Proceedings of the 2007 IEEE Symposium on Approximate
Dynamic Programming and Reinforcement Learning (ADPRL 2007)

SVM Viability Controller Active Learning: Application to Bike Control

Laetitia Chapel
Cemagref LISC
Aubière, France
Email: laetitia.chapel@cemagref.fr

Guillaume Deffuant
Cemagref LISC
Aubière, France
Email: guillaume.deffuant@cemagref.fr

Abstract— **It was shown recently that SVMs are particularly adequate to define action policies to keep a dynamical system inside a given constraint set (in the framework of viability theory). However, the training set of the SVMs face the dimensionality curse, because it is based on a regular grid of the state space. In this paper, we propose an active learning approach, aiming at decreasing dramatically the training set size, keeping it as close as possible to the final number of support vectors. We use a virtual multi-resolution grid, and some particularities of the problem, to choose very efficient examples to add to the training set. To illustrate the performances of the algorithm, we solve a six-dimensional problem, controlling a bike on a track, problem usually solved using reinforcement learning techniques.**

I. Introduction

In general, control techniques and procedures try to provide action policies which optimise some criteria along a trajectory. Yet, in some cases, the problem is rather to provide action policies which keep a set of constraints satisfied. For example, in ecology or economics, it is easier to define a set of constraints that the system must satisfy than of an optimum to reach. Viability theory [1] focuses on such problems. In this theory, the states from which there exists an action policy which keeps the constraints satisfied are called viable states. The set of all the viable states, called viability kernel, is particularly important in this theory because it is used to define a variety of action policies: Inside the kernel, there is at least one action that allows the system to keep the set of constraints satisfied, and outside, whatever the action policy applied, the system breaks the constraints after a finite time.

Viability theory is more and more used to solve difficult control problems in robotics [2], [3], economy [4] or ecology [5], [6]. However, there is no explicit formula to define the viability kernel. The algorithm developed by Saint-Pierre [7] and directly inspired by the work of Aubin [1] is based on a discrete grid of the state space, and provides the viability kernel as a subset of points of this grid. This approach faces directly the curse of dimensionality, which limits its use to problems in low dimensional state space.

The method proposed in [8] builds on Saint-Pierre's algorithm [7], but uses a classification function, based on support vector machines (SVMs) to define the viability kernel. Providing a differentiable function of the distance to the boundary of the viability kernel approximation, SVMs allow to use optimization techniques to compute the controls, which enables to tackle problems with continuous control in large dimension spaces. Moreover, SVMs generally provide a very parsimonious controller, limited to the support vectors, compared with value or policy iteration techniques which require to keep the whole set of states. Lagoudakis and Parr [9] also use SVMs to solve optimal control problems, with a different approach. They train SVMs iteratively on sets the states associated with the corresponding best action. Their technique is therefore limited to problems with a small set of actions. Another difference with our approach is that they use random distributions of states to train the SVMs, whereas we use grids to control the precision of our approximation. In both cases, the dimensionality curse problem remains, because the size of the SVM training set to get a given accuracy of approximation increases exponentially with the dimension of the state space, and with the accuracy of the approximation.

In this paper, we propose some improvements of the active learning approach described in [10]. Active learning algorithms try to minimize the number of instances of the training set, in order to save computation and time, but also the cost of labeling unneeded examples. Indeed, in some applications like text classification, the cost of labeling data can be very important. The aim of active learning is to select the more informative instances to label and to include into the training set. An active learner begins with a small training set and iteratively increases its size. SVMs have drawn much attention in this context, because they provide a subset of the most informative instances, the support vectors, to define the separating surface. For instance, [11] proposed a heuristic to select the instances, based on their proximity to the SVM separating surface, because it maximally reduces the version space. The same procedure is developed in [12], with a more geometrical justification, and shown to frequently decrease the computing time.

We combine an active learning procedure with a multi-resolution grid, in order to dramatically decrease the size of the training set, while keeping an accurate approximation of the kernel. The aim is to use a training set which size is close to the number of support vectors of the SVM. Munos and Moore [13] similarly use variable resolution discretization and compare different splitting methods in reinforcement learning algorithms. The main difference with our method is that our multi-resolution grid is "virtual", it is never instantiated explicitly. We only use it as a mean to get relevant examples to add to the SVM training set. We add only pairs of examples of opposite labels, which are neighbours on the finest resolution grid. The added pairs of examples are distributed sparsely, using the coarse grid as a reference. They constrain very rapidly the SVM, which makes no error at the finest resolution after few iterations.

We illustrate our approach on problems in 2, 3 and 6 dimensions. In the last problem, the aim is to balance a bicycle: Several authors e.g. [14], [9] already treated this problem using reinforcement learning and shaping. We show that viability theory can be an alternative to solve it. We present first the main concepts of viability theory and rapidly recall the SVM viability controller algorithm. In section 3, we describe the active learning procedure using a multi-resolution grid. Section 4 reports the results of experiments on examples of different dimensions. Finally, we discuss the results and draw some perspectives.

In this paper, we assume that we have the dynamics of the model, the set of constraints to be satisfied and that the dynamics are deterministic.

