# Approximating Viability Kernels With Support Vector Machines

Guillaume Deffuant, Laetitia Chapel, and Sophie Martin

*Abstract*—**We propose an algorithm which performs a progressive approximation of a viability kernel, iteratively using a classification method. We establish the mathematical conditions that the classification method should fulfill to guarantee the convergence to the actual viability kernel. We study more particularly the use of support vector machines (SVMs) as classification techniques. We show that they make possible to use gradient optimisation techniques to find a viable control at each time step, and over several time steps. This allows us to avoid the exponential growth of the computing time with the dimension of the control space. It also provides simple and efficient control procedures. We illustrate the method with some examples inspired from ecology.**

*Index Terms*—**Dynamical systems, optimal control, support vector machines (SVMs), viability kernel.**

## I. INTRODUCTION

Viability theory [1] aims at controlling dynamical systems with the goal to maintain them inside a given set of admissible states $K$, here called the viability constraint set. 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 [2]–[4]. The approach can be also adapted to robotics and control in general [5]–[7].

The main concepts of the viability theory are as follows.

- *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 $K$ indefinitely.
- *Viability kernel*: The set of all viable states is called the *viability kernel* and is denoted $\text{Viab}(K)$.

Aubin [1] proved the viability theorems which enable to determine viable states, without considering the combinatorial exploration of control actions series. These theorems also provide the control functions that maintain viability. The simplest is the "heavy" control procedure, which we will use later on in this note. This procedure specifies to change the control only if the system reaches the boundary of the viability kernel, and to choose the first control which keeps the system inside the kernel (by theorem, we are sure that such a control exists). Such a procedure is also particularly relevant to assisted control problems [5]: The control is corrected automatically only when the user control would lead the system to cross the viability kernel boundary.

References [5] and [6] suppose the availability of a procedure stating if a given state is in the viability kernel or not. They generate a set of state examples, associated with $+1$ when the state is viable, and $-1$ otherwise. Then they use a learning procedure (nearest neighbours or SVMs) to approximate the viability kernel. However, in general, stating directly if a given state is viable or not is computationally untractable, especially when the control space is of large dimension. It requires a combinatorial search over a large number of time steps (the time horizon), which is exponential in nature. The method adopted in [7], defining a continuous barrier function on the constraint set works

probably well when the dynamics is not too complex, but in general, it gives no guarantee of good control, because the shape of the barrier can be quite different from the one of the actual viability kernel.

In this note, we propose an algorithm which, given the dynamics of the system and the viability constraints, provides an approximation of a viability kernel with a reasonable computational effort, using a classification procedure. Our method builds on Saint-Pierre's algorithm [8] which computes the exact discrete viability kernel of the approximated discrete problem defined on a grid.

We consider the support vector machines (SVMs) [9], [10] as a particularly relevant classification procedure in this context. Their main interest is that they directly provide a kind of barrier function, which can be used to search for viable controls with a gradient optimisation procedure. Such procedures are much less time consuming than the systematic search performed by Saint-Pierre, which can only be limited to very small dimension control spaces. Moreover, it gives the possibility to optimise over several time steps, without an exponential growth of the computing time. We propose some first experiments of the method on a simple dynamical system representing the evolution of a population on a limited space. The first results show that the optimisation over several time steps significantly improves the viability kernel approximation, for a given grid resolution. Moreover, we briefly describe an example in which the method gives satisfactory results with a control space of dimension 51.

In Section II, we express the new algorithm of viability kernel approximation, using a classification procedure, and we state the conditions that such a classification procedure should fulfill. In Section III, we consider the use of SVMs, describe the method allowing to optimise controls over several time steps and the associated "heavy" control procedure. Section IV reports the results of a set of experiments in different dimensions. Finally, we discuss the results and draw some perspectives.

