SVM Viability Controller Active Learning

Laetitia Chapel LAETITIA.CHAPEL@CEMAGREF.FR
Guillaume Deffuant GUILLAUME.DEFFUANT@CEMAGREF.FR
Cemagref - LISC, 24 rue des Landais BP 50085, 63172 Aubiere Cedex, France

Abstract

We use support vector machines (SVMs) to compute the actions which maintain a dynamical system within a defined subset of its state space. The principles of our method are inspired by the viability theory. We use SVMs to approximate the viability kernel which is the set of states from which it is possible to maintain the system. The actions to perform on the system can then be easily computed from the SVM, whatever the starting point. The major limitation of the approach is the exponentially growing number of training examples when the dimension of the state space increases. We use active learning to limit this number.

1. Introduction

We focus on a general control problem: how to maintain a dynamical system inside a given set of admissible states? Such a problem is frequent in ecology or economics, where the systems die or badly deteriorate when they leave some regions of the state space, and it can thus be called a viability problem. The viability theory (Aubin, 1991) provides several concepts and results which help to solve it. The set of admissible states K, is called the viability constraint set. The main concepts of the viability theory are:

- *Viable state*: A state is called *viable* if there exists at least one control function for which the whole trajectory from this state remains in K.
- *Viability kernel*: The set of all viable states is called the *viability kernel* and is denoted $Viab(K)$.

Aubin (1991) proved the viability theorems which enable to determine the viable states, without considering the combinatorial exploration of control actions series. These theorems also provide the control functions that maintain viability.

Saint-Pierre proposed an algorithm (Saint-Pierre, 1994) which computes the exact discrete viability kernel of the approximated discrete problem defined on a grid. It is very fast but its result is the set of points of the grid that are viable, which is not very convenient to manipulate. Moreover, using standard optimisation methods to compute the controls is not possible in this approach. When the control space is of high dimension, using such methods is necessary, because the computational time of an exhaustive search for a viable control grows exponentially with the dimension of the control space.

We proposed to use classification procedures in order to approximate the boundary of viability kernels. An important benefit is expected if the classification provides an explicit analytical expression of the viability kernel approximation, because it makes possible to use standard optimisation methods to compute the control. We established the mathematical conditions that the classification procedure should fulfill in order to guarantee the convergence to the actual viability kernel (Deffuant et al., submitted). We consider the support vector machines (SVMs) (Vapnik, 1995; Vapnik, 1998; Cristianini & Shawe-Taylor, 2000) as a particularly relevant classification procedure in this context. In particular, SVMs provide an analytical definition of the viability kernel approximation which is easy to use for optimizing the control, even on several time steps. Then the SVM can easily be used to control the system.

However, one important limitation of the approach remains: the number of examples to train the SVM grows exponentially with the dimension of the state space. This problem is general in optimal control problems, but it is even more crucial in our approach because of the time for training the SVM which grows quadratically with the number of training examples. Our general line of research is to apply active learning

Appearing in *Kernel machines for reinforcement learning* workshop, Pittsburgh, PA, 2006. Copyright 2006 by the author(s)/owner(s).

techniques to minimise the number of training examples. In this paper, we propose a first step in this direction. Instead of considering all the examples of the grid, we choose the ones which are close to the boundary, and have better chances to become support vectors. This approach allows us to use a number of training examples which approaches the one we get with one dimension less.

We propose some first experiments of the method on a simple dynamical system representing the evolution of a population on a limited space.

This paper is organized as follows: in the first section, we recall the principles for learning an SVM viability controller with a grid of points covering the constraint set. Then, we describe the active learning algorithm, and illustrate it on an example. Finally, we propose a discussion and draw some perspectives.