II. SVM VIABILITY CONTROLLER TRAINED ON A GRID

A. *Viability kernels to build viability controllers*

Viability theory [1] provides tools and methods in order to control a dynamical system such that it remains inside a set of admissible states K, called the viability constraints set.

Consider a dynamical system defined by its state $\vec{x}(t) \in \mathbb{R}^d$ and suppose that its evolution can be modified by controls $\vec{u}$, in discrete time:

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

Aubin [1] defines the concept of viable state, as a state $\vec{x}_0$ for which there exists at least one control function that allows the trajectory $\vec{x}(t)$, satisfying (1), to remain in K indefinitely:

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

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

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

The issue is then to find the control function $t \to \vec{u}(t)$ that allows to keep the system in the viability constraint set indefinitely. The viability kernel can be used to define control policies: It is sufficient to choose any control that allows a state $\vec{x}(t) \in Viab(K)$ to stay inside the kernel at the next iteration (by definition, we know that such a control exists). The simplest rule (called heavy control) is to keep the control constant as long as the state remains inside the kernel and to apply the first control which keeps the system in $Viab(K)$ otherwise.

The main task to solve a viability problem is therefore to compute the viability kernel. Aubin [1] proves the viability theorems that enable to determine the viability kernel, under some general conditions (Marchaud systems[1] for example). These theorems enable to determine a viable state, without exploring the combination of all the control actions in time.

In general, there is no explicit analytical definition of the viability kernel, and it is thus necessary to approach it numerically. Saint-Pierre [7] developed a method which computes a discrete approximation of the viability kernel. It uses the definition of $Viab(K)$ as the largest subset E of K such that there exist at least one control action that enables $\vec{x}(t)$ to stay in E at the next time step:

$$\forall \vec{x}(t) \in E, \exists \vec{u}(t) \in U(\vec{x}), \vec{x}(t+dt) \in E. \tag{4}$$

Moreover, Saint-Pierre's method uses a discrete approximation of the dynamical system (1), on a grid G covering K. The procedure gradually removes the points for which there exists no control allowing the system to stay inside the current discrete approximation of the kernel. Saint-Pierre [7] proves that the discrete approximation of the viability kernel tends to $Viab(K)$ when the resolution of the grid tends to 0. The algorithm is fast but suffers several limitations:

- The definition of the problem on a grid limits the application to problems of low dimensional state because it is subject to the curse of dimensionality;
- It also uses a discrete set of controls and is thus limited to systems with low dimensional controls;
- The final result as a set of points (which can be huge) is not very convenient.

More recently, Ultra-Bee schemes were tested to solve viability problems [15]. The viability problem is considered as an optimal control problem, solved with a value function approach. The method shows good results, particularly because of its anti-diffusive properties. However, like for the Saint-Pierre's algorithm [7], it is based on a grid covering the state space. In the same direction, [16] proposed an alternative approach to approximate viability kernel by using the value-function of a dynamic programming problem. They defined the kernel as the set of states for which the value function is below a given threshold. But the determination of this threshold is not easy.

B. *SVM viability kernel approximation*

Saint-Pierre's work is the starting point of a recent algorithm of viability kernel approximation [8] using support vector machines (SVMs [17], [18]). SVM classification functions provide several advantages: They are very concise continuous approximations of a viability kernel which enable to use optimization techniques to compute the controls. They also enable to derive more sophisticated control policies, using the distance between a point and the boundary of the kernel. We now recall the main step of this algorithm.

SVMs are particular kernel methods, based on a kernel function $k(\vec{x}, \vec{y})$ which defines a scalar product in a feature space where the data are (implicitly) projected. Considering a

[1]The most severe condition is that for each state $\vec{x}$, the set of velocities $\varphi(\vec{x}, \vec{u})$, $\vec{u} \in U(\vec{x})$, is convex

training set $S = \{(\vec{x}_i, y_i), i = 1 \text{ to } l\}$, where $\vec{x}_i$ is a vector in a space $\mathscr{X} \in \mathbb{R}^N$ and $y_i \in \{-1, +1\}$, SVM training on a set S is solving a quadratic problem on a set of variables $\alpha_i \geq 0$, which provides function f:

$$f(\vec{x}) = \sum_{i=1}^{n} \alpha_i y_i k(\vec{x}_i, \vec{x}) + b. \tag{5}$$

The support vectors are vectors $\vec{x}_i$ such that $\alpha_i > 0$. When $f(\vec{x}) \geq 0$, $\vec{x}$ is classified $+1$, otherwise -1.

We use Platt's SMO method [19] to train the SVMs.

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

$$\forall \vec{x} \in K, \exists \vec{x}_h \in G \text{ such that } d(\vec{x}, \vec{x}_h) \leq h, \tag{6}$$

where d denotes the distance based on the norm "max":

$$d(\vec{x}, \vec{y}) = max_i \, |x_i - y_i| \, . \tag{7}$$

The algorithm of approximation of the viability kernel is iterative. At each step n, we build a training set S_n, by associating label $+1$ to points $\vec{x}_h$ of G that are viable relatively to the $(n-1)^{th}$ approximation of the viability kernel, and label -1 otherwise. Training a SVM on this set provides a SVM function f_n which defines V_n, the n^{th} approximation of the viability kernel, as follows:

$$V_n = \{\vec{x} \in K \text{ such that } f_n(\vec{x}) \geq -1\}. \tag{8}$$

We define function V_n with the -1 margin of the SVM because it enables to dilate the set of viable states and avoid rejecting points too easily, in order to fulfill the conditions of the algorithm's convergence. In addition, the SVM function must be constructed such that it makes no errors on the grid. The conditions on the regularity and dilatation of V_n are discussed in [8].