## II. APPROXIMATING A VIABILITY KERNEL WITH A CLASSIFICATION METHOD

### A. Notations

We consider a dynamical system defined by its state $\vec{x}(t) \in X$ and assumes that its evolution can be influenced by a control $\vec{u}(t)$

$$\begin{cases} \vec{x}'(t) = \varphi\left(\vec{x}(t), \vec{u}(t)\right) & \text{(1a)} \\ \vec{u}(t) \in U\left(\vec{x}(t)\right). & \text{(1b)} \end{cases}$$

We suppose $X \subset R^n$. The set of available controls depends on the state, $\vec{u}(t)$ is chosen in a subset $U(\vec{x}(t)) \subset R^q$. We consider a viability constraint set $K$, a compact subset of $X$, in which we want to maintain the system. The subset $\text{Viab}(K)$ of all viable states $x_0 \in K$ is called the *viability kernel* of $K$:

$$\text{Viab}(K) = \left\{\vec{x}_0 \in K, \exists \vec{u}(.), \forall t \geq 0, \vec{x}(t) \in K\right\}. \quad (2)$$

We discretise the dynamical system in time. We consider a given time interval $dt$, and we define the set-valued map $G: X \rightsquigarrow X$

$$G(\vec{x}) = \left\{\vec{x} + \varphi(\vec{x}, \vec{u})dt, \text{ for } \vec{u} \in U(\vec{x})\right\}. \quad (3)$$

We suppose that $G$ is $\mu$-Lipschitz with closed images. We would like to approximate the viability kernel $\text{Viab}_G(K)$ of $K$ under the discrete dynamical system defined by $G$, which is the subset of $K$ from which

it is possible to maintain the system indefinitely inside $K$. The viability theorems show that $\text{Viab}_G(K)$ is the largest subset $E$ of $K$ such that

$$\forall \vec{x} \in E, G(\vec{x}) \cap E \neq \emptyset. \qquad (4)$$

We define a grid $K_h$ as a finite set of elements of $K$, such that:

$$\forall \vec{x} \in K, \exists \vec{x}_h \in K_h, \text{ such that } \|\vec{x} - \vec{x}_h\| \leq \beta(h) \qquad (5)$$

and $\beta(h) \to 0$ when $h \to 0$. Such a grid exists since $K$ is compact.

The algorithm presented in the next section proceeds in several steps, progressively defining the viability kernel approximation. At each step $n$, we define a discrete set $K_h^n \subset K_h^{n-1} \subset K_h$, and a continuous set, noted $L(K_h^n)$, which is a generalization from this discrete set, and which constitutes the current approximation of the viability kernel.

Moreover, we use the following notations.

- $l$ is a classifier learning procedure which associates a set $S$ of training tuples $(\vec{x}_i, y_i) \in K \times \{-1, 1\}$ with a classification function $l_S(x) : K \to \{-1, 1\}$.
- $d(E, F)$ is the distance between two closed subsets $E$ and $F$.
- $E \setminus F$ is the complementary set of $F$ in $E$ (supposing that $F \subset E$).
- $\mathbf{B}$ is the ball of center 0 and radius 1.

### B. The Algorithm of Viability Kernel Approximation Using a Classification Procedure

The steps of the algorithm are the following.

- Initialize the sets $K_h^0 = K_h$ and $L(K_h^0) = K$.
- Iterate:
  — Define the discrete set $K_h^{n+1}$, from $K_h^n$ and $L(K_h^n)$ as follows:

$$K_h^{n+1} = \left\{ \vec{x}_h \in K_h^n, \text{ such that } d\left(G(\vec{x}_h), L(K_h^n)\right) \leq \mu\beta(h) \right\}. \quad (6)$$

  — If $K_h^{n+1} \neq K_h^n$ then run the classification procedure $l$ on the learning sample obtained with the points $\vec{x}_h$ of the grid $K_h$, associated with label $+1$ if $\vec{x}_h \in K_h^{n+1}$, and with label $-1$ otherwise. Let $l_h^{n+1}$ be the obtained classification function from $K$ to $\{-1, 1\}$. $L(K_h^{n+1})$ is defined as follows:[1]

$$L\left(K_h^{n+1}\right) = \left\{ \vec{x} \in K, \text{ such that } l_h^{n+1}(\vec{x}) = +1 \right\}. \quad (7)$$

  — Else, stop and return $L(K_h^n)$.