2. SVM viability controller trained on a complete grid

2.1. Viability theory

Consider a dynamical system which survives only inside a *viability constraint set* K, defined by its state $\vec{x}(t) \in \mathbb{R}^n$, and suppose that its evolution can be modified by controls $\vec{u}(t)$, in discrete time:

$$\begin{cases} \vec{x}(t+dt) = \vec{x}(t) + \varphi(\vec{x}(t), \vec{u}(t)).dt, \text{ for all } t \geq 0 \\ \vec{u}(t) \in U(\vec{x}(t)) \subset \mathbb{R}^q. \end{cases} \quad (1)$$

The viability problem is to determine a control function $t \rightarrow \vec{u}(t)$ which enables to keep the viability constraints satisfied indefinitely. Aubin (1991) defined the concept of a *viable state* for which such a function exists:

$$\begin{cases} \vec{x}(0) = \vec{x}_0 \\ \exists \vec{u}(.), \forall t \geq 0, \vec{x}(t) \in K. \end{cases} \quad (2)$$

The set of all the viable states is called the *viability kernel*, noted $Viab(K)$:

$$Viab(K) = \{\vec{x}_0 \in K, \exists \vec{u}(.), \forall t \geq 0, \vec{x}(t) \in K\}. \quad (3)$$

Viability kernels have interesting properties: equilibria, trajectories of periodic solutions, limit sets and attractors, if any, are all contained in the viability kernel. Moreover, it is shown that the viability kernel is instrumental to define viable control policies. The simplest rule (called heavy control) is to apply any control while the next time step is anticipated in the kernel, and to choose the first control which keeps the system in the viability kernel otherwise (we are sure that at least one exists). This procedure guarantees that the trajectory always remains in K from any state of Viab(K). We can easily derive more sophisticated control policies if the distance from a point to the boundary of the viability kernel can be computed.

The main task to solve a viability problem is therefore to determine its viability kernel. The viability theorems enable to determine viable states without considering the combinatorial exploration of control actions series over time. These theorems hold true for a large class of systems, called Marchaud systems: beyond imposing some weak technical conditions, the only severe restriction is that, for each state $\vec{x}$, the set of velocities $\varphi(\vec{x}, \vec{u})$ when $\vec{u}$ ranges over $U(\vec{x})$ is convex.

In general, there are no explicit formulas providing the viability kernel. The viability theorems are the basis for the algorithm proposed by Saint-Pierre (1994), which computes approximations of viability kernels. Considering a grid K_h of resolution h covering the admissible states, it determines a discrete approximation of the dynamical system. Then, the algorithm computes a series K_h^n of subsets of K_h, beginning with $K_h^0 = K_h$. At step $n+1$ the algorithm initializes $K_h^{n+1} = K_h^n$, and eliminates from K_h^{n+1} all the points which have none of their successors in K_h^n. It can be shown that after a finite number of steps p, we get $K_h^{p+1} = K_h^p$, and the algorithm stops.

Saint-Pierre shows that, under some conditions on φ, the finite discrete viability kernel tends to the viability kernel of the initial viability problem when the resolution h of the grid tends to 0.

The algorithm is very fast, but at each step the approximation of the viability kernel is defined as a discrete set of points, which not very convenient to handle (this set can be very large). Moreover it requires to test exhaustively all the possible controls because it is impossible to use standard optimisation methods to find a control which keeps the system inside K.

More recently, other techniques were tested to solve viability problems: the Ultra-bee Scheme (Bokanowski et al., to appear). The viability problem is considered as an optimal control problem, solved with a value function, with particular anti-diffusive techniques. This method shows good results but it is currently only defined for two-dimensional problems.

Our approach builds on Saint-Pierre's algorithm, and introduces SVMs to approximate the viability kernel.

2.2. SVM viability kernel approximation

We consider a grid K_h covering the viability constraint set K:

$$\forall \vec{x} \in K, \exists \vec{x}_h \in K_h \text{ such that } \parallel \vec{x} - \vec{x}_h \parallel < \beta(h). \quad (4)$$

We define the set-valued map $G : X \rightsquigarrow X$:

$$G(\vec{x}) = \{\vec{x} + \varphi(\vec{x}, \vec{u})dt \text{ for } \vec{u} \in U(\vec{x})\}. \quad (5)$$

We suppose that G is μ-Lipschitz with closed images, and we define the discrete differential inclusion:

$$\begin{cases} \vec{x}^{n+1} \in G(\vec{x}^n) \\ \vec{x}^0 = \vec{x}_0. \end{cases} \quad (6)$$