To compute the states which are viable relatively to V_n, we look for the control action which tends to bring the system back to V_n as much as possible. We use the fact that function f_n approximates an algebraic distance to the boundary of V_n to compute this best control $\vec{u}^*$. We define $\vec{x}^*$, as the point obtained from $\vec{x}$, by applying the best control $\vec{u}^*$ (with respect to SVM function f_{n-1}) during the time interval dt:

$$\vec{x}^* = \vec{x} + \varphi(\vec{x}, \vec{u}^*) dt. \tag{9}$$

This definition can easily be enlarged to several time steps.

Then, if $\vec{x}_h^* \in V_n$, we associate x_h with label $+1$, -1 otherwise. The procedure stops when training sets S_{n+1} and S_n are equal. Algorithm 1 summarizes the approximation procedure. At the beginning of the algorithm, when no SVM is available, we use the border of K to compute the labels. The convergence of the algorithm to the actual viability kernel is discussed in [8].

Algorithm 1 SVM viability kernel approximation algorithm

```
Define $S_0$ from $K$
Compute $f_0$ from $S_0$
$n = 0$
**repeat**
  $S_{n+1} \leftarrow \emptyset$
  **for all** $\vec{x}_h \in G$ **do**
    **if** $f_n(\vec{x}_h^*) \geq -1$ **then**
      $S_{n+1} \leftarrow S_{n+1} \bigcup (\vec{x}_h, +1)$
    **else**
      $S_{n+1} \leftarrow S_{n+1} \bigcup (\vec{x}_h, -1)$
    **end if**
  **end for**
  Compute $f_{n+1}(\vec{x})$ from $S_{n+1}$
  $n = n + 1$
**until** $S_n = S_{n+1}$
**return** $f_{n+1}$
```

C. SVM viability controller

Viability kernels enable to define control policies. The aim is not to find an optimal controller like in reinforcement learning, but to keep the constraints always satisfied. In this section, we introduce the heavy controller, but other procedures can be derived from the viability kernel.

The idea of heavy control procedure comes from Aubin [1]. The principle is to apply a constant control until the system would come outside the viability kernel at the next time step, and then to choose any control which keeps the system inside the viability kernel (by definition, we know that there exists at least one control that allows the trajectory to stay in the kernel). The SVM viability controller adapts the heavy control procedure, including a security distance to the boundary of the approximation.

Let V be the final approximation of the viability kernel, obtained with the final SVM function f. We denote ∂V the boundary of V. To introduce a security distance, we define Δ a given positive real number. Given an initial state $\vec{x}_0$ and an initial control $\vec{u}_0 \in U(x)$, the procedure associates a control $\vec{u}_{k+1}$ at the $(k+1)^{th}$ control iteration as follows:

- If $f(\vec{x}_k + \varphi(\vec{x}_k, \vec{u}_k) dt) \geq \Delta$, we keep the same control ($\vec{u}_{k+1} = \vec{u}_k$),
- Else, $\vec{u}_{k+1} = \vec{u}^*$.

In addition to the security distance Δ, we can define more or less cautious controller, by anticipating on several time steps. The conditions in which this procedure guarantees to keep the system inside $Viab(K)$ are discussed in [8].

III. Active learning procedure

A. Principles

The previous algorithm provides a result that converges to the actual viability kernel when the resolution h of the grid tends to 0. But we face the curse of dimensionality: The size of the grid (and thus the SVM training set size) grows exponentially with the state space dimension. However, the number of support vectors which are finally retained in the SVM is in general very small compared to the size of the grid. Typically, we get a few thousand support vectors for a grid of a few million points. In this paper, we aim at minimizing the SVM training set size, to keep it as close as possible to the number of support vectors, in order to dramatically decrease the size of the memory necessary to deal with large grids. To achieve this aim, we use the concept of active learning. An active learner attempts to minimize the number of instances to label and to include in the training set. It progressively adds well-chosen instances to an initially small training set.

As a consequence, we must train the SVM with partial training sets, in which examples are progressively added, until we are sure that the SVM makes no mistake on the whole grid. Therefore, there is an additional loop in the global algorithm 1, devoted to the iterative active learning of the SVM. The choice of examples to add is the typical problem of active learning. A compromise between the number of SVM training iterations and the size of the training set must be found. In our particular problem, we look for pairs of close points with opposite labels, because such points constrain very sharply a classification function. Compared with a strategy adding single examples, we get a slightly bigger learning set, but much less iterations.

Let f_{n-1} be the SVM function obtained at the previous global iteration. The active learning procedure involves two steps:

- Compute the initial training set S_n^0 and SVM function f_n^0 from f_{n-1};
- Repeat: Define S_n^{k+1} from S_n^k and f_n^k, then train f_n^{k+1} on it, until f_n^{k+1} makes no mistake on the whole grid.