### C. Convergence of the Algorithm

The convergence of the algorithm to the actual viability kernel as the resolution $h$ tends to 0 is guaranteed when the classification procedure satisfies some conditions which are specified in the following theorem. These conditions express that any point of $L(K_h^n)$ must be close to one point of $K_h$ with a positive label, and any point of $K \setminus L(K_h^n)$ must be close to a point of $K_h$ with a negative label. The constraint on the distance to the point with a negative label is stricter because if $L(K_h^n)$ is smaller than $\text{Viab}(K)$, the mistake cannot be fixed in the next iterations (whereas it can be fixed when the mistake is on the other side).

*Theorem 1:* If there exists a real $\lambda \geq 1$ such that for any iteration $n$, the approximation $L(K_h^n)$ satisfies the following conditions:

$$\forall \vec{x} \in L\left(K_h^n\right) \quad d\left(\vec{x}, K_h^n\right) \leq \lambda\beta(h) \qquad (8)$$

[1]We suppose that $L(K_h^{n+1})$ is a closed set without loss of generality, because otherwise we define it as the closure of the set defined in (7).

$$\forall \vec{x} \in K \setminus L\left(K_h^n\right) \quad d\left(\vec{x}, K_h \setminus K_h^n\right) \leq \beta(h) \qquad (9)$$

then, the algorithm of viability kernel approximation provides a result which converges to the actual viability kernel when the resolution $h$ of the grid tends to 0.

*Proof:* The proof of convergence involves four steps.

1) The algorithm stops after a finite number of steps $p$.
   At each step, we have: $K_h^{n+1} \subset K_h^n$, by construction. Because $K_h$ is finite, we shall get, after a finite number of steps $p$: $K_h^{p+1} = K_h^p$.

2) For all $n$, the viability kernel of $K$ under $G$ is included in $L(K_h^n)$.
   We have $\text{Viab}_G(K) \subset K = L(K_h^0)$. Suppose that $\text{Viab}_G(K) \subset L(K_h^n)$. Consider $\vec{x} \notin L(K_h^{n+1})$.
   - If $\vec{x} \notin L(K_h^n)$ then $\vec{x}$ is not viable, by hypothesis.
   - If $\vec{x} \in L(K_h^n)$, because of condition 9, we have:

$$\exists \vec{x}_h \in K_h \setminus K_h^{n+1} \text{ such that } \|\vec{x} - \vec{x}_h\| \leq \beta(h). \qquad (10)$$

   Moreover, by construction of the algorithm, we have:

$$\vec{x}_h \in K_h \setminus K_h^{n+1} \Rightarrow d\left(G(\vec{x}_h), L\left(K_h^n\right)\right) > \mu\beta(h). \qquad (11)$$

   Because $G$ is $\mu$-Lipschitz:

$$G(\vec{x}) \subset \left(G(\vec{x}_h) + \mu\beta(h)\mathbf{B}\right). \qquad (12)$$

   Therefore, $G(\vec{x}) \subset (K \setminus L(K_h^n))$. By hypothesis, any point of $K \setminus L(K_h^n)$ is not viable, therefore all the successors of $\vec{x}$ are not viable, which implies that $\vec{x}$ is not viable.
   Therefore, $\vec{x} \notin L(K_h^{n+1}) \Rightarrow \vec{x}$ not viable, thus $\text{Viab}_G(K) \subset L(K_h^{n+1})$.

3) The result of the algorithm, $L(K_h^p)$, is included in the viability kernel of $K$ under $G + \mu(1 + \lambda)\beta(h)\mathbf{B}$.
   Let $G_h : X \to X$, such that $G_h(\vec{x}) = G(\vec{x}) + \mu(1 + \lambda)\beta(h)\mathbf{B}$. $G_h$ has closed images. Consider a point $\vec{x}$ of $L(K_h^p)$. Because of condition 8:

$$\exists \vec{x}_h \in K_h^p \text{ such that } \|\vec{x} - \vec{x}_h\| \leq \lambda\beta(h). \qquad (13)$$

   Since, $K_h^{p+1} = K_h^p$, $\vec{x}_h \in K_h^{p+1}$ and, by definition of the algorithm

$$\vec{x}_h \in K_h^{p+1} \Rightarrow d\left(G(\vec{x}_h), L\left(K_h^p\right)\right) \leq \mu\beta(h). \qquad (14)$$

   Because $G$ is $\mu$-Lipschitz, and using the triangular inequality:

$$d\left(G(\vec{x}), L\left(K_h^p\right)\right) \leq \mu(1 + \lambda)\beta(h). \qquad (15)$$

   This implies that $G_h(\vec{x}) \cap L(K_h^p) \neq \emptyset$. Therefore, from any $\vec{x} \in K_h^p$, there exists a trajectory which remains indefinitely in $K_h^p$. Therefore, $K_h^p \subset \text{Viab}_{G_h}(K)$.

