

Approximate Safety Enforcement Using Computed Viability Envelopes

Maciej Kalisiak
Dept. of Computer Science
University of Toronto
mac@dgp.toronto.edu

Michiel van de Panne
Dept. of Computer Science
University of British Columbia
van@cs.ubc.ca

Abstract—A numerical method is proposed for the constraint of the state of a dynamical system such that it cannot enter a predefined failure region. The proposed approach to this viability problem involves an explicit numerical approximation of a viability envelope, coupled with a practical strategy for enforcing containment that is based upon a predictive look-ahead strategy. The approach can be applied to achieve automated “intervention when necessary” to enforce system safety at interactive rates. Applications are shown to several low-dimensional systems, including steering control of a vehicle constrained to a given environment geometry.

I. INTRODUCTION

Controlling dynamical systems near the limits of their performance can often be a challenging task, as is evidenced by the skill required of a pilot flying near stall-speed during a landing flare, or the finesse of a race-car driver in safely executing a high-speed turn. Much of the requisite skill involves an accurate perception of where the limits of the system are at any point in time, and how these limits project themselves onto the control inputs.

The above observations naturally motivate the need for a mechanism for “enforced safety,” namely ensuring that the applied controls for a system always result in the avoidance of predefined failure regions. This entails over-riding a user’s desired control input in situations that are deemed sufficiently critical. For situations where the system is slower in approaching the limits of safe operation, a similar mechanism can provide suggestive guidance. These ideas are formalized and elaborated throughout the remainder of this paper.

The starting point of our approach is a known model of the system dynamics and a defined *failure* region. The definition of what constitutes a failure is necessarily application-dependent; for flight control it could encompass the onset of a stall, while for a driving scenario it might include the wheels of the car leaving the road or beginning to lose traction. The *viability envelope* consists of all “points of no return,” past which the system will inevitably succumb to failure. It may be immediate or may come much later, but it is no longer possible to return to normal operation. Our method begins by building an explicit model of the viability envelope in a preprocessing step.

Given a model of the viability envelope, the online portion of the algorithm predicts a state trajectory assuming constant control input. When this predicted trajectory crosses, or *breaches* the viability envelope, a containment-preserving correction is applied. The temporal buffer effect of the look-

ahead allows choice in how the correction may be applied. One option is to simply override the user’s control input with a non-infringing one. Alternatively, one could employ haptic feedback to negotiate with the user, initially providing a guiding force which hints at better courses of action when a breach is distant, and then increasing the corrective forces as the breach draws near. Such “shared control” has been explored in [1].

Viability Theory [2]–[4] provides a theoretical framework for the class of problems being addressed here. Our work builds on this framework by developing algorithms for explicit approximations of the viability region and the online enforcement of it. In viability theory terms, our work provides a concrete class of viability kernels. Potential field methods can do well at solving some types of viability problems using empirically-defined buffer zones and enforcing local viability [5]. The use of virtual potential fields has also been explored for implementing a lane-keeping aid for car steering [6], [7] and can in some cases be shown to provide analytic guarantees with regard to system containment in the absence of human input. Our work addresses the problem of controllability in a more direct manner by using an explicit representation of the viability envelope.

Another related approach applies machine learning techniques to model the controllable regions of a set of given controllers [8]. However, this work only models whether the given control policy is capable of controlling the system from a given state, and not the more general question of the viability of a state with respect to the set of *all* feasible control policies, nor does it deal with the issue of when and how to apply corrective actions. More distantly related problems are those of computing reachable sets [9], and solutions to kinodynamic motion planning [10], [11].

II. FRAMEWORK

Consider a rocket that is constrained to fly vertically, with a user controlling the applied thrust. Wanting to reuse the rocket, the user would like to bring it down safely by performing a soft landing. This is complicated by the rocket having a limited amount of thrust and thereby requiring strict control of the downward velocity to ensure sufficient time to decelerate. The problem can be recast into control-theoretic terms: the task is to bring the rocket to the goal state, $(z, \frac{dz}{dt}) = (0, 0)$, while staying within the controllable region of state-space. We