At each step n, we define a discrete set $K_h^n \subset K_h^{n-1} \subset K_h$ which contains all the viable states at the n^{th} iteration. This discrete set can be generalized by a continuous one, noted $L(K_h^n)$, which constitutes the current approximation of the viability kernel. This set is obtained by training an SVM on a learning set S, from the points $\vec{x}_h$ of the grid associated with label $+1$ if $\vec{x}_h \in K_h^n$ and with label -1 otherwise. The SVM provides a function f_n:

$$f_n(x) = \sum_{i=1}^{n} \alpha_i y_i k(x_i, x) + b. \quad (7)$$

where $k(.,.)$ is the kernel function used in the SVM. The support vectors are defined by $\alpha_i > 0$. The continuous set $L(K_h^n)$ is defined by:

$$L(K_h^n) = \{\vec{x} \in K \text{ such that } f_n(\vec{x}) \geq -\delta\}. \quad (8)$$

Parameter δ allows to dilate more or less the set $L(K_h^n)$, in order to fulfill the conditions of the algorithm's convergence.

Algorithm 1 shows the different steps of the approximation procedure.

Deffuant et al. (submitted) show that the convergence of the general algorithm is guaranteed when the resolution of the grid tends to 0, when the SVM function satisfies some conditions on the regularity of $L(K_h^n)$ and on the dilatation of the set $L(K_h^n)$.

The main advantage of using SVMs to approximate viability kernels is that it provides a analytical expression of the boundary of the kernel. This property can be used to find controls that keep the system inside the current viability kernel. For points that are not too far from the boundary of $L(k_h^n)$, the directions where $f_n(\vec{x})$ increases are going inside the current approximation and the directions where $f_n(\vec{x})$ decreases are going outside. Therefore, a way to determine if there is at least one control to allow to stay in the current viability kernel at the next step is to look for the control that maximises $f_n(\vec{x} + \varphi(\vec{x}, \vec{u})dt)$. To solve this problem, we use a gradient descent procedure because it is fast and easy to implement. Furthermore, instead of considering the evolution of a state at the next step, it is possible to consider a sequence of j times steps and to make the optimization to determine j optimal controls. We show on our example that increasing this number can considerably enhance the quality of the approximation.

Algorithm 1 SVM controller training on complete grid

```
K_h^0 = K_h
L(K_h^0) = K
S ← ∅
repeat
  for all x_h ∈ K_h do
    if d(G(x_h), L(K_h^n)) ≤ μβ(h) then
      S ← S + (x_h, +1)
    else
      S ← S + (x_h, −1)
    end if
  end for
  Compute f_{n+1}(x) from S
  Define L(K_h^{n+1}) and K_h^{n+1}
until K_h^n = K_h^{n+1}
return L(K_h^{n+1})
```

2.3. SVM viability controller

The idea of heavy control procedure comes from Aubin. Its principle is to change the action applied on the system only when it approaches the limit of the viability kernel. By definition, while this limit is not crossed, there always exists an action which maintains the system within it. The idea is here to change the action only when it is necessary. It is straightforward to adapt this idea when an analytical approximation of the viability kernel is available. We first define the functions of distance from a point inside $L(K_h^p)$ to the boundary of the viability kernel (noted $\partial Viab(K)$) and the boundary of its approximation (noted $\partial L(K_h^p)$):

$$v(\vec{x}) = d(\vec{x}, \partial Viab(K)). \quad (9)$$

$$a(\vec{x}) = d(\vec{x}, \partial L(K_h^p)). \quad (10)$$

For Δ a given positive real number, we define:

$$A_\Delta = \{\vec{x} \text{ such that } a(\vec{x}) \geq \Delta\}. \quad (11)$$

Considering an initial $\vec{x}_0 \in A_\Delta$, and a randomly chosen

control $\vec{u}_0 \in U(\vec{x})$, the procedure associates a control $\vec{u}_{n+1}$ at the $(n+1)^{th}$ iteration as follows:

- If $(\vec{x}_n + \varphi(\vec{x}_n, \vec{u}_n)dt) \in A_\Delta$, we keep the same control ($\vec{u}_{n+1} = \vec{u}_n$),
- Else, $\vec{u}_{n+1} = \arg\max_{\vec{u} \in U(x)} a\left(\vec{x}_n + \varphi(\vec{x}_n, \vec{u})dt\right)$.