Algorithm 2 sums up the main steps of the active learning viability kernel approximation procedure.

Algorithm 2 Active learning global procedure

```
$S_0 \leftarrow$ initialiseFirstTrainingSet()
$f_0 \leftarrow$ SVMTrain($S_0$)
n = 0
loop
    $S_{n+1}^0 \leftarrow$ initialiseTrainingSet($f_n$)
    if globalStoppingCriterion($f_n$) then
        return $f_n$
    end if
    $f_{n+1}^0 \leftarrow$ SVMTrain($S_{n+1}^0$)
    k = 0
    repeat
        $S_{n+1}^{k+1} \leftarrow$ updateTrainingSet($S_{n+1}^k$)
        $f_{n+1}^{k+1} \leftarrow$ SVMTrain($S_{n+1}^{k+1}$)
        k++
    until $S_{n+1}^{k-1} = S_{n+1}^k$
    $f_{n+1} \leftarrow f_{n+1}^k$
    n++
end loop
```

The main contribution of this paper is the active learning procedure for updating the training set. The main idea is to add a limited number of pairs of instances which constrain the SVM f_n^k as much as possible in each update of the training set in the internal loop (on k).

To describe the procedure in more detail, we need additional notations.

B. Multi-resolution grid

We use a virtual grid of the state space, through which we can go at different resolutions: Each point of the coarsest grid represents the root of a tree, and the children are points along all d dimensions, leading to 2^d children at each level.

We suppose that the state space is the set $[0,1]^d$, and we set $h = 1/(m-1)$, where m is an integer. The coordinates x_i of a point $\vec{x}$ of the grid of resolution $h/2^j$, where j is an integer, are given by integers q_i with $0 \leq q_i \leq m \times 2^j$, with:

$$x_i = \frac{q_i h}{2^j} \text{ for } i = 1,...,d. \tag{10}$$

Therefore, the grid of resolution $h/2^j$ includes $((m \times 2^j) + 1)^d$ points. To simplify, in the following, we shall refer to resolution $h/2^j$ as resolution j, and we call G_j the corresponding grid. We call j_M the finest resolution. For all resolutions j, we have:

$$\forall \vec{x} \in K, \exists \vec{y} \in G_j \text{ such that } d(\vec{x},\vec{y}) \leq \frac{h}{2^{j+1}}. \tag{11}$$

Considering $\vec{x} \in K$, we call $G_j(\vec{x})$ the point of G_j which is the closest to $\vec{x}$.

The multi-resolution grid is organized as a tree. For $j < j_M$, we define the children of point $\vec{x} \in G_j$ as the set $C(\vec{x})$ of points belonging to G_{j+1}:

$$C(\vec{x}) = \Big\{ \vec{y} = \vec{x} + \varepsilon_i \frac{h}{2^{j+1}}, \text{ such that } \vec{y} \in K \tag{12}$$

$$\text{with } \varepsilon_i \in \{0,1\} \text{ and } i = 1,...,d \Big\}. \tag{13}$$

The children of a point is therefore a set of size 2^d, except for a subset on some borders of K (in which the number of children is lower, because some of the “natural” children do not belong to K). Note that $\vec{x} \in C(\vec{x})$. The set of children allows us to define recursively the tree of level k at point $\vec{x}$:

$$T_0(\vec{x}) = \{\vec{x}\}\,; T_{k+1}(\vec{x}) = \bigcup_{\vec{y} \in T_k(\vec{x})} C(\vec{y}). \tag{14}$$

C. Updating the training set

To update the training set S_n^{k+1}, we first consider each point $\vec{x}$ of the coarse grid G_0 and test if it is closer than h to the border ∂V_n^k of the current viability kernel approximation V_n^k. If it is, and if it is well-classified, then we go through each tree level: $T_0(\vec{x})$, $T_1(\vec{x})$,..., $T_{j_M}(\vec{x})$, considering only the points which are closer than the considered resolution to ∂V_n^k, until we find a point which is misclassified by f_n^k. Let us define function $l(\vec{x}, f)$:

$$l(\vec{x}, f) = +1 \text{ if } f(\vec{x}) \geq -1, \ -1 \text{ otherwise.} \tag{15}$$

Following this procedure (detailed in Algorithm 3), if we find one misclassified point, we add to S_n^{k+1} a pair of points from the finest grid G_{j_M}, with opposite labels, and chosen in order to maximize the learning efficiency (called getClosestPair (see next section) in algorithm 4, which sums up the procedure). In this procedure, we add at most one pair of training examples for each point of the coarse grid. The advantage of using a coarse grid is that the pairs of points we add are well-spread on the whole grid.

If we don't keep in memory all the tested points associated with their label, the main drawback of this procedure is to compute at each iteration the labels of all the examples which are close to ∂V_n^k, if this surface did not change much, we make the same computations several times. This can be expensive in computational time, especially when the dynamics of the model is complex, and when the optimization is made over several time steps. An interesting compromise is to keep in memory only the coarse grid points for which there was no misclassification, and the first misclassified example otherwise. Generally, this list is much smaller than the total set of tested examples. With this information, we can use the previous SVM to label directly the examples. If a coarse grid point $\vec{x}$, is in the list of well-classified points, then the previous SVM provides the good labels to all its children. If one child of $\vec{x}$ is in the list of misclassified points, then the previous SVM provides the good labels to all children up to this one. If $\vec{x}$ is in none of the lists, then the tests must be done completely. With this approach, the complete computation of a label is made only once for each point.