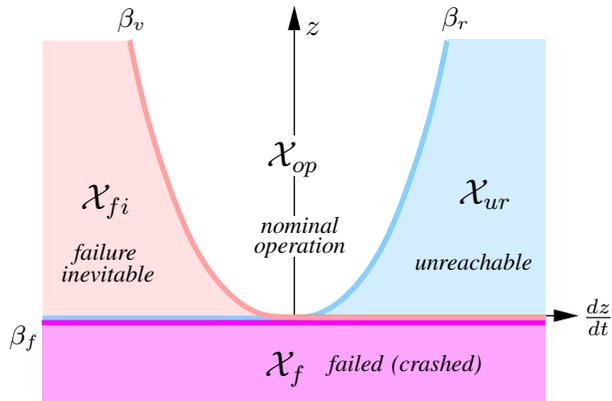


Fig. 1. Taxonomy of \mathcal{X} , the rocket’s state-space; z represents altitude. A safe landing requires bringing the system’s state to the origin while staying within \mathcal{X}_{op} , or at least within (i.e., above) the viability envelope β_v .

now look at how analytically-defined viability envelopes can be used to enforce safety in this example task.

The state-space for this problem, \mathcal{X} , can be partitioned into four regions using three boundaries, as shown in Fig. 1. β_f delimits \mathcal{X}_f , the failure region of state-space (i.e., where the rocket has crashed), while β_v and β_r mark the bounds of controllable (i.e., viable) and reachable space, respectively. β_v and β_r split $\mathcal{X} - \mathcal{X}_f$ into: \mathcal{X}_{fi} , the set of states for which failure is absent but inevitable, \mathcal{X}_{ur} , the set of controllable but unreachable states, and \mathcal{X}_{op} , the desired region of operation. In this example, the *viability envelope* β_v is defined by the “switching” curve for the corresponding bang-bang control problem. Similar partitions of \mathcal{X} can be made for any other dynamical system. Lastly, since we frequently refer to the controllable and uncontrollable regions, we assign them the symbols \mathcal{X}_{in} and \mathcal{X}_{out} , respectively (i.e., “in-” and “out-” side the β_v envelope). Thus $\mathcal{X}_{in} = \mathcal{X}_{op} \cup \mathcal{X}_{ur}$, and $\mathcal{X}_{out} = \mathcal{X}_f \cup \mathcal{X}_{fi}$. The task at hand then is to bring the system’s state to $(0, 0)$ while staying within \mathcal{X}_{in} .¹

A. Single-step containment

The simplest strategy for ensuring that the state stays within the envelope is to prevent an exit the instant it is about to occur. This can be achieved by giving the user full reign until such time, and then simply overriding the unsafe control input with a safe one. Within a discrete time framework, the control strategy can be formalized as

$$u_k = \begin{cases} v_k & \text{if } x_{k+1} = F(x_k, v_k) \in \mathcal{X}_{in} \\ N(x_k, v_k) & \text{otherwise} \end{cases} \quad (1)$$

where u_k is the control input applied at time step k , v_k is the control input the user requested, $x_k \in \mathcal{X}$ is the system’s state, F is a function that embodies the system dynamics in a discrete time setting, so that $x_{k+1} = F(x_k, u_k)$, and N is a function that returns an appropriate, safe control input. It should be noted that since $x_k \in \mathcal{X}_{in}$, at least one such safe control is guaranteed to exist. Also, since it is desirable

¹We constrain ourselves to \mathcal{X}_{in} rather than the more exclusive \mathcal{X}_{op} since later approximations and trade-offs will cause parts of \mathcal{X}_{ur} near β_r to, in fact, become reachable.

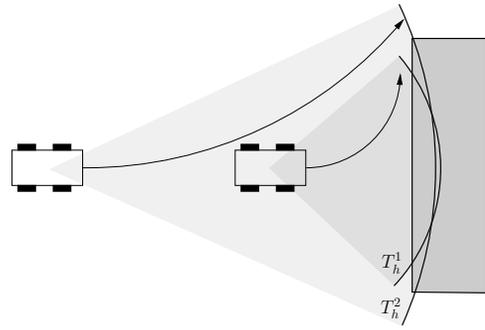


Fig. 2. A larger time horizon allows milder corrections. Above, the driver on the left has more space and time to maneuver to avoid the obstacle, and so is able to use a gentler turn. In general, using a larger horizon does not preclude the use of the shorter control policy, hence it is guaranteed, at the very least, to do no worse.

to limit the system’s intrusiveness, N should return the safe control input *nearest* to v_k .