In practice, we can define a more or less cautious controller, by anticipating on k time steps instead of one. Starting from $\vec{x}_n$, we check for $i = 1, .., k$ if applying k times the control $\vec{u}_n$, leads to a point $t(\vec{x}_n, \vec{u}_n, \vec{u}_n, ..., \vec{u}_n)$ which belongs to A_Δ. If it does, we move of one step (with a constant control). If not, we determine a sequence of controls that keep $t(\vec{x}_n, \vec{u}_{n+1}, \vec{u}_{n+2}, ..., \vec{u}_{n+k})$ inside A_Δ, and we apply the corresponding control $\vec{u}_{n+1}$.

The conditions in which this procedure guarantees to keep the system inside $Viab(K)$ are discussed in (Deffuant et al., submitted).

2.4. Application example

We use the Sequential Minimal Optimization algorithm to compute the SVMs, because it has the good property to require a memory space growing linearly with the sample size (Platt, 1998). We consider a gaussian kernel:

$$K(\vec{x}_1, \vec{x}_2) = exp\left(-\frac{\|\vec{x}_1 - \vec{x}_2\|^2}{2\sigma^2}\right) \tag{12}$$

and we use the library LIBSVM (Chang & Lin, 2001), which implements a SMO-type algorithm, to compute the SVM.

We consider a simple dynamical system of population growth on a limited space. The state $(x(t), y(t))$ of the system represents the size of a population $x(t)$, which grows or diminishes with the evolution rate $y(t)$. The population must remain in an interval $K = [a, b]$, with $a > 0$. The dynamical system was studied by Maltus and later on by Verhulst, and then redeveloped by Aubin (2002) with an inertia bound. The inertia bound c limits the derivative of the evolution rate at each time step. The system in discrete time defined by a time interval dt can be written as follows:

$$\begin{cases} x(t+dt) = x(t) + x(t)y(t)dt \\ y(t+dt) = y(t) + u(t)dt \\ \text{with } -c \leq u(t) \leq +c. \end{cases} \tag{13}$$

It is possible to derive analytically the viability kernel of this problem (Aubin, 2002), it has the following expression (for $dt \to 0$):

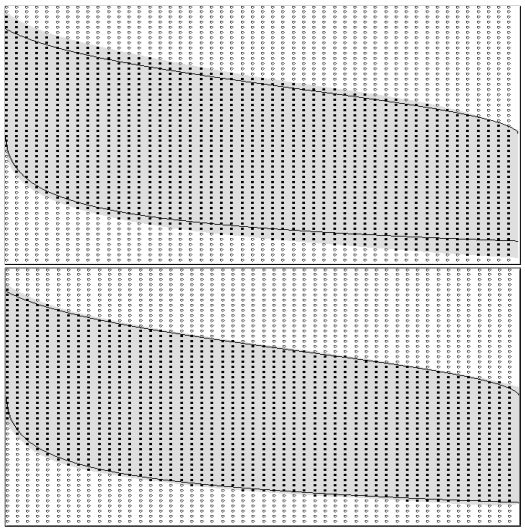

Figure 1. Final approximation using a whole grid of 2601 points (51 points by dimension). Above, the optimisation is made on 2 time steps, and below on 8 time steps. dt is computed for defining moves of size $\beta(h)$ in one time step.

$$\text{Viab}(K) = \{(x, y) \text{ such that:} \begin{cases} x \in [a, b] \text{ and} \\ y \in \left[-\sqrt{2c\log\left(\frac{b}{x}\right)}, \sqrt{2c\log\left(\frac{x}{a}\right)}\right] \end{cases}\}. \tag{14}$$

Figure 1 shows the result of the viability kernel approximation with an optimisation on different number of time steps.

Figure 2 shows an example of the functioning of the controller for the population problem in dimension 2. We start from the approximation of the viability kernel given by Figure 1 with 8 time steps. We put $\Delta = 12$ and we compare two types of controller: a first one, anticipating on 2 time steps, and a more cautious one, with 8 time steps anticipation.