4) Conclusion.
   Since $G$ is $\mu$-Lipschitz and $K$ is compact,

$$\forall \epsilon > 0, \exists \eta > 0 \text{ such that } h < \eta$$
$$\Rightarrow \text{Viab}_{G_h}(K) \subset \left(\text{Viab}_G(K) + \epsilon\mathbf{B}\right). \quad (16)$$

Moreover, for all $h > 0$, $\text{Viab}_G(K) \subset L(K_h^p) \subset \text{Viab}_{G_h}(K)$. Therefore

$$L\left(K_h^p\right) \to \text{Viab}_G(K) \text{ when } h \to 0. \qquad (17)$$

$\blacksquare$

### III. SVMs AS PARTICULARLY RELEVANT CLASSIFICATION PROCEDURES

#### A. Presentation of SVMs

Here, we review some important features of the SVMs. For more details; see, for instance, [10].

We consider a set of examples $\{(\vec{x}_i, y_i)\}_{i=1}^{N}$, where $\vec{x}_i$ is a real vector, and $y_i \in \{-1, 1\}$ is the label.

SVMs define separations between the examples of each label. To introduce some nonlinearity in the separating function, we project the examples into a feature space ($\psi(\vec{x})$ denotes the projection of $\vec{x}$ into the feature space). Fortunately, this function does not need to be explicit; the kernel[2] $k$, defining a scalar product of two projections into the feature space ($k(\vec{x}, \vec{y}) = \langle \psi(\vec{x}), \psi(\vec{y}) \rangle$), is sufficient to make all computations. The SVM is obtained by solving a quadratic problem with linear constraints.

The solution of the quadratic problem defines the function $f$, which is then used for the classification: A point $\vec{x}$ is labelled $+1$, if $f(\vec{x})$ is positive, $-1$, otherwise. The expression of $f$ is:

$$
\begin{aligned}
f(\vec{x}) &= \langle \vec{w}, \psi(\vec{x}) \rangle + b \\
&= \sum_{i=1}^{N} \alpha_i y_i k(\vec{x}_i, \vec{x}) + b, \text{ with } \alpha_i \geq 0, 1 \leq i \leq N.
\end{aligned}
\tag{18}
$$

The parameters $\alpha_i$ come from the lagrangian expression of the problem. The points $\vec{x}_i$ for which $\alpha_i > 0$ are the support vectors. They are sufficient to define the function. When $k$ is non linear, the resulting function is also non linear. It is particularly relevant in our context to use a gaussian kernel $k$ defined in (19) because it leads to SVMs which can approximate any classification function:

$$
k(\vec{x}, \vec{y}) = \exp\left(\frac{-\|\vec{x} - \vec{y}\|^2}{2\sigma^2}\right).
\tag{19}
$$

The practical use of SVM with a gaussian kernel requires the values of two parameters. We will use such a function to define the continuous sets $L(K_h^n)$.

#### B. Fulfillment of the Theorem Conditions

We have no rigorous demonstration that SVM classifiers fulfill the theorem conditions. However, in practice, in the examples we tested, we easily found SVM parameters leading to a boundary which makes no classification error, and has no irregularities higher than $\beta(h)$, which implies that the conditions are met. Some arguments let us think that it should generally be the case. First consider the nearest neighbour (NN) classification procedure: This procedure attributes to point $\vec{x}$ the label ($+1$ or $-1$) of its closest neighbour of $K_h$. It can be easily shown that this method fulfills conditions (8) and (9) with $\lambda = 1$. If $L(K_h^n)$ is defined as the subset of $K$ which are given a positive label with the NN procedure, then by definition, each point of $L(K_h^n)$ will be closer than $\beta(h)$ to a point of $K_h^n$, and each point of $K \setminus L(K_h^n)$ will be also closer than $\beta(h)$ to a point of $K_h \setminus K_h^n$. Now, consider the SVM function $f_n$ obtained from the learning set defined by the discrete set $K_h^n$. With a Gaussian kernel, the feature space is of infinite dimension, and it is always possible to find parameter values that will lead to no classification errors. As soon as $\sigma \geq \beta(h)$, the boundary of the SVM classification will be smoother than the one of the NN classifier. In this