B. Multi-step containment

The above method has one undesirable property: it produces severe corrections. For a user controlling the system, the experience of having control abruptly torn from their hands is likely to be disorienting and frustrating. This can be mitigated by responding earlier to upcoming breaches, by testing *multiple* subsequent time steps for a breach, instead of just one. As Fig. 2 illustrates, using a larger *time horizon* (T_h) generally leads to milder corrections. The desired magnitude of the horizon is clearly task- and system-dependent, but practical considerations impose an upper bound. For example, a car need not consider a turn in the road ahead if it is very far away. It is worth noting that, for the most part, the time horizon’s magnitude is driven by human response times and mental acuity, rather than by the complexity or particulars of the physical system being controlled.

In the multi-step approach, the user is given free reign until their control input leads to a breach within the time horizon, at which time the input is automatically supplanted by a safe alternative. The latter is chosen by extrapolating state trajectories for all constant-valued² control inputs available, up to T_h into the future, and selecting the best candidate. Although control is still usurped from the user, the effect is significantly milder. A bonus benefit of the multi-step approach is that the information on temporal breach proximity can be fortuitously used elsewhere, for example in haptic guidance, in mapping control inputs to meaningful feedback levels.

The concept of temporal proximity of a breach figures prominently in our work, so it is helpful to introduce a concise notation for it. Thus, let $T_{eb}(x, u)$ denote the *time to envelope breach* for a control input u , if the system is initially in state x . If u causes a breach within the time horizon, then $0 \leq T_{eb}(x, u) \leq T_h$; if not, it is convenient to let $T_{eb}(x, u) = +\infty$.

Using T_{eb} we can now fully characterize our online algorithm as follows. The method runs in one of four modes of

²This assumption embodies the *generalized inertia principle* from viability theory. In our method it serves as the best guess of user’s future input, a guess inherent in computing $T_{eb}(x, u)$.

operation, based on the assessed meta-state of the system:

- L1: $T_{eb}(x_k, v_k) > T_h$
- L2: $T_{eb}(x_k, v_k) \leq T_h$,
and $\exists u \in \mathcal{U} : T_{eb}(x_k, u) > T_h$
- L3: $\forall u \in \mathcal{U} : T_{eb}(x_k, u) \leq T_h$,
and $x_k \in \mathcal{X}_{in}$
- L4: $x_k \in \mathcal{X}_{out}$

The corresponding applied (corrected) control input is then given by

$$u_k = \begin{cases} v_k & \text{if L1} \\ B(x_k, v_k) & \text{if L2} \\ \arg \max_{u \in \mathcal{U}_{bf}} T_{eb}(x_k, u) & \text{if L3} \\ \text{N/A} & \text{if L4} \end{cases} \quad (2)$$

where \mathcal{U}_{bf} is a set-valued function which describes the set of constant-valued control inputs which are *breach-free* for a given state (i.e., $\mathcal{U}_{bf}(x_k) \subseteq \mathcal{U}$), and B is a function that picks an appropriate, safe control, biased in some way by the user's control input, v_k . It should be noted that whenever we refer to any control or trajectory as “breach-free”, we implicitly mean “... within T_h ”.

In brief, the four modes represent progressive levels of severity of the system meta-state. L1 and L2 constitute normal operation, while L3 and L4 correspond to crisis handling modes. In particular, L1 corresponds to the most benign case, where the user's control input v_k does not breach the envelope within T_h , and hence is applied as is. In L2 the user's input does lead to a breach, but other values exist that do not; an appropriate input is chosen from among these. In L3 all the control inputs lead to a breach. Since $x_k \in \mathcal{X}_{in}$, an achievable breach-free control policy is guaranteed to exist, but in this case it will involve time-varying control inputs. Choosing the control with the largest T_{eb} is likely to maximize the chances that the system will track, at least initially, one of these desired non-constant inputs, although this is not guaranteed. Finally, in L4, the system state is already outside the envelope. No control law is provided for this case as it does not occur with analytic envelopes; it is listed here for completeness, and plays a greater role in further sections.

III. PRACTICAL APPROXIMATIONS

The framework, as outlined above, is difficult to implement, especially in an interactive setting. This section presents the approximations which make this goal achievable.

A. Discretization of \mathcal{U}

A recurring problem throughout the framework is the need to search all of \mathcal{U} for some desired control input, or performing a computation on each of its members. A simple remedy is to discretize \mathcal{U} . We thus define $\hat{\mathcal{U}}$ as the set of controls uniformly sampled from \mathcal{U} , and use this subset wherever \mathcal{U} is called for. This then, for example, allows direct computation of the “nearest safe input” function N , which can now be formally defined as

$$N(x_k, v_k) = \arg \min_{u \in \hat{\mathcal{U}}_{bf}} |u - v_k| \quad (3)$$

where $\hat{\mathcal{U}}_{bf}(x_k) \subseteq \hat{\mathcal{U}}$ is the discretized equivalent of $\mathcal{U}_{bf}(x_k)$. One can now also easily establish the system's mode (i.e., $\in \{L1, L2, L3, L4\}$), since T_{eb} has to be computed only for a finite set of control inputs.