This example illustrates some advantages of SVM viability controllers: good quality of the approximation, facility to compute the control actions. However, a strong limitation remains: the SVM is trained on all the examples of the grid, and their number grows exponentially with the dimension of the state space. To try to decrease this number, we consider an active learning approach.

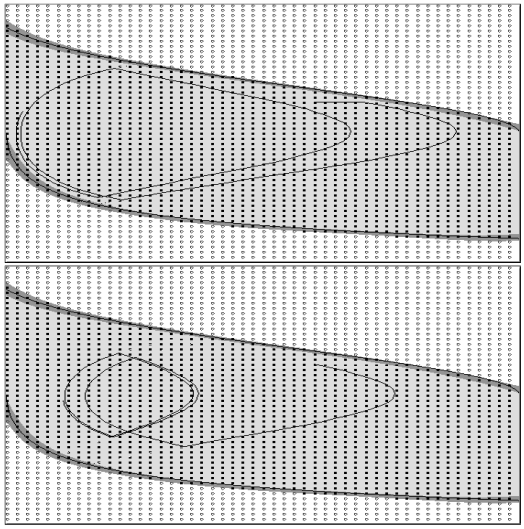

Figure 2. Example of heavy trajectories from a point in A_Δ. The difference between $L(K_h^p)$ and A_Δ is in dark gray. The trajectories include 200 time steps, and start with the point located close to the upper border of the viability kernel. Above: 2 time steps anticipation. Below: 8 time steps anticipation.

3. SVM viability controller active learning

3.1. Principles

In many supervised tasks, the cost of labeling instances is important. Active learning tries to minimize the number of labeled examples and actively choose the training data. Support Vector Machines have received much attention in this context (Tong & Chang, 2001; Schohn & Cohn, 2000), because they provide a subset of the most informative points, the support vectors, to define the separating surface. The SVM active learner thus attempts to minimize the number of the non-support vectors labeled. Schohn and Cohn (2000) propose a simple heuristic to choose the instances, based on their proximity to the SVM separating surface. They show that this approach frequently significantly decreases the computing time.

In our problem of learning a viability controller, labeling instances is time consuming, because it requires to optimise the control. But the main problem is to limit as much as possible the number of instances used to train the SVM, because to take them all becomes rapidly impossible when the dimension of the state space increases. In this section, we propose to select the most "informative" instances, in accordance with the concepts of SVM active learning. Moreover, approximating a viability kernel is a very specific problem, and some heuristics to choose the most interesting instances can easily be elaborated.

3.1.1. What are the states that must be labeled?

To compute $L(K_h^{n+1})$, one must label the points that are likely to leave the set K_h^n at the next j time steps whatever the applied control. These points are necessarily located inside L_h^n and close to $\partial L(K_h^n)$, because there is a maximum distance that the system can move during a fixed number of time steps. It is useless to compute the label of the other points, because one can be sure that it won't change. To identify the interesting points, we calculate the distance between a point and $\partial L(K_h^n)$ and compare it with the maximal distance of a move. Let us define γ this maximal move:

$$\gamma = j \times dt \tag{15}$$

where j is the number of time steps and dt the time step.

3.1.2. What are the points that can be kept in the training set?

The support vectors defining $L(K_h^{n+1})$ are always among the points of the grid which are the closest to $\partial L(K_h^{n+1})$. It is therefore enough to select the examples for which the label would change if they were translated of $2\beta(h)$ in the direction normal to $\partial L(K_h^{n+1})$. Since $L(K_h^{n+1})$ is not determined yet, we approximate this direction by the direction normal to $\partial L(K_h^n)$. In order to find these points, we start from points x_h inside $L(K_h^n) + 2\beta(h)\mathbf{B}$ ($\mathbf{B}$ being the ball of radius 1) which are at a maximum distance γ of $\partial L(K_h^n)$, and with a negative label. Then we translate them at a distance of $2\beta(h)$, in the direction of the gradient of f_n. This defines the point $g(\vec{x}_h)$:

$$g(\vec{x}_h) = \vec{x}_h - \frac{\nabla_x f_h^n(\vec{x}_h)}{\|\nabla_x f_h^n(\vec{x}_h)\|} \times 2\beta(h) \tag{16}$$

If the label of $g(\vec{x}_h)$ is positive, it means that the boundary of $L(K_h^{n+1})$ lies between $\vec{x}_h$ and $g(\vec{x}_h)$, and we store $\vec{x}_h$ and $g(\vec{x}_h)$ into the learning set.