[2] In this section, the term "kernel" refers to a SVM kernel, a distinct and unrelated meaning from when used elsewhere to refer to a viability kernel.

case, if we define the approximation $L(K_h^n)$ from the sample defined by $K_h^n$ as:

$$
L(K_h^n) = \{\vec{x} \in K, \text{ such that } f_n(\vec{x}) \geq 0\}.
\tag{20}
$$

Conditions (8) and (9) are fulfilled with $\lambda = 1$. One may wonder why not using the NN classifier, instead of SVMs. One important reason is that SVMs are more efficient classifiers, especially in high dimension. The other reason is that SVMs provide easy means to optimise the control by a gradient ascent, and therefore to avoid the exponential growth of the computational time when the dimension of the control space increases.

#### C. Optimizing the Control by a Gradient Ascent

The analytical expression of function $f_n$ can be used to find controls which keep the system inside the current kernel approximation. In the neighbourhood of the boundary of $L(K_h^n)$, the directions where $f_n(\vec{x})$ increases are going inside $L(K_h^n)$, and the directions where $f_n(\vec{x})$ decreases are going outside $L(K_h^n)$. Therefore, $f_n(\vec{x})$ provides directly a kind of barrier function, similar to the one used in [7], except that the barrier is not on the boundary of $K$, but on approximations of the viability kernel (and is not based on a logarithmic function). In this context, maximising $f_n(\vec{x})$ provides a control that keeps the system inside $L(K_h^n)$, if this control exists.

It is also possible to determine a sequence of optimal controls on $j$ time steps in a reasonable computational time. Let us first define $t(\vec{x}, \vec{u}_1, \ldots, \vec{u}_j)$, the point reached after $j$ time steps

$$
\begin{cases}
t(\vec{x}, \vec{u}_1) = \vec{x} + \varphi(\vec{x}, \vec{u}_1) dt, & (21a) \\
t(\vec{x}, \vec{u}_1, \ldots, \vec{u}_j) = t(\vec{x}, \vec{u}_1, \ldots, \vec{u}_{j-1}) \\
\quad + \varphi\left(t(\vec{x}, \vec{u}_1, \ldots, \vec{u}_{j-1}), \vec{u}_j\right) dt. & (21b)
\end{cases}
$$

We consider a point $t(\vec{x}, \vec{u}_1, \ldots, \vec{u}_j)$ which is out of $L(K_h^n)$, but in the neighbourhood of the boundary. For instance we consider $0 > f_n(t(\vec{x}, \vec{u}_1, \ldots, \vec{u}_j)) > -1$. Then, we perform a gradient ascent, until either we find controls that put the system back inside $L(K_h^n)$, or we reach the maximum. The gradient ascent is done as follows. Let $\eta$ be a parameter ($0 < \eta < 1$). We initialise $(\vec{u}_1^0, \ldots, \vec{u}_j^0) = (\vec{u}_1, \ldots, \vec{u}_j)$, and then we iterate

$$
\left(\vec{u}_1^{k+1}, \ldots, \vec{u}_j^{k+1}\right) = \left(\vec{u}_1^k, \ldots, \vec{u}_j^k\right) + \eta \nabla_u f_n\left(t\left(\vec{x}, \vec{u}_1^k, \ldots, \vec{u}_j^k\right)\right).
\tag{22}
$$

The necessary derivatives can be directly approximated, or analytically computed when the derivatives of $\varphi$ are easy to express. This procedure must take into account the fact that the controls must remain in set $U(\vec{x})$. The maximum is reached either when the gradient is null, or when it is normal to the boundary of $U(\vec{x})$. At the first time step of the viability kernel approximation, i.e. when there is no SVM yet, we use a similar procedure on a barrier function on the boundary of $K$, as in [7]. This procedure allows us to deal with problems with controls in high dimension, which is not possible with Saint-Pierre's algorithm, or with the method proposed in [5]. Moreover, in the following experiment, we show that considering several time steps of controls can significantly improve the viability kernel approximation.