At the first loop of the procedure, we define f_n^0 using the support vectors of f_{n-1}, which gives it good chances to be good on the coarse grid. This initialisation also calls the procedure providing pairs of points on the finest grid, chosen to maximise the learning efficiency. We now describe this procedure, which is essential in our approach, in the next paragraph.

To sum up,

- The tested points are those located near the boundary, at the resolution of the coarse grid;
- The trained points are pairs of points of opposite labels, located on the coarse grid.

Algorithm 3 firstMisclassifiedInAllChildren(point $\vec{x}$)

```
for $j = 0$ to $j_M$ do
    for all $\vec{y} \in T_j(\vec{x})$ do
        if $d(\vec{y}, \partial V_n^k) < \frac{h}{2^j}$ then
            if $l(\vec{y}, f_n^k) \neq l(\vec{y}^*, f_{n-1})$ then
                return $\vec{y}$
            end if
        end if
    end for
end for
return nil
```

Algorithm 4 updateTrainingSet(set S)

```
$S_n^{k+1} \leftarrow S_n^k$
for all $\vec{x} \in G_0$ do
    $\vec{y} \leftarrow$ firstMisclassifiedInAllChildren($\vec{x}$)
    if $\vec{y} \neq$ nil then
        $S_n^{k+1} \leftarrow S_n^{k+1} \bigcup$ getClosestPair($\vec{y}$)
    end if
end for
```

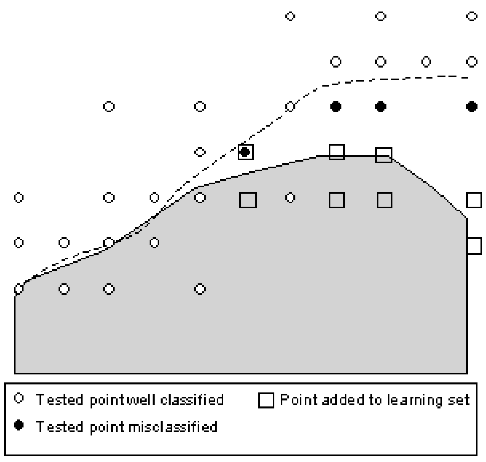
○ Tested point well classified
● Tested point misclassified
□ Point added to learning set

Fig. 1. Training set update. The gray area is the area to approximate. The dotted line is the separating surface of the current SVM in the training loop. We note that in the part of the surface where there is no mistake, the tests up to the finest grid are made around the separating surface. On the contrary, in the zones where there are mistakes, the number of tests is lower. The pairs of added points to update the training set are represented by squares. In this case, 4 pairs are added.

D. Providing a pair of points of opposite labels on the finest grid

We consider point $\vec{x} \in G_j$, which is misclassified. We look for a pair of points $\vec{x}_1$ and $\vec{x}_2$, neighbours on the finest grid, such that $l(\vec{x}_1^*, f_{n-1}) \neq l(\vec{x}_2^*, f_{n-1})$, and which are as close as possible to $\vec{x}$. We navigate on grid G_{j_M}, following the direction of the gradient $\nabla f_n^k(\vec{x})$. The procedure involves two steps:

- We look for the point x_2 such that $d(\vec{x}, \vec{x}_2) = h$ and $l(\vec{x}^*, f_{n-1}) \neq l(\vec{x}_2{}^*, f_{n-1})$. This step allows to navigate more quickly on the grid.
- We then divide recursively the segment into two parts, keeping the part cutting the border, until the size of the segment is the finest resolution.

The algorithm 5 sums up this procedure and Fig. 1 represents the points requested and added to the training set.

E. Initial training set S_n^0 and global stopping criterion

In order to limit the size of S_n^0, we start from the support vectors (SVs) of function f_{n-1}. S_n^0 is made of the set of the pairs of neighbours of opposite labels in the most refined grid provided by algorithm 5, applied to the SVs of f_{n-1}. The size of S_n^0 is thus limited to ($2\times$ Number of SV of f_{n-1}). The global stopping criterion is met if all the SVs of f_{n-1} remain

Algorithm 5 getClosestPair(point $\vec{x}_1$)

```
$x_2 \leftarrow x_1$
**repeat**
  $\vec{x}_2 = G_{j_M}\left(\vec{x}_2 + l(\vec{x}_1^*, f_{n-1})\frac{\nabla f_n^k(\vec{x}_1))}{\|\nabla f_n^k(\vec{x}_1))\|}h\right)$
**until** $l(\vec{x}_1^*, f_{n-1}) \neq l(\vec{x}_2^*, f_{n-1})$
**while** $d(\vec{x}_1, \vec{x}_2) > h/2^{j_M}$ **do**
  $\vec{y} = G_{j_M}((\vec{x}_1 + \vec{x}_2)/2)$
  **if** $l(\vec{y}^*, f_{n-1}) = l(\vec{x}_2^*, f_{n-1})$ **then**
    $\vec{x}_2 \leftarrow \vec{y}$
  **else**
    $\vec{x}_1 \leftarrow \vec{y}$
  **end if**
**end while**
**return** $\{(\vec{x}_1, l(\vec{x}_1^*, f_{n-1})), (\vec{x}_2, l(\vec{x}_2^*, f_{n-1}))\}$
```

at the border (each SV is the first member of the pair returned by algorithm 5), and if there is no error around f_{n-1} at the most refined resolution.