3.2. Active learning algorithm

Algorithm 2 presents the implementation of the above described principles. S is the set of SVM training examples.

This algorithm selects all the negative examples of the grid which are closer than $2\beta(h)$ of $\partial L(K_h^{n+1})$, and adds to each of them a positive example located closer than $2\beta(h)$. This training set covers locally the space around $\partial L(K_h^{n+1})$, with a resolution of $2\beta(h)$. The effect of this training set is the same as the one of the

Algorithm 2 SVM controller active learning

```
repeat
  S ← ∅
  for all x_h ∈ K_h^n + 2β(h)B such that d(x_h, ∂L_h^n) ≤ γ do
    if d(G(x_h), L(K_h^n)) > μβ(h) then
      Compute g(x_h)
      if d(G(g(x_h)), L(K_h^n)) ≤ μβ(h) then
        S ← S + (x_h, −1)
        S ← S + (g(x_h), +1)
      end if
    end if
  end for
  Compute f_{n+1} on S
  Define L(K_h^{n+1}) and K_h^{n+1}
until K_h^n = K_h^{n+1}
```

whole grid, except that the SVM is not constrained on the borders of K, and can thus make undesired classifications in these unconstrained areas. It is then necessary to check that we make no mistakes for the points lying on the boundary of K. If there are misclassified points, we add them to the training set and redefine $L(K_h^{n+1})$ until there is no error.

At the first iteration of the algorithm, no SVM is available yet and we label the points which are at a distance lower than γ from the border of K. To compute $g(\vec{x}_h)$, we take the direction normal to the boundary of K.

3.3. Application example

We consider the same dynamical system as previously.

Figure 3 presents the last steps of an example of progressive approximation of the viability kernel. In gray, we represent the current viability kernel $L(K_h^n)$ and the points are those used to train $L(K_h^{n+1})$. The last figure represents the final kernel approximation, using the same parameters than Figure 1 with 8 time steps. With the active learning, we use only 234 points at the maximum (11% of the size of the grid). The final approximation of the viability kernel is very similar to the one obtained with the whole grid.

The population model can be artificially extended to larger dimensions: In dimension 4 for instance, we consider points of the space defined as: $\vec{x} = (x_1, x_2, x_3, y)$, with dynamics defined only on components x_1 and y:

$$\begin{cases} x_1(t+dt) = x_1(t) + x_1(t)y(t)dt \\ x_2(t+dt) = x_2(t) \\ x_3(t+dt) = x_3(t) \\ y(t+dt) = y(t) + u(t)dt \end{cases} \quad (17)$$
$$\text{with } -c \leq u(t) \leq +c.$$