#### D. SVM Heavy Controller

The principle of heavy control [5], [11] 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. Again, we use the property that $f(x)$, the function associated with the SVM, can play the role of a barrier function in the neighbourhood of the viability kernel approximation boundary. More precisely, the procedure is as follows:

*Procedure:* For $\Delta$ a given positive real number, we define:

$$A_\Delta = \{\vec{x}, \text{ such that } f(\vec{x}) \geq \Delta\}.$$

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 ($\vec{u}_{n+1} = \vec{u}_n$),
- Otherwise, $\vec{u}_{n+1} = \arg \max_{\vec{u} \in U(x)} f(\vec{x}_n + \varphi(\vec{x}_n, \vec{u})dt)$.

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 $i = 1, \ldots, k$ if applying $k$ times the control $\vec{u}_n$, leads to a point $t(\vec{x}_n, \vec{u}_n, \vec{u}_n, \ldots, \vec{u}_n)$ which belongs to $A_\Delta$. If it does, we move of one step with $u_{n+1} = u_n$. If not, we determine a sequence of controls that keep $t(\vec{x}_n, \vec{u}_{n+1}, \vec{u}_{n+2}, \ldots, \vec{u}_{n+k})$ inside $A_\Delta$, and we apply the corresponding control $\vec{u}_{n+1}$.

We are not able yet to provide precise mathematical conditions which guarantee that this control scheme keeps the system in $K$. On the example presented later, it appears that using several time step can considerably enhances the chances to remain in $K$. In any case, we argue that putting the barrier function on the boundary of a reasonably good approximation of the viability kernel provides much higher guarantee than putting this barrier on the boundary of $K$ (as it is done in [7]).

## IV. EXPERIMENTS

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 [12].

### A. Practical Simplification of the Algorithm

In the theorem, we need to determine if the best set of controls yields a point which is at a distance higher than $\mu\beta(h)$ from the set $L(K_h^n)$. This distance is not direct to compute, and, in a first stage, we approximated it with an Euler schema, following the gradient of $f_n$. However, in addition to being time consuming, this operation leads to cautious approximations and tends to increase the diffusive effect (which is only partially tackled by the optimisation on several time steps). We noticed that, for some problems at least, the following simpler rule to define $K_h^{n+1}$[3] leads to much reduced diffusive effects

$$K_h^{n+1} = \{\vec{x}_h \in K_h^n, \text{ such that } f_n(\vec{x}_h + \varphi(\vec{x}_h, \vec{u}^*)dt) \geq -\delta$$
$$\text{and } (\vec{x}_h + \varphi(\vec{x}_h, \vec{u}^*)dt) \in K\}. \quad (23)$$

We chose $\delta = 1$ because it limits the definition of the current kernel approximation to the $-1$ margin of the SVM. This means that the support vectors of label $-1$ are located inside the approximate kernel, and generally close to its boundary. This tends to satisfy a condition which is stricter than (9), but does not guarantee it.

### B. Simple Model 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,

---
[3]We wrote it here in the case of a one time step optimisation for sake of simplicity, but the extension to several time steps is straightforward.
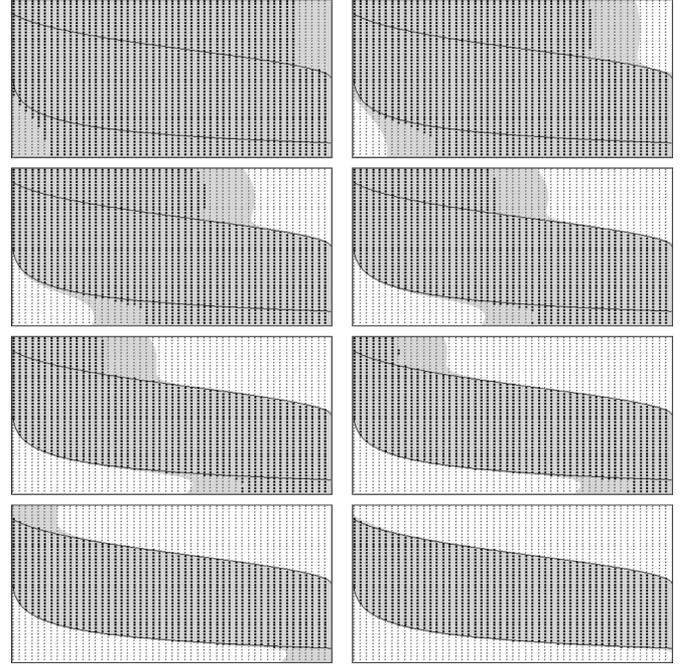