The size of $\hat{\mathcal{U}}$ is chosen to be as small as possible to reduce computational load, but large enough so that under most circumstances it captures at least one breach-free input. For simple systems (e.g., those amenable to bang-bang control) the discretization can be very sparse, since either the minimal or maximal input is frequently breach-free. For complex systems, on the other hand, even very dense discretizations can sometimes fail to produce a suitable candidate, especially if the controllable subset of \mathcal{U} is small and inconveniently distributed. This carries important repercussions, most principally that the L3 policy of setting u_k to the control with the largest T_{eb} , as shown in (2), likely becomes ineffective then, and admits breaches. The best one can do in this case is to treat the situation more severely, applying the L4 control law instead, which we discuss in the next subsection.

B. Approximate envelopes

Analytic descriptions of the viability envelopes can usually only be obtained for the simplest of systems. For more complex systems the envelope may be approximated through some form of empirical sampling of the state space, and the use of classification methods from machine learning to infer the controllability of the system for arbitrary query states. We have explored the use of both, Support Vector Machines and Nearest Neighbor techniques, and found the latter to be preferable for reasons of speed and algorithmic “transparency”, which make it amenable to application-specific customization and extension.

In general then, the envelope approximation is captured using a NN classifier

$$NN(x) = \begin{cases} 1 & \text{if } \min_{z \in \hat{\mathcal{X}}_{in}} |x - z| \leq \min_{z \in \hat{\mathcal{X}}_{out}} |x - z| \\ 0 & \text{otherwise} \end{cases} \quad (4)$$

where $\hat{\mathcal{X}}_{in}$ and $\hat{\mathcal{X}}_{out}$ are sets of samples which are known to be inside and outside the envelope, respectively. These sets are obtained in an offline pre-computation step, as detailed in the following section. Fig. 3 shows an example of a part of a viability envelope that was computed for a 2D dynamical system.

By virtue of being approximations, these envelopes will under- and over-approximate at various points bordering the true envelope, leading to false negatives ($NN(x) = 0$ when $x \in \mathcal{X}_{in}$) and false positives ($NN(x) = 1$ when $x \in \mathcal{X}_{out}$). The former is less of a problem than the latter since marking extra regions of state-space as uncontrollable merely results in a more conservative envelope. The key problem is the presence of the false positives, which can deceive the system into unknowingly entering \mathcal{X}_{out} . That is, the L4 mode now becomes a practical possibility.

If confronted with L4 and the unavailability of failure, we choose to minimize a metric of the failure's severity. This is

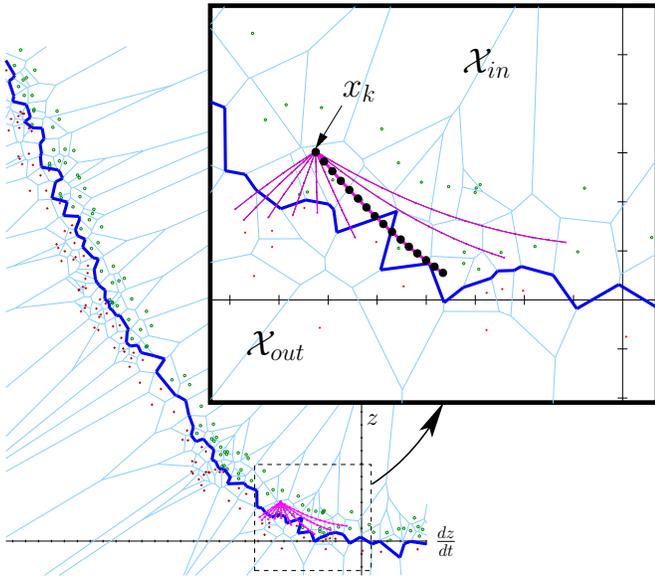


Fig. 3. A rocket’s 2D NN envelope; the set of trajectories originating at x_k show the computation of $T_{eb}(x_k, u) \forall u \in \tilde{U}$, with $T_{gr} = 1$. The user’s zero-thrust input (leftmost, not emphasized) is being overridden by the one shown in bold. Also shown are the band of samples adjoining the envelope, and the resultant Voronoi tessellation. (cf. the *true* envelope β_v in Fig. 1)

done by selecting the “least detrimental” control input, one which causes the system to spend the least amount of time in \mathcal{X}_{out} . Because the time spent in \mathcal{X}_{out} for any input may be arbitrarily large, the search for envelope re-entries needs to be bounded using a suitable criterion.