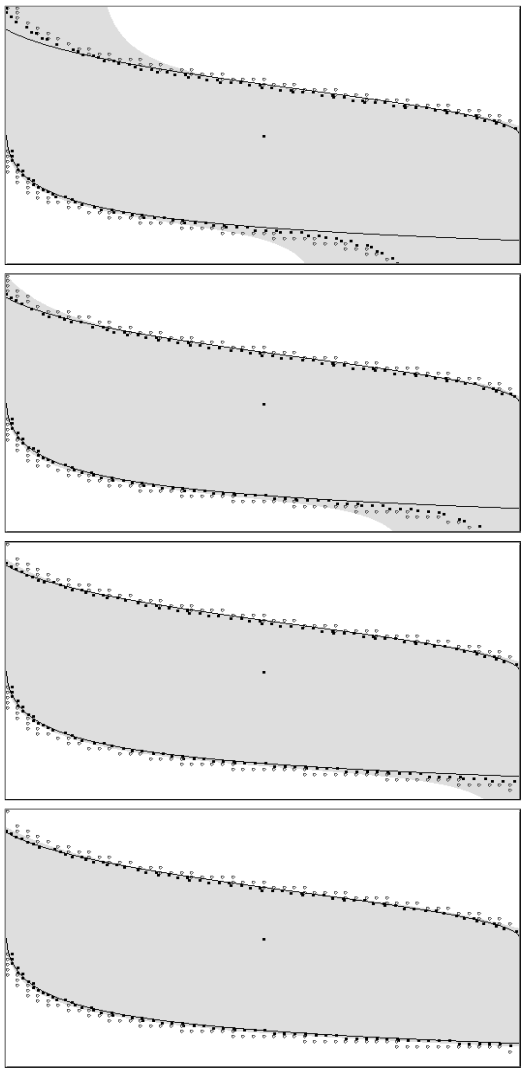

Figure 3. Progressive approximation of the viability kernel using the SVM controller active learning. The last figure represents the final approximation.

Figure 4 presents a projection of the final approximation of the viability kernel for the problem in dimension 4. With the active learning, we use only 26% of the size of the grid.

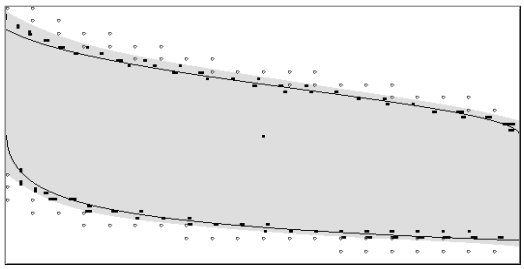

Figure 4. Approximation of the viability kernel for the population problem in dimension 4. The horizontal axis represents variable x_1, the vertical axis y, $x_2 = 3$ and $x_3 = 3$. The total grid includes 194,481 points and we use only $51,370$ points at the maximum. dt is computed for defining moves of size $2\beta(h)$ in one time step and the optimisation is made on 4 time steps.

4. Discussion

SVM viability controllers show interesting features. They enable to use optimisation methods to define the controls, which opens the possibility to deal with larger dimensionality control spaces. Moreover, SVM properties can be used to minimise the number of training examples in an active learning approach. We presented a first step in this direction, which builds a training set locally surrounding the boundary of the next viability kernel approximation. This method allows us to get a number of examples corresponding to a problem of one dimension less. In the future, we plan to reduce more the training set, and to make several trainings on closer and closer training examples. Our goal is to use training sets which are not much larger than the final number of support vectors. This could allow us to tackle problems in significantly larger dimensions, because we noted that the final number of support vectors is generally very small compared to the total number of points in the grid.

References

Aubin, J.-P. (1991). *Viability theory*. Birkhäuser.

Aubin, J.-P. (2002). Elements of viability theory for the analysis of dynamic economics. *Ecole thématique du CNRS 'Economie Cognitive'*.

Bokanowski, O., Martin, S., Munos, R., & Zidani, H. (to appear). An anti-diffusive scheme for viability problems. *Applied Mathematics and Numerics*.

Chang, C.-C., & Lin, C.-J. (2001). *LIBSVM: a library for support vector machines*. Software available at `http://www.csie.ntu.edu.tw/~cjlin/libsvm`.

Cristianini, N., & Shawe-Taylor, J. (2000). *Support vector machines and other kernel-based learning methods*. Cambridge University Press.

Deffuant, G., Martin, S., & Chapel, L. (submitted). Approximating viability kernel with support vector machines. Submitted on IEEE Transactions on Automatic Control.

Platt, J.-C. (1998). *Fast training of support vector machines using sequential minimal optimization* (Technical Report). 98-14, Microsoft Research, Redmond, Washington.

Saint-Pierre, P. (1994). Approximation of the viability kernel. *Applied Mathematics & Optimization*, *29*, 187–209.

Schohn, G., & Cohn, D. (2000). Less is more: Active learning with support vector machines. *ICML '00: Proceedings of the Seventeenth International Conference on Machine Learning* (pp. 839–846). San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.

Tong, S., & Chang, E. (2001). Support vector machine active learning for image retrieval. *MULTIMEDIA '01: Proceedings of the ninth ACM international conference on Multimedia* (pp. 107–118). New York, NY, USA: ACM Press.

Vapnik, V. (1995). *The nature of statistical learning theory*. Springer Verlag.

Vapnik, V. (1998). *Statistical learning theory*. Wiley.