introducing two sets, denoted as the "*good set*" and the "*bad set*". These are subsets of K and represent, respectively, collection of all "points" which satisfy or violate the viability property under attention. These sets are constructed so their union coincides with the set K and their intersection is empty. Formally, we define $K_G = \{x \in K | y \in K\}$,$K_B = \{x \in K | y \notin K\}$

An iterative approach is used to compute the viability kernel: we shall first used samples points as input to the neural network. The E criterion is used to select which

samples are outside the viability kernel (called "bad sets") and inside the viability kernel (called "good sets").

The randomized algorithm for computing viability kernel is presented as follow:

Algorithm 4 RA for computing a viability kernel

$i = 0$
Choose n
Generate samples points over K using halton sequence
Train the neural network on samples
Initialize $K_G = \{x \in K\}$, $K^0 = K$ and $K_B = \{x \notin K\}$
repeat
  Compute $y = x + \epsilon F(x)$ with $x \in K_G$
  Use NN to find out the bad point R
  $K_G = K_G / R$
  $K_B = K_B \cup R$
  train NN on the set $K^{i+1} = K_G \cup K_B$
until $K^{i+1} = K^i$ OR $K^{i+1} = \emptyset$
$Viab_G(K) = K^{i+1}$

Now we comment on how we compute y (successor of x). In the algorithm is stated that

$$y = x + \epsilon F_\epsilon(x)$$

where $F(x)_\epsilon = F(x) + Ml\epsilon B$ with $l = \sup_x \frac{f(x,u)}{x} \quad \forall u \in U$, $M = \sup_x f(x,u) \forall u \in U$ and B is a ball. We use dichotomy technique to find out the appropriate control u in U.

4. EXAMPLES

To validate the algorithm presented in this paper, we have worked on linear system and nonlinear systems and we have compared our results with Level set Toolbox developed in [10]. We have perform all the computations on the a Opteron 64 processor running on Linux with 4 GB memory.

4.1 Linear system

We have applied our approach to linear system with canonical controllability form. And we have compared our approach to Level Set Toolbox method.

2D example Consider the double integrator

$$\begin{aligned} \dot{x}_1 &= x_2 \\ \dot{x}2_2 &= u \end{aligned}$$

with $u \in U = [-1, 1]$ and $x \in K = [-1, 1] \times [-1, 1]$.

3D example

$$\begin{aligned} \dot{x}_1 &= x_2 \\ \dot{x}2_2 &= x_3 \\ \dot{x}2_3 &= u \end{aligned}$$

with $u \in U = [-1, 1]$ and $x \in K = [-1, 1]^3$.

To compare, first we have computed the "exact" solution for our example using level set toolbox with high accuracy, and afterward we have used the technique developed (Method 1) in this paper with tolerance of 5% of error

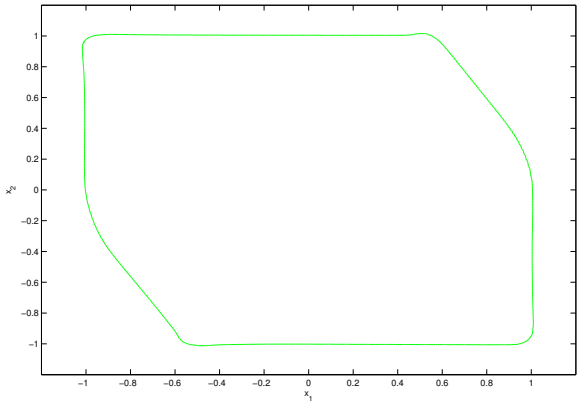

Fig. 3. Viability set for 2D

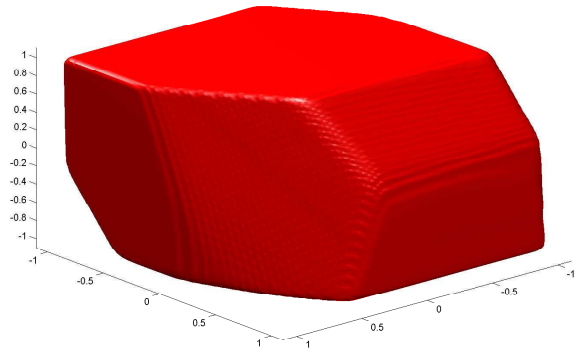

Fig. 4. Viability set for 3D

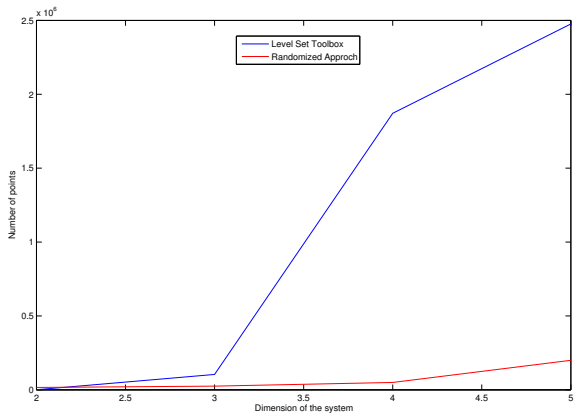

Fig. 5. Growth of number points respect the dimension

from the exact solution. And also we have computed the viability set with less accuracy (Method 2)within 5% of error using level set toolbox