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 [19]. We consider a gaussian kernel:

$$k(\vec{x}_1, \vec{x}_2) = exp\left(-\frac{\|\vec{x}_1 - \vec{x}_2\|^2}{2\sigma^2}\right) \quad (16)$$

and we use the library LIBSVM [20], which implements a SMO-type algorithm written in Java, to compute the SVM. We made the experiments on a 2.8 Ghz computer and 512 Megabytes of RAM.

A. A two-dimensional problem :Population problem

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 [21] 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:

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

It is possible to derive analytically the viability kernel of this problem [21], in order to compare the approximation to the actual viability kernel.

Figure 2 shows the result of the viability kernel approximation and an example of heavy trajectory. The points on the graph are those used to compute the last SVM. Table II

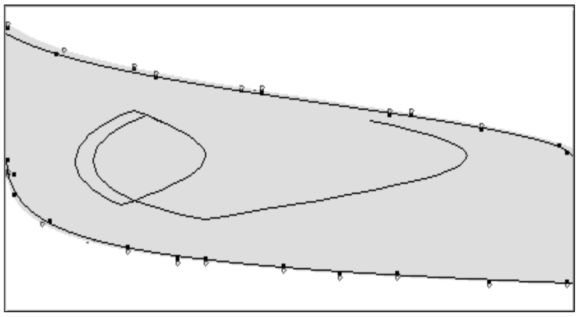

Fig. 2. Final approximation (in gray) using a whole grid of 6561 points on the finest grid (81 points by dimension) and example of heavy trajectory. The optimisation is made on 3 time steps. *dt* is computed for defining moves of size *h* in one time step. The continuous lines delimit the actual viability kernel.

presents the main characteristics of the approximation. The trajectory includes 80 time steps, anticipating on 5 time steps and with a security distance of $\Delta = 8$.

This example shows some advantages of the SVM viability controllers using active learning and adaptive grid: Good quality of approximation, facility to compute the control action, and limited number of points to train the SVM. At the maximum, the training set size is less than 2% of the size of the finest grid and less than 12% of the points of the finest grid have been requested for one iteration. The full computation of the approximation takes 98 sec. In this example, the dynamics are simple and the number of support vectors of the final approximation is very small, and so the time to evaluate one action.

B. A three-dimensional problem :Car on the Hill

We consider the well-known car on the hill problem. The state is two-dimensional (position and velocity) and the system can be controlled with an continuous one-dimensional action (the thrust). For a description of the dynamics and the state space constraints, one can refers to [22]. The aim of the car on the hill system is not only to keep the car inside a given set of constraints, but also to reach a target (the top of the hill).

In viability theory, this type of problem is called capture basin problems, and they can be seen as an extension of the standard viability problems. Let target C be a closed subset of K. The set of initial set $\vec{x}_0 \in K$ such that C is reached in finite time before leaving K is called the capture basin of C in K and is noted $Capt(K, C)$. The capture basin of a system is the viability kernel of an extended system, where a state describing the evolution of time is added:

$$\begin{cases} \vec{x}'(t) = \varphi(\vec{x}(t), \vec{u}(t)) \\ \vec{\tau}'(t) = -1 \text{ if } \vec{x} \notin C, \ 0 \text{ if } \vec{x} \in C. \end{cases} \quad (18)$$

and $(\vec{x}(t), \vec{\tau}(t)) \in K' \times \mathbb{R}^+$ is the viability constraint set of the extended system.

The car on the hill problem can thus be solved with a viability approach in 3 dimensions (adding one dimension for time). In this particular case, the 3 dimensions problem can be transformed into several problems in 2 dimensions. Each

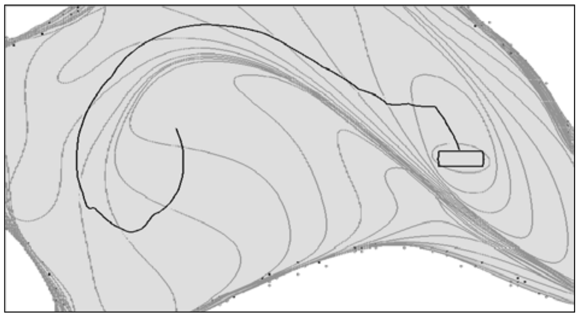

Fig. 3. Final approximation (in gray) using a whole grid of 6561 points on the finest grid (81 points by dimension) and example of heavy trajectory. The optimisation is made on 8 time steps, $dt = 0.015$.

problem in 2 dimensions is in a plane defined at a given value of time left before reaching the target (we do not have the space here to describe the method in full details). Figure 3 represents the projection of the viability kernel limits in such hyperplanes and an example of a heavy control trajectory for the car on a hill problem. Table II presents the main characteristics of the approximation. We notice that, like in the population problem, the maximal number of points used to learn the boundary is very small (about 4.5% of the total size of the grid). The computation of the approximation of the viability kernel takes 30 minutes.