A complementary measure one may take is to use a conservative envelope, one which errs on the side of safety when placing the boundary. We have not yet explored any methods for doing this, but a straight-forward one would be to shrink the original envelope by a small percentage. The benefit of this is that any shallow breach of this envelope, such as given by the least-detrimental criterion above, will likely not incur a breach of the true envelope, thus maintaining system safety.

Finally, we employ a *grace period* when identifying envelope crossings, primarily to combat the error-induced noisy nature of NN envelopes. We define T_{gr} , the grace period, as the maximum amount of time that a trajectory may enter the alternate region without being labeled as a transition. Conversely, a transition is only pronounced if the trajectory excursion into the latter region lasts longer than T_{gr} . The rationale for this is that, as trajectories τ_1 and τ_2 of Fig. 4 suggest, the longer a trajectory stays within the latter region, the more likely it is that the perceived transition did in fact occur, and was not an artifact of the envelope representation. A trajectory such as τ_3 in Fig. 4 thus does not qualify as a transition according to this criterion.

IV. IMPLEMENTATION

A. Computation of $T_{eb}(x, u)$

We define $T_{eb}(x, u)$ in the discrete case as the time period to the first state that is classified as being in \mathcal{X}_{out} . Although measuring T_{eb} in seconds may seem natural, it is more

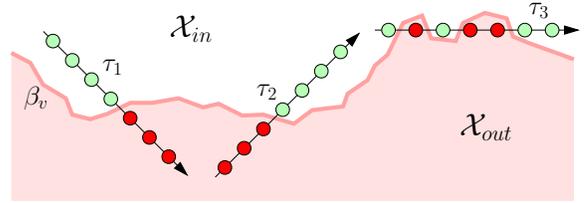


Fig. 4. Using a grace period to combat envelope (approximation) noise; for $T_{gr} = 2$, τ_1 forms a definite breach, τ_2 a definite re-entry, and τ_3 merely a “brushing” of the envelope (i.e., *not* a transition).

practical to express it as an integer number of fixed time steps Δt , with T_h and T_{gr} measured likewise. In computing T_{eb} there is usually no need to search for a breach past T_h ; it is always sufficient to know that $T_{eb} > T_h$ instead of the actual value (and thus our earlier use of $T_{eb} = +\infty$ in this case).

When searching for re-entries, we search up to an additional T_h into the future. If a re-entry is not found within that span, the next best thing is to select the control input whose state-space trajectory comes the closest. The relative proximity of the various trajectory endpoints can be effectively approximated as the average distance to the k nearest NN samples from \mathcal{X}_{in} , with $k = 3$ usually being sufficient; $k = 1$ tends to be unreliable.

B. Blending function

There are a number of ways to implement $B(x_k, v_k)$ that appears in (2). We have used a conservative approach, namely $B(x_k, v_k) = N(x_k, v_k)$. A more flexible and general approach is to implement it as a blending function

$$B_f(x_k, v_k) = \alpha v_k + (1 - \alpha)N(x_k, v_k) \quad (5)$$

where

$$\alpha = \min \left[\frac{T_{eb}(x_k, v_k) - 1}{T_h}, 1 \right] \quad (6)$$

This modulates the strength of the correction based on the immediacy of a breach, and thus allows the user more freedom at longer lead times. The approach gives corrections whose magnitudes vary between those of single-step containment and the $B = N$ case above.

C. Envelope construction

As mentioned earlier, we employ NN methods to classify query points based upon samples that are known to be viable or unviable. Algorithm 1 describes how the classified sample points are obtained, while (4) describes their application in the classifier.