	2D	3D	4D	5D
n Method 1	16000	25000	50000	200000
q Method 1	12	18	25	55
n Method 2	549	103823	1874161	2476099

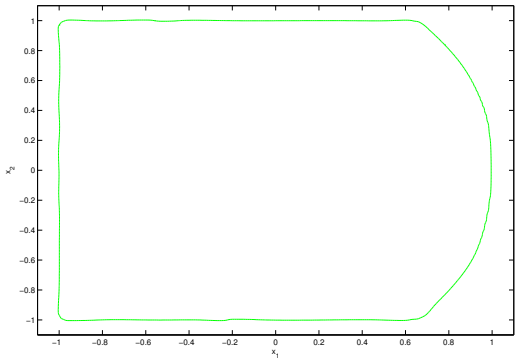

Fig. 6. Viability set for Nonlinear system

The neural network was trained with n as shown in the table for each system extracted using Halton distribution and K "bloated" by 10% in all directions, hence NN could learn "bad" and "good" set at the initialization.

4.2 Nonlinear system

To show that our technique works too for the case of nonlinear system, we have worked in simple nonlinear system defined as follow

$$\begin{aligned} \dot{x}_1 &= x_2^2 \\ \dot{x}_2 &= u \end{aligned}$$

with $u \in U = [-1, 1]$ and $x \in K = [-1, 1] \times [-1, 1]$.

We have computed an approximation of the viable set using 15 neurons and 17000 samples within 5% error from the "exact" solution. Which make clear that our method perform well in linear case as well in nonlinear case.

5. CONCLUSION

The aim of this paper was to propose an analytical approximation method that circumvent the cruse of dimensionality encountered in viability computations. We have used our algorithm on a set of different examples with good results. In current work we are trying to extended this proposed technique to more complicated system. A direct advantage of our method is provide us an analytical solution of the viability kernel. We are currently working to improve the structure of our neural networks to obtain better solution with less points.

The advantage introduced by the randomization technique have to be balance with the loss of the guarantee to obtaining exact solution. We are working on the convergence proof in order to provide statistical guarantee of the convergence. The proof is mix of uniform convergence empirical means and viability theory. We investigate to reduce computation time, we believe that jut better coding (e.g. programming in C) can bring down the computation time substantially.

ACKNOWLEDGEMENTS

Work supported by the European Commission under the HYCON Network of Excellence, IST-511368 and the HYGEIA project,NEST-4995.

REFERENCES

[1] J. P. Aubin. *Viability theory.* Bikhauser, Boston, 1991.
[2] C. M. Bishop. *Pattern Recognition and Machine Learning.* Springer, New York, 2006.
[3] P. Cardaliaguet, M. Quincampoix, and P. Saint-Pierre. Stochastic and differential games: Theory and numerical methods. In *Annals of the International Society of Dynamic Games.* Birkaser, 1999.
[Djeridane et al.] B. Djeridane, E. Cruck, and J. Lygeros. Randomized algorithm: A viability computation. In *submitted to IFAC08.*
[4] F.F. Foresee and M.T. Hagan. Gauss-newton approximation to bayesian regularization. *IJCNN*, (3):1930–1935, 1997.
[5] J.E. Gentle. *Random Number Generation and Monte Carlo Methods.* Springer-Verlag, New York, 1998.
[6] S. Haykin. *Neural Networks: A comprehensive foundation.* Prentice-Hall, New Jersey, 1999.
[7] P. F. Hokayem, C.T. Abdallah, and P. Dorato. Quasimonte carlo methods in robust control desgin. In *IEEE Mediterranean Conference on Control and Automation*, Rhodes, Greece, June 2003.
[8] J. Lygeros. On reachability and minimum cost optimal control. *Automatica*, 40:917–927, 2004.
[9] D.J. Mackay. A practical bayesian framework for backpropagation networks. *Neural Comp.*, (3):448–472, 1992.
[10] I. Mitchell. *A Toolbox of Level Set Methods version 1.1.* http://www.cs.ubc.ca/ mitchell/ToolboxLS, 2005.
[11] I. Mitchell, A.M. Bayen, and C. Tomlin. Validating a hamilton jacobi approximation to hybrid reachable sets. In M. DiBenedetto and A. Sangiovanni-Vincentelli, editors, *Hybrid systems: Computation and Control*, pages 418–432. Springer-Verlag, 2001.
[12] R. Tempo, G. Calafiore, and F. Dabbene. *Randomized Algorithms for Analysis and Control of Uncertain Systems.* Springer, New York, 2005.
[13] M. Vidyasagar. *A Theory of Learning and Generalization.* Springer-Verlag, 1997.
[14] T. Wong, W. Luk, and P. Heng. Sampling with hammersley and halton points. *Journal of Graphics Tools*, (2):9–24, 1997.