The trajectory includes 96 time steps, anticipating of 8 time steps and with a security distance of $\Delta = 3$. For states which are close to the viability kernel boundary, the viability controller provides the same actions as the ones provided by optimal control approaches. In average, the controller takes about 0.4 sec to choose one action.

C. A six-dimensional problem :Driving a bike on a track

Several researchers of [14], [9] test their algorithms on the problem to balance a bicycle and to drive it to a goal. They solve this problem by using reinforcement learning and shaping. The problem is a six-dimensional state space problem: Angle of the handlebars are displaced from normal and velocity of the angle, angle from the bicycle to vertical and its velocity, position of the front wheel and angle of tilt of the center of the mass. The bike can be controlled by modifying the torque applied to the handlebars and by displacing the center of its mass. We consider the same dynamical system (for a description of the dynamics, see [14]), but instead of learning to reach a target, we learn the bike to drive on a track, without falling and going outside the road.

1) Viability kernel approximation: First of all, to approximate the viability kernel of the problem, constraints must be put on all the dimensions of the problem state space. Tab. I details the different boundaries of the state space, which define the viability constraint set K. Tab. II gives some details about the SVM active learning algorithm.

We notice that the training set size is relatively small compared to the size of the finest grid (only 6.5%). This ratio is not as good as the one for the population problem: The

State	Intervals
Angle the handlebars are displaced from normal (θ)	$\pm\frac{\pi}{2}$ radians
Velocity of the angle ($\dot{\theta}$)	± 6 radians/second
Angle from vertical to bicycle (ω)	$\pm\frac{\pi}{15}$ radians
Velocity of the angle ($\dot{\omega}$)	± 0.75 radians
Position of the front wheel (x)	± 2 meters
Angle of tilt of the center of the mass (ϕ)	$\pm\frac{\pi}{2}$ radians

TABLE I
DIMENSIONS AND CONSTRAINTS

Problem	Pop	Car	Bike
Number of points of the coarse grid	121	121	15625
Number of points of the finest grid	6561	6561	531441
Maximal depth of tree	4	4	2
Average training number by iteration	6	8	21
Number of SV of the last approximation	28	32	3914
Max. size of the training set	124	294	34028
Max. number of points requested	761	1024	258900

TABLE II
DETAILS OF THE ALGORITHM OF THE APPROXIMATION OF VIABILITY KERNEL FOR THE POPULATION, CAR ON THE HILL AND BIKE PROBLEMS.

main reason is that the adaptive grid is defined only with a tree of depth 2. The algorithm presented here allows to dramatically decrease the number of the training set size, but the time to compute the solution is still exponential with the dimensionality, because we check the SVM on all the finest grid points around the border of the separating surface, and the dynamics are complex. The time used to train the SVM becomes negligible compared to the time to check it. We need several days of computation to find the approximation of the kernel. Lagoudakis and Parr [9] use a number of rollouts which is much smaller than the size of our finest grid (about 100 times).

2) Driving the bike on the track: The bike is then controlled by the SVM on the track. The track is in dimension 2. It includes 2 parts: Straight lines and curves. The controller views the track in the bike's relative coordinates, considering the tangent direction of the track at the position of the bike. To control the bike on the bends, we have only to update the angle of tilt of the center of the mass with the slope of the curve. By this means, the controller is operational on any track of any shape, provided that the curves are not too sharp.

Fig 4 represents an example of trajectory of the bike, starting from the initial state in dimension 6 $\vec{x}_0 = (0,0,0,0,0,0)$ and an initial control $\vec{u}_0 = (0,0)$. We represent a bike trajectory, starting from a point of departure $\vec{x}'_0$ located on a straight line of the track, and during 800 time steps. When the bike approaches a dangerous area (the bounds of the tracks or the boundary of the viability kernel approximation), the controls are modified, in order to keep the system viable. We noticed that, in the straight line of the track, the bike go straight on: The control allows it to be safe and does not need to be updated. But, when it approaches the bend, it must change its trajectory by modifying the controls, and the bike begins to turn.

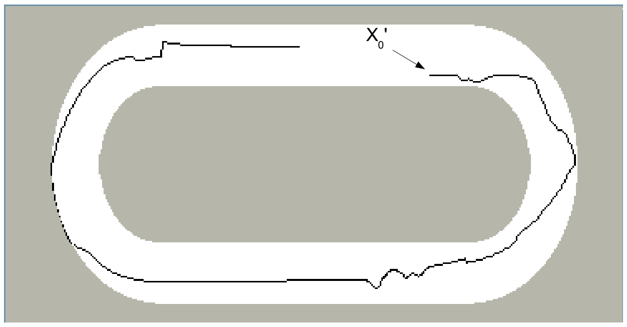

Fig. 4. Trajectory of the bike on a track, starting from x_0' and during 800 time steps. The track is in white. The controller is defined with a security distance $\Delta = 2$ and anticipating on 4 time steps.