The oracle (\vec{x}) is a function that authoritatively answers the question of whether the given state is controllable. It may obtain its knowledge by analytic, empirical, or heuristic means. The reason we do not consult the oracle directly during online simulation is that frequently these are extremely slow. The NN classifier essentially serves to embody the oracle’s knowledge in a form that is optimized for query speed, and serves as a universal encoding to which all other representations may be easily translated. Although the use of empirical data or heuristics can render the oracle’s accuracy to be imperfect,

Algorithm 1 Computation of $\hat{\mathcal{X}}_{in}, \hat{\mathcal{X}}_{out}$ for the NN classifier

```

 $\hat{\mathcal{X}}_{in}, \hat{\mathcal{X}}_{out} \leftarrow \{\emptyset\}$ 
for  $i = 1$  to  $n$  do
   $\vec{x} \leftarrow \text{rand\_uniform}(\mathcal{X})$ 
  if  $\text{oracle}(\vec{x}) = 1$  then
     $\hat{\mathcal{X}}_{in} \leftarrow \hat{\mathcal{X}}_{in} + \vec{x}$ 
  else
     $\hat{\mathcal{X}}_{out} \leftarrow \hat{\mathcal{X}}_{out} + \vec{x}$ 
   $\hat{\mathcal{X}}_{in}, \hat{\mathcal{X}}_{out} \leftarrow \text{scale\_samples}(\hat{\mathcal{X}}_{in}, \hat{\mathcal{X}}_{out})$ 
 $\hat{\mathcal{X}}_{in}, \hat{\mathcal{X}}_{out} \leftarrow \text{dump\_redundant}(\hat{\mathcal{X}}_{in}, \hat{\mathcal{X}}_{out})$ 

```

we have found that it is not a problem, as long as none of the errors are particularly egregious.

As with most learning methods, it is necessary to scale or normalize the training data prior to use, given that the NN classifier uses an L_2 norm distance metric in state-space. At present we select appropriate scaling factors manually, based on some understanding of the shape of the controllable region. The parameters should be chosen so that the significant features of the envelope surface (i.e., bumps, valleys, ridges, etc.) are of similar magnitude in all the state-space dimensions in which they lie, to avoid being trivialized and hidden in the noise or error inherent in the classifier’s representation of the surface.

A final measure taken to reduce unnecessary load is to discard redundant samples, ones which do not contribute to the NN decision surface, and consequently ones whose removal does not change it. Although a number of methods exist to do this [12], [13], we employ a simpler technique which trades-off thoroughness (i.e., does not drop every redundant sample) for large gains in speed. We make use of the fact that the samples are uniformly distributed, and compute the average inter-sample distance δ_s . We then discard all samples which are further than $k\delta_s$ from the decision surface³. The value of k is chosen by trying a number of possibilities, typically $k \in \{5, 10, 20, \dots\}$, and seek the first one that results in a consistent subset, namely one that properly classifies every sample from the original sets. This yields a well-structured band of samples around the decision surface.

V. RESULTS

We have successfully applied the viability envelope method to four systems: (1) the rocket, as discussed in section II; (2) a dynamical model of bicycle balance having a 2D state space $(\theta, d\theta/dt)$, where θ represents the tilt of the bicycle; (3) a steerable car restricted to an infinitely long straight road of limited width, having a 2D state space (y, θ) where y represents the distance of the car from one of the curbs, and θ represents the car’s orientation with respect to the road; and (4) a steerable car restricted to a terrain of arbitrary, bounded geometry. The last example is the most complex in terms of having a 3D state-space (x, y, θ) and no easily-modeled analytic solution. Due to space limitations, we restrict our results and discussions to this last example.

³This can be approximated by measuring instead the distance to the nearest NN sample of opposite class.

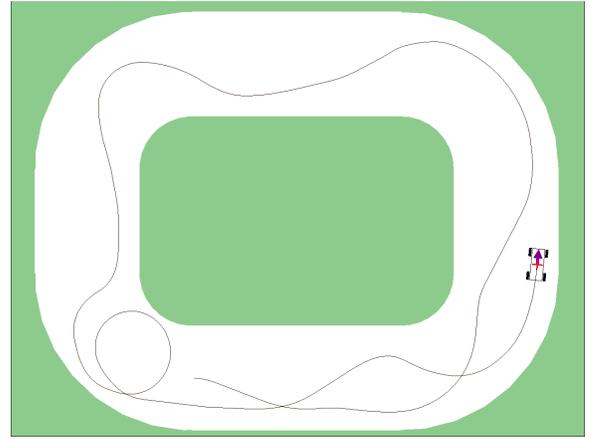


Fig. 5. A car constrained to stay on the track; see Fig.7 for plot of corresponding control inputs.