Fig. 1. Example of progressive approximation of the viability kernel in two dimensions for the population problem. The horizontal axis represents the population size $(x)$ and the vertical axis its evolution rate $(y)$. $K$ is the rectangle which limits the points of the grid. The grid includes 2601 points (51 points per dimension). The dark points of the grid are the ones belonging to $K_h^{n+1}$. The approximation of the kernel $L(K_h^n)$ is represented in gray. The parameter $dt$ is computed for defining moves of size $2\beta(h)$ in one time step and the optimization is made on ten time steps.
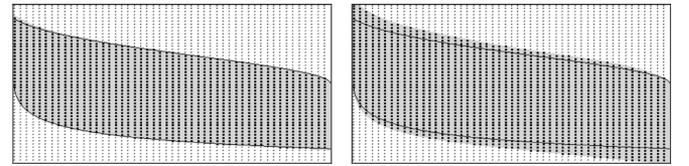


Fig. 2. Comparison of the final approximation of the viability kernel for optimisations with different time steps: On the left, the control optimisation is made on one time step and on the right on ten time steps. $dt$ is computed for defining moves of size $2\beta(h)$ in one time step.

and then redeveloped by Aubin [13] 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:

$$\begin{cases} x(t + dt) = x(t) + x(t)y(t)dt & (24a) \\ y(t + dt) = y(t) + u(t)dt & \text{with } -c \leq u(t) \leq +c. \quad (24b) \end{cases}$$

The advantage of this model is that it is possible to derive analytically the viability kernel [13] and we can compare the approximation given by the algorithm with the theoretical viability kernel.

Fig. 1 shows an example of progressive approximation of the viability kernel. The final approximation of the viability kernel is presented in Fig. 2, which also shows that increasing the number of time steps of control optimization enhances the approximation accuracy.

### C. Example of Heavy Controller Action

The leeway on the determination of the value of $\Delta$ can be made up for by using a more or less cautious controller: By anticipating on several time steps, the initial state tends to go away from the boundary of $A_\Delta$ and to move away from a dangerous area.
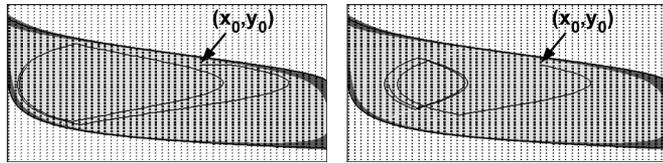
Fig. 3. Example of heavy trajectories from a point $(x_0, y_0)$. The difference between $L(K_h^P)$ and $A_\Delta$ is in dark gray. The trajectories include 200 time steps. On the left: two time steps anticipation. On the right: eight time steps anticipation.

Fig. 3 shows an example of the functioning of the controller for the population problem in two dimensions. We start from the approximation of the viability kernel given by Fig. 2 with 10 time steps. We put $\Delta = 12$ and we compare two types of controller: The first anticipating two time steps ahead, and a more cautious one, with ten time steps anticipation. The computation time required to find a control in both cases is much below the millisecond on a standard PC.

### D. Model of the Southern Benguela Ecosystem

We also tested our approach on a more complex model of ecosystem: The Southern Benguela ecosystem, involving five different groups of species [3] in a dynamical model of biomass evolution. The state space is in six dimensions and the controls in 51 dimensions (including the uncertainty on the coefficients and an evolution rate of the fisheries). This makes Saint-Pierre's algorithm impossible to use, because it would need discretise a 51-dimensional space. We checked that the results obtained with our approach are similar to the ones obtained in [3], with a method specifically developed for the problem. Unfortunately, we cannot present the details of these results within the space of this note (see [14] for details).