Even if it is very expansive to compute the viability kernel of the system, SVMs provide a parsimonious controller. It takes 2 seconds to compute one action on average. The time consuming part is the optimisation to find the control, when the controller anticipates an exit from the viability kernel. It is probably possible to decrease this time significantly with more appropriate choices of the optimisation parameters.

V. Conclusion and perspectives

We proposed a new procedure of active learning with SVMs, which is adapted to the problem of viability kernel approximation. It uses a virtual multi-resolution grid, and the distance to the separating surface defined at the previous iteration. The gradient of the SVM function is essential in the whole process, to compute if the distance is larger than the considered resolution, or to determine the pairs of examples, with opposite labels, which are added to the training set. The whole procedure allows us to decrease dramatically the number of examples to keep in memory for such problems. This opens the possibility to tackle larger dimension problems. However, the procedure is still time-consuming due to the number of points to test at each iteration of the learning procedure.

We illustrate the performance of the algorithm on three examples of different dimensions. One is a six-dimensional problem, usually solved by reinforcement learning techniques: The aim is to drive balance a bike and to drive it on a track, using SVM viability controller active learning algorithm.

In the future, we intend to test more extensively the method on other problems of different dimensions. Moreover, we plan to launch some theoretical investigations about the conditions of convergence of the algorithm, and possibly on more appropriate kernel methods.

References

[1] J.-P. Aubin, *Viability Theory*. Birkhäuser, 1991.

[2] M. Kalisiak and M. van de Panne, “Approximate safety enforcement using computed viability envelopes,” in *IEEE International Conference on Robotics and Automation*, vol. 5, 2004, pp. 4289–4294.

[3] P. Faloutsos, M. van de Panne, and D. Terzopoulos, “Autonomous reactive control for simulated humaniods,” in *IEEE International Conference on Robotics and Animation*, 2003.

[4] C. Bene, L. Doyen, and D. Gabay, “A viability analysis for a bio-economic model,” *Ecological Economics*, vol. 36, no. 3, pp. 385–396, 2001.

[5] N. Bonneuil, “Making ecosystem models viable,” *Bulletin of Mathematical Biology*, vol. 65, no. 6, pp. 1081–1094, 2003.

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

[7] P. Saint-Pierre, “Approximation of the viability kernel,” *Applied Mathematics & Optimization*, vol. 29, no. 2, pp. 187–209, 1994.

[8] G. Deffuant, S. Martin, and L. Chapel, “Approximating viability kernel with support vector machines,” submitted on IEEE Transactions on Automatic Control.

[9] M. G. Lagoudakis and R. Parr, “Reinforcement learning as classification: Leveraging modern classifiers,” in *ICML ’03: Proceedings of the 20th International Conference on Machine Learning*, 2003, pp. 424–431.

[10] L. Chapel and G. Deffuant, “Svm viability controller active learning,” in *Kernel machines and Reinforcement Learning Workshop - ICML 2006*, Pittsburgh, USA, 2006.

[11] S. Tong and E. Chang, “Support vector machine active learning for image retrieval,” in *MULTIMEDIA ’01: Proceedings of the ninth ACM international conference on Multimedia*, 2001, pp. 107–118.

[12] G. Schohn and D. Cohn, “Less is more: Active learning with support vector machines,” in *ICML ’00: Proceedings of the Seventeenth International Conference on Machine Learning*, 2000, pp. 839–846.

[13] R. Munos and A. Moore, “Variable resolution discretization in optimal control,” *Machine Learning Journal*, vol. 49, pp. 291–323, 2002.

[14] J. Randlov and P. Alstrom, “Learning to drive a bicycle using reinforcement learning and shaping,” in *ICML ’98: Proceedings of the Fifteenth International Conference on Machine Learning*, 1998, pp. 463–471. [Online]. Available: citeseer.ist.psu.edu/randlv98learning.html

[15] O. Bokanowski, S. Martin, R. Munos, and H. Zidani, “An anti-diffusive scheme for viability problems,” *Appl. Numer. Math.*, vol. 56, no. 9, pp. 1147–1162, 2006.

[16] P. A. Coquelin, S. Martin, and R. Munos, “A dynamic programming approach to viability problems,” in *Proceedings of the 2007 IEEE International Symposium on Approximate Dynamic Programming and Reinforcement Learning*, 2007.

[17] V. Vapnik, *The Nature of Statistical Learning Theory*. Springer Verlag, 1995.

[18] B. Scholkopf and A. J. Smola, *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond*. Cambridge, MA, USA: MIT Press, 2002.

[19] J.-C. Platt, “Fast training of support vector machines using sequential minimal optimization,” 98-14, Microsoft Research, Redmond, Washington, Tech. Rep., 1998.

[20] C.-C. Chang and C.-J. Lin, *LIBSVM: a Library for Support Vector Machines*, 2001, software available at http://www.csie.ntu.edu.tw/ cjlin/libsvm.

[21] J.-P. Aubin, “Elements of viability theory for the analysis of dynamic economics,” *Ecole thématique du CNRS ‘Economie Cognitive’*, 2002.

[22] A. Moore and C. Atkeson, “The parti-game algorithm for variable resolution reinforcement learning in multidimensional state-spaces,” *Machine Learning*, vol. 21, pp. 199–233, 1995.