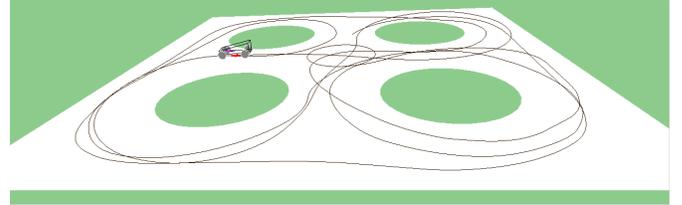


Fig. 6. Less conventional terrain for the car (perspective view).

Fig. 5 shows a trial run for a car on a track using a 3D viability envelope. The user is able to interactively steer the car at will but is prevented by the system from leaving the track. Fig. 7 shows how the safety constraints project onto the control input space for this problem. The user input v_k consists of a sequence of right and left turns of the steering wheel, as represented by the smooth line on the graph. The unviable control inputs are given by the shaded areas. The applied control input, u_k is computed as given by Equation 2 and is represented by the line taking discrete steps, reflecting the discretization of the control input space. Lastly, Fig. 8 shows the set of viable and unviable steering directions for various states of the car during a simulation.

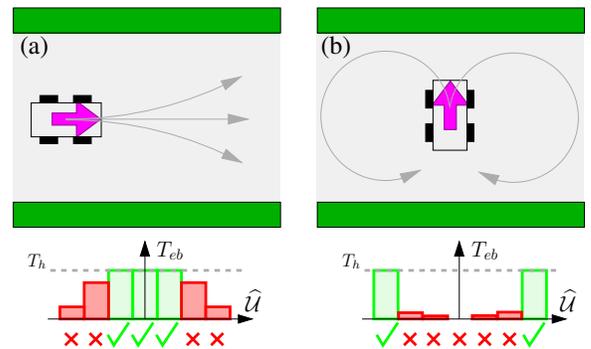


Fig. 8. T_{eb} behavior and viability of $\hat{\mathcal{U}}$ for a fixed-velocity car; the gray trajectories correspond to breach-free controls, which appear checkmarked and green in the T_{eb} vs. $\hat{\mathcal{U}}$ bar graphs underneath. Case (b) is instructive: although the car’s distance from the opposing curb suggests some leeway, the car is in fact nearly upon the point of no return (i.e., envelope), as hinted by the many $T_{eb} \approx 0$, and must immediately choose one of the breach-free inputs.

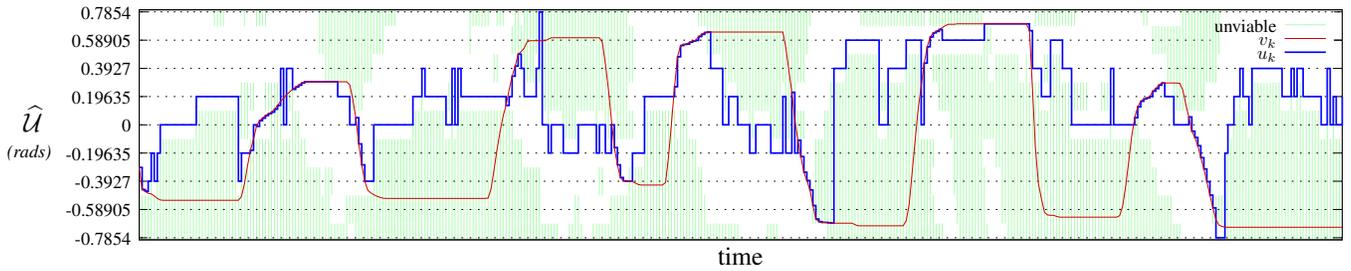


Fig. 7. Plot of v_k , u_k , and the viability of $u \in \hat{\mathcal{U}}$ for the simulation run shown in Fig. 5. u is the deflection of the front wheels, in radians.

TABLE I
SYSTEM PARAMETERS PER SCENARIO

scenario	# samples	$ \hat{\mathcal{U}} $	T_h	T_{gr}
rocket	1,218	9	15	1
bike	3,513	9	10	1
car (straight road)	N/A	9	10	0
car (track)	42,117	31	10	1
car (4 obstacles)	176,545	15	10	1

Our method runs interactively on a 2.4GHz Pentium IV. We target a 30Hz interactive simulation rate to give the user reasonable responsiveness. Table I lists the various parameters used for the scenarios: number of NN samples⁴, discretization of control space, time horizon, and grace period. The latter two are expressed in terms of number of simulation steps (i.e., $\frac{1}{30}$ s). It should be also noted that the original number of NN samples for each case is much larger; the reported number is that of the remaining set when redundant samples have been removed by `dump_redundant` ($\hat{\mathcal{X}}_{in}$, $\hat{\mathcal{X}}_{out}$).