## V. Discussion

We demonstrated that it is possible to approximate viability kernels using a classification procedure, if this procedure satisfies some general conditions. We also showed that SVMs are an interesting classification procedure in this context. The main reason is that SVMs provide directly a type of barrier function on the limit of the current viability kernel, which enables to use a gradient method to compute the controls. This opens the possibility to optimise the control in large dimension spaces and over several time steps with a reasonable computing time. We showed on examples that using several time steps significantly improves the final approximation, for a given resolution of the grid, and that the method allowed us to solve a problem with a control space of dimension 51. Moreover, the same approach gives the possibility to define more or less cautious heavy control procedures, with good guarantees to keep the system in the viability kernel approximation. All these possibilities are new compared with Saint-Pierre's method.

A lot of work lies ahead. We left aside several theoretical questions which deserve more rigorous investigations: guarantees for SVMs to fulfill the conditions of the theorem, guarantees on the control scheme, adaptation of the theoretical framework to the practical simplification we made in the examples. Moreover, we believe our work is a first step to exploit SVM good properties for solving control problems in state spaces of high dimensionality. In particular, active learning strategies could yield accurate viability kernel approximations, using very small training sets.

## References

[1] J. Aubin, *Viability Theory*. Boston, MA: Birkhäuser, 1991.

[2] C. Bene, L. Doyen, and D. Gabay, "A viability analysis for a bio-economic model," *Ecol. Econ.*, vol. 36, no. 3, pp. 385–396, 2001.

[3] C. Mullon, P. Curry, and L. Shannon, "Viability model of trophic interactions in marine ecosystems," *Natural Resource Model.*, vol. 17, no. 1, pp. 27–58, 2004.

[4] N. Bonneuil, "Making ecosystem models viable," *Bull. Math. Biol.*, vol. 65, no. 6, pp. 1081–1094, 2003.

[5] M. Kalisiak and M. van de Panne, "Approximate safety enforcement using computed viability envelopes," in *Proc. IEEE Int. Conf. Robot. Automat.*, 2004, vol. 5, pp. 4289–4294.

[6] P. Faloutsos, M. van de Panne, and D. Terzopoulos, "Autonomous reactive control for simulated humaniods," in *Proc. IEEE Int. Conf. Robot. Animat.*, 2003, pp. 917–924.

[7] R. J. Spiteri, D. K. Pai, and U. M. Ascher, "Programming and control of robots by means of differential algebraic inequalities," *IEEE Trans. Robot. Automat.*, vol. 16, no. 2, pp. 135–145, Apr. 2000.

[8] P. Saint-Pierre, "Approximation of viability kernel," *Appl. Math. Optim.*, vol. 29, pp. 187–209, 1994.

[9] V. Vapnik, *Statistical Learning Theory*. New York: Wiley, 1998.

[10] N. Cristianini and J. Shawe-Taylor, *Support Vector Machines and Other Kernel-Based Learning Methods*. Cambridge, U.K.: Cambridge Univ. Press, 2000.

[11] J.-P. Aubin, *Neural Networks and Qualitative Physics: A Viability Approach*. Cambridge, U.K.: Cambridge Univ. Press, 1996.

[12] J. Platt, "Fast training of support sector machines using sequential minimal optimization," in *Advances in Kernel Methods—Support Vector Learning*, C. B. B. Schlkopf and A. Smola, Eds. Cambridge, MA: MIT Press, 1999, ch. 12, pp. 185–208.

[13] J.-P. Aubin and P. Saint-Pierre, "An introduction to viability theory and management of renewable resources," in *Advanced Methods for Decision Making and Risk Management*, J. Kropp, Ed., J. Scheffran, Ed. Hauppauge, NY: Nova, 2006, ch. 2, pp. 52–96.

[14] L. Chapel, G. Deffuant, S. Martin, and C. Mullon, "Defining yield policies in a viability theory approach," in *Proc. 5th Eur. Conf. Ecolog. Model.*, 2005, pp. 35–36.