For the case of the car on the straight roadway we have also implemented haptic feedback using a *Phantom* device through which the user steers the car. Preliminary results are promising, and we hope to investigate this extension further.

VI. DISCUSSION

There is a necessary compromise that must be struck between control flexibility and smoothness. By giving the user more flexibility in control when a breach is still relatively distant (e.g., with the blending function), allowing him or her to “push into the envelope”, the system potentially incurs a larger correction later on when the envelope is approached more aggressively as a result.

The worst-case time complexity of the online algorithm is $O(|\hat{\mathcal{U}}|T_h)$, where the system is applying least-detrimental control selection in L3 or L4, and thus must simulate and inspect $|\hat{\mathcal{U}}|$ trajectories, each consisting of $2T_h$ time steps (search for re-entries). This worst case also holds for L2, where all control inputs tested, other than v_k , are breach-free. L1 time complexity is always $O(1)$, the constant time it takes to simulate $x_{k+1} = F(x_k, v_k)$ and establish its presence in \mathcal{X}_{in} .

Recent tests on higher dimensional systems suggest that the NN representation becomes intractable due to exponential growth in samples needed. Thus there is a need to explore other, more scalable representations.

⁴In the case of a car on a straight road, which was the first we experimented with, a hand-generated polygonal envelope was used instead of a NN classifier.

VII. CONCLUSION

In this paper we have presented a method of enforcing the controllability of a user-steered system, using an explicitly computed approximation of the viability envelope. We have also detailed an implementation and applied it to the motion of a number of simple vehicles.

In future work we intend to apply the method to more complex systems, where complexity implies both more discontinuous dynamics as well as systems with higher dimensional state-spaces. In a related line of inquiry, we plan to look into working with multi-dimensional control input spaces. A key question to answer here is how to distribute any corrections among the control parameters. We also hope to revisit our original motivation for this work and further investigate how haptic feedback can be applied to yield a more effective system-user interaction by better communicating imminent corrections to a user.

REFERENCES

- [1] M. Steele and R. B. Gillespie, “Shared control between human and machine: Using a haptic steering wheel to aid in land vehicle guidance,” *Human Factors and Ergonomics Society 45th Annual Meeting*, October 2001.
- [2] J.-P. Aubin and A. Cellina, *Differential Inclusions*. Springer-Verlag, 1984.
- [3] J.-P. Aubin, *Viability Theory*, ser. Systems & Control: Foundations & Applications, C. I. Byrnes, Ed. Birkhäuser, 1991.
- [4] —, “A survey of viability theory,” *SIAM J. of control and optimization*, vol. 28, no. 4, pp. 749–788, July 1990.
- [5] R. J. Spiteri, D. K. Pai, and U. M. Ascher, “Programming and control of robots by means of differential algebraic inequalities,” *IEEE Trans. on Robotics and Automation*, vol. 16, no. 2, pp. 135–145, April 2000.
- [6] E. J. Rossetter, “A potential field framework for active vehicle lanekeeping assistance,” Ph.D. dissertation, Stanford University, August 2003.
- [7] E. J. Rossetter, J. P. Switkes, and J. C. Gerdes, “A gentle nudge towards safety: Experimental validation of the potential field driver assistance system,” in *Proceedings of the American Control Conference*, 2003.
- [8] P. Faloutsos, M. van de Panne, and D. Terzopoulos, “Autonomous reactive control for simulated humanoids,” in *IEEE International Conference on Robotics and Automation*, 2003.
- [9] I. Mitchell, “Application of level set methods to control and reachability problems in continuous and hybrid systems,” Ph.D. dissertation, Stanford University, 2002.
- [10] E. Frazzoli, M. A. Dahleh, and E. Feron, “Real-time motion planning for agile autonomous vehicles,” *AIAA Journal of Guidance, Control, and Dynamics*, vol. 25, no. 1, pp. 116–129, 2002.
- [11] B. Donald, P. Xavier, J. Canny, and J. Reif, “Kinodynamic motion planning,” *Journal of ACM*, vol. 40, no. 5, pp. 1048–1066, Nov 1993.
- [12] P.E.Hart, “The condensed nearest neighbor rule,” *IEEE Trans. Inform. Theory*, vol. IT-14, no. 3, pp. 515–516, May 1968.
- [13] B. V. Dasarthy, *Nearest Neighbor(NN) norms: NN pattern classification techniques*. IEEE Computer Society Press, 